Detection of Phishing URL

Mihret Kemal
Tagore Kosireddy
Saket Pawar

MA 5790
December 2024

Abstract

Phishing attacks, which aim to steal sensitive personal information through fraudulent websites, continue to pose a significant threat to internet security. According to Forbes, over 500 million phishing attacks were reported globally in just a single year, highlighting the urgent need for effective detection systems. The primary goal of this study is developing a predictive model for classifying URLs as either legitimate or phishing. This study will also explore the relationships between various features extracted from the URLs and their classification, along with the separation of degenerate variables from the predictor set. Both linear and nonlinear models will be fit to the training data using cross-validation methods, and the most promising model will be used to predict phishing URLs on a test set. The overall best model will be selected based on performance metrics, with the aim of providing an effective tool for detecting phishing attacks and enhancing internet security.

Table of Contents

1 Background

Phishing URLs are malicious web links designed to deceive users into visiting fraudulent websites. These sites aim to steal sensitive information such as login credentials, financial data, or personal details. Often crafted to mimic legitimate URLs, phishing links exploit user trust and are distributed via emails, social media platforms, or messaging services. Their ability to blend into everyday digital interactions makes phishing URLs a persistent and evolving threat in cyberspace.

In fact, Forbes reported that over 500 million phishing attacks occurred in 2022, highlighting the growing scale of this cybersecurity challenge.The detection of phishing URLs relies on analyzing various features that distinguish them from legitimate links. For example, the URL length and domain length can indicate suspicious behavior, as phishing URLs often use unusually long strings to obfuscate their intent. By identifying these patterns, it is possible to mitigate the risks associated with phishing attacks.

Machine learning has emerged as a powerful tool for leveraging features to detect phishing URLs. By analyzing patterns such as those mentioned above, machine learning models can classify URLs with high accuracy. In this report we will try building different machine learning models for detection of phishing URLs from the legitimate one. As phishing attacks grow more sophisticated, developing predictive systems that use these features is critical to enhancing cybersecurity and protecting users from online threats.

2 Introduction

This project addresses a critical cybersecurity challenge: phishing attacks, by focusing on the accurate detection of phishing URLs. **The primary goal of this project is to develop and evaluate various linear and nonlinear machine learning models to classify URLs as either legitimate or phishing.** This classification relies on analyzing diverse features of URLs, enabling a more robust approach to identifying and mitigating phishing threats.

2.1 Variable Introduction and Definitions

The data used is the PhiUSIIL Phishing URL Dataset, sourced from the UCI Machine Learning Repository, consisting of 235,795 samples. This dataset is designed for binary classification, where the response variable is labeled as either 1 (Legitimate URL) or 0 (Phishing URL). Of the 56 columns in the dataset, four (Filename, URL, Domain, and Title) provide extra information about the URLs and are excluded from model building, leaving 51 predictors. These predictors comprise 33 continuous variables and 18 categorical variables, offering a diverse set of features for robust analysis and machine learning. Below is a list of the variable names as used in this analysis, with their descriptions.

Variable Name	Description
URLLength	The length of the URL
DomainLength	The length of the domain name
IsDomainIP	Indicates if the domain is an IP address
TLD	The top-level domain (e.g., "com", "de", "uk")
URLSimilarityIndex	The similarity index of the URL
CharContinuationRate	The ratio of consecutive letters in the URL
TLDLegitimateProb	The probability that the TLD is legitimate
URLCharProb	The probability of characters in the URL being legitimate
TLDLength	The length of the TLD
NoOfSubDomain	The number of subdomains in the URL
HasObfuscation	Indicates if the URL contains obfuscation
NoOfObfuscatedChar	The number of obfuscated characters in the URL
ObfuscationRatio	The ratio of obfuscated characters in the URL
NoOfLettersInURL	The number of letters in the URL
LetterRatioInURL	The ratio of letters in the URL
NoOfDegitsInURL	The number of digits in the URL
DegitRatioInURL	The ratio of digits in the URL
NoOfEqualsInURL	The number of "=" symbols in the URL
NoOfQMarkInURL	The number of "?" symbols in the URL
NoOfAmpersandInURL	The number of "&" symbols in the URL
NoOfOtherSpecialCharsInURL	The number of other special characters in the URL
SpacialCharRatioInURL	The ratio of special characters in the URL
IsHTTPS	Indicates if the URL uses HTTPS
LineOfCode	The number of lines of code in the webpage
LargestLineLength	The length of the largest line in the webpage
HasTitle	Indicates if the webpage has a title
DomainTitleMatchScore	The match score between the domain and the webpage title
URLTitleMatchScore	The match score between the URL and the webpage title
HasFavicon	Indicates if the webpage has a favicon

Robots	Indicates if the webpage has a robots.txt file
IsResponsive	Indicates if the webpage is responsive
NoOfURLRedirect	The number of URL redirects
NoOfSelfRedirect	The number of self redirects
HasDescription	Indicates if the webpage has a meta description
NoOfPopup	The number of popups in the webpage
NoOfiFrame	The number of iframes in the webpage
HasExternalFormSubmit	Indicates if the webpage has an external form submit
HasSocialNet	Indicates if the webpage contains social network links
HasSubmitButton	Indicates if the webpage has a submit button
HasHiddenFields	Indicates if the webpage has hidden fields
HasPasswordField	Indicates if the webpage has a password field
Bank	Indicates if the webpage is related to banking
Pay	Indicates if the webpage is related to payments
Crypto	Indicates if the webpage is related to cryptocurrency
HasCopyrightInfo	Indicates if the webpage contains copyright information
NoOfImage	The number of images on the webpage
NoOfCSS	The number of CSS files used in the webpage
NoOfJS	The number of JavaScript files used in the webpage
NoOfSelfRef	The number of self-references in the webpage
NoOfEmptyRef	The number of empty references in the webpage
NoOfExternalRef	The number of external references in the webpage

3 Preprocessing of the predictors

In the preprocessing stage of the dataset, it was ensured that there were **no missing values**, so no imputation was necessary. Additionally, four dummy variables were added based on the TLD (Top-Level Domain) predictor to enhance the model's ability to differentiate between various domain types. These dummy variables are as follows: Internationalized_domain, indicating whether the domain includes internationalized characters; Country_domain, which identifies if the domain corresponds to a country-specific TLD (e.g., ".uk", ".de"); Common_domain, flagging if the domain is from a commonly used TLD (e.g., ".com", ".net"); and Number_domain, a variable representing whether the domain contains numeric characters in the

TLD. Furthermore, all categorical predictors in the dataset were encoded into binary form (0 and 1) to allow the machine learning model to process these variables effectively.

a. Near Zero Variance Predictors

Six predictors with near-zero variance were identified and removed namely: IsDomainIP, HasObfuscation, HasExternalFormSubmit, Crypto, Internationalized_Domain, and Numeric_Domain. These predictors were deleted as they provided little to no variance and did not contribute to the model's predictive power. The bar plot (Figure 1) shows the frequency distribution of the near zero variance predictors.

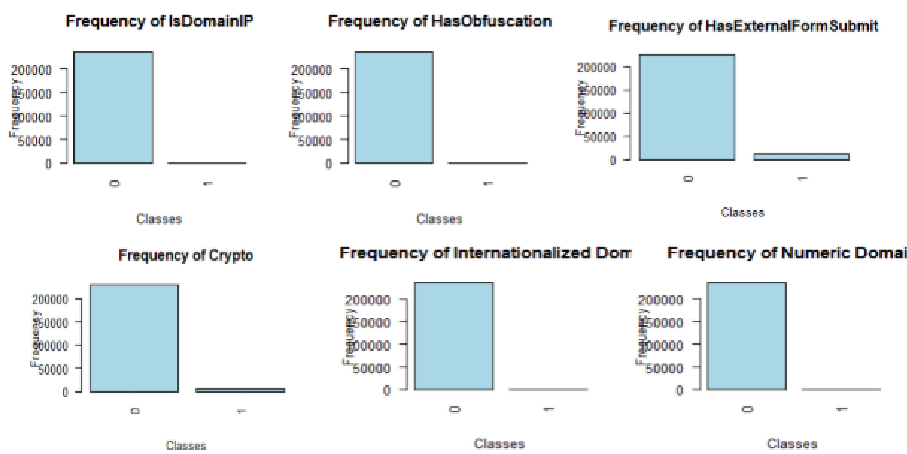


Figure 1: Frequency distribution of near zero variance predictors

b. Correlations

A correlation plot of the remaining predictors was created to explore the relationship between predictors. Figure 2 shows the correlation plot with the blue representing positive correlations between predictors and red representing negative correlations between predictors. Predictors with greater than 0.75 correlations were then systematically removed. We identified five such predictors and removed them from the dataset. The predictors deleted were DomainTitleMatchScore, NoOfOtherSpecialCharsInURL, URLLength, NoOfDegitsInURL, and NoOfObfuscatedChar. These columns were removed to reduce multicollinearity and improve the model's performance by ensuring that only independent predictors are used. Note that we will not do this step for all the models, only specific models require the removal of highly correlated predictors.

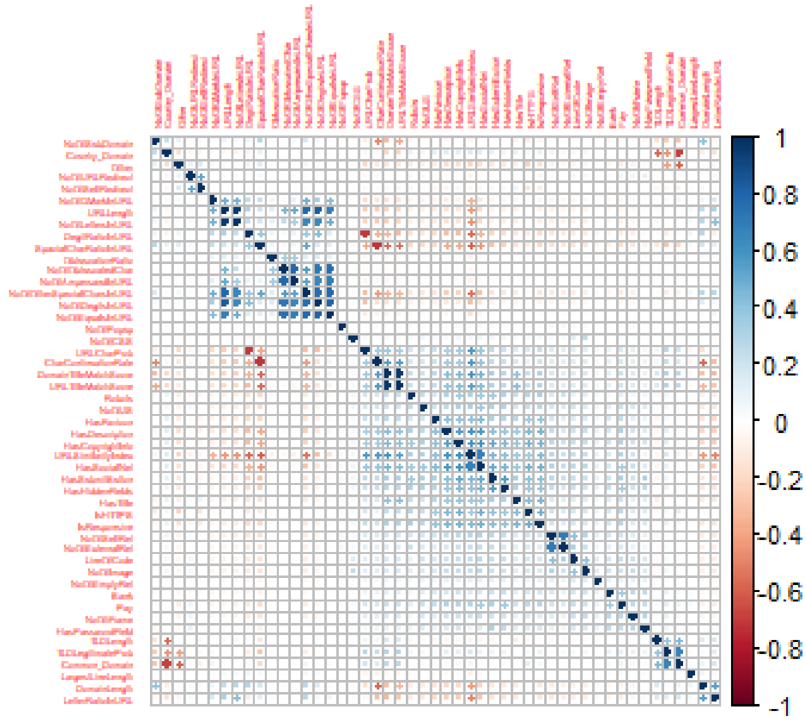


Figure 2: Correlation plot for predictors

c. Transformations

Before fitting any models to the data, it is crucial to analyze the distributions of the continuous variables. Figure 3 and 4 shows histograms and box plots of some of the continuous predictors before any transformations have been applied. As observed, most predictors exhibit a strong right-skewed distribution and contain several outliers.

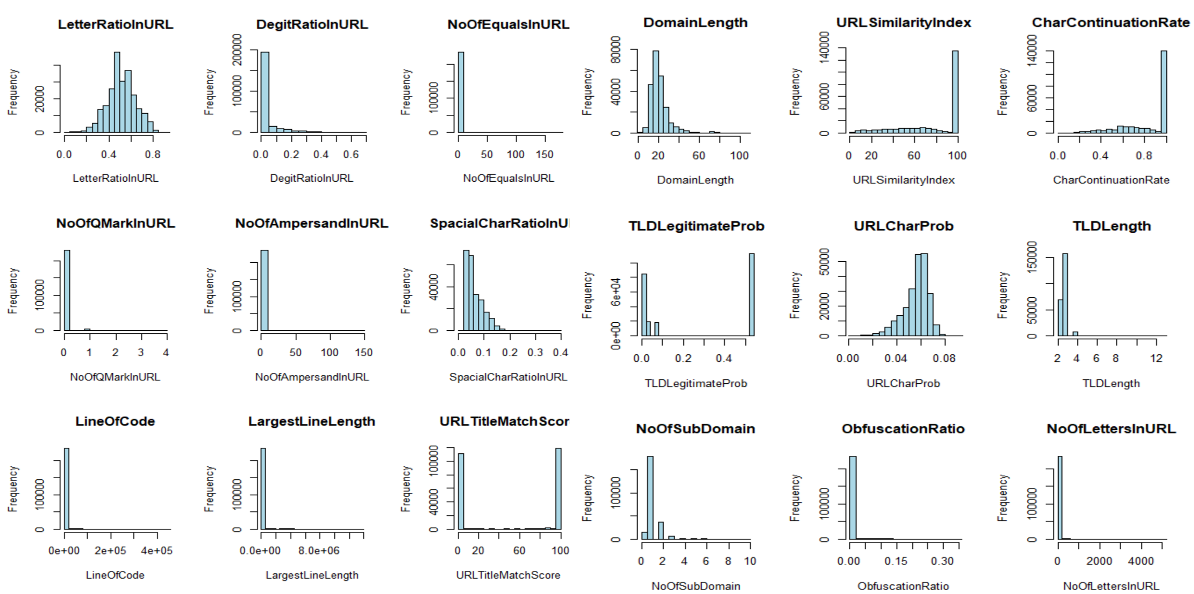


Figure 3: Histogram to check distribution

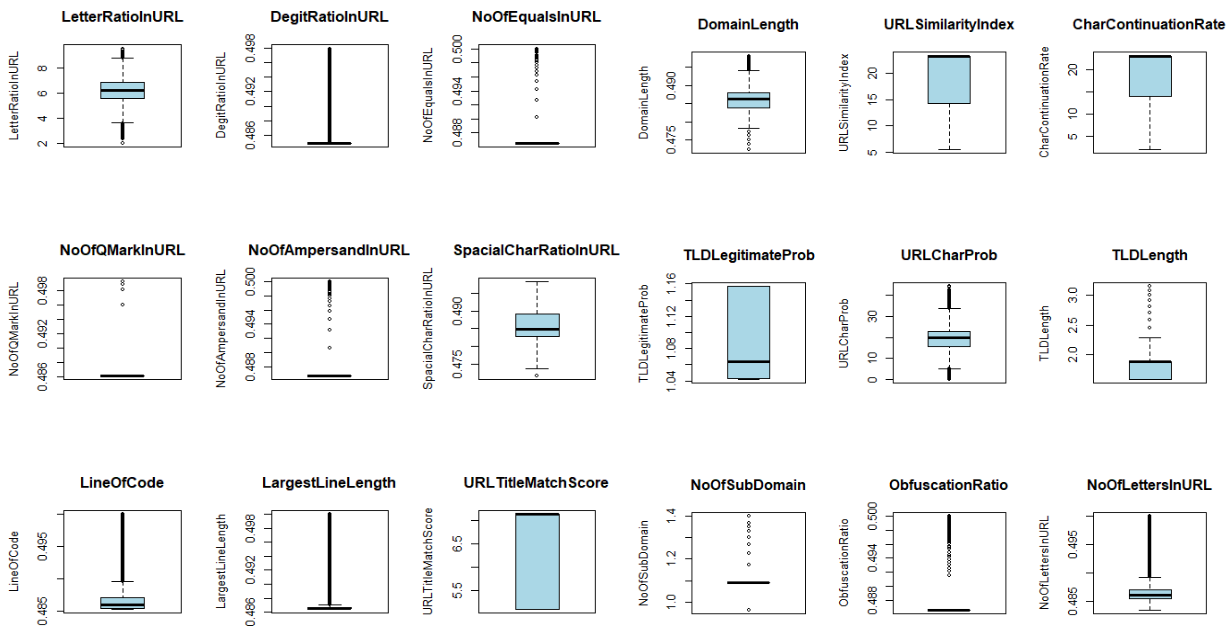


Figure 4: Boxplot for checking outliers

To address the skewness, outliers, and varying scales of the continuous variables, the data will first undergo centering and scaling. Subsequently, a Box-Cox transformation was applied to make the distributions more normal and symmetric. This was followed by a spatial sign transformation to mitigate the impact of outliers. Below(Figure 5 and 6) are histograms and box plots illustrating the data after these transformations. While not all predictors showed significant improvement, some displayed better symmetry and reduced outliers.

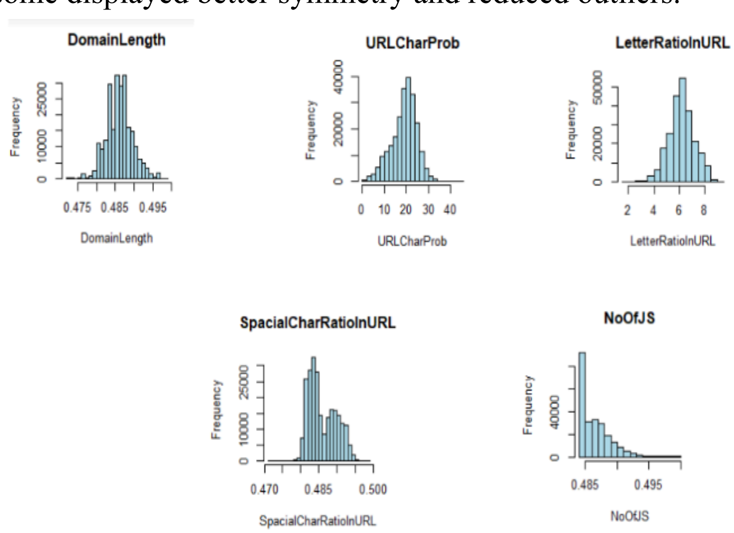


Figure 5: Distribution after box cox transformation

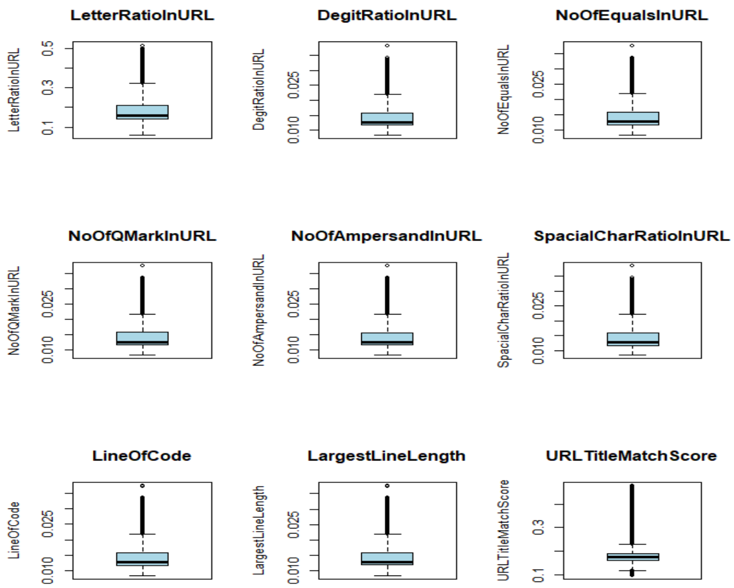


Figure 6: Boxplot after spatial sign transformation

Coming to categorical predictor exploration, the bar plots below (Figure 7) illustrate the distributions of the categorical predictor variables. It is evident that most of these variables exhibit relatively imbalanced frequencies.

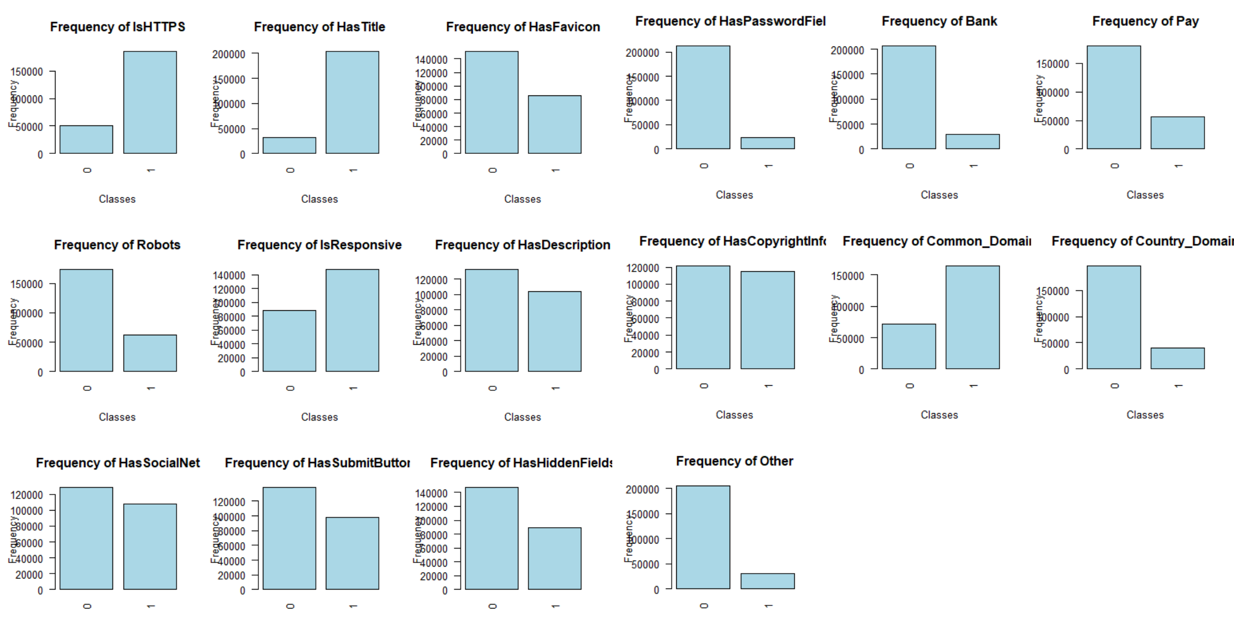


Figure 7: Bar Graph for the categorical predictors

After preprocessing, we also explored applying Principal Component Analysis (PCA) and identified 16 principal components (PCs) that captured 95% of the variance in the data. However, PCA was not used for model building as it can result in the loss of interpretability, which is critical for understanding the contributions of individual predictors in the context of phishing URL detection.

4 Splitting of the Data

The target variable frequency distribution is not perfectly balanced, as shown in the bar graph below, with an approximate ratio of 57:43. To ensure representative sampling, the data was split into training and testing sets using an 80/20 split with stratified random sampling. This approach preserves the class distribution within both subsets, which is crucial given the imbalance. For resampling during model evaluation, K-fold cross-validation with K=3 was employed. This choice was made due to the large dataset size (over 200k samples), as a 10-fold cross-validation would be time-consuming.

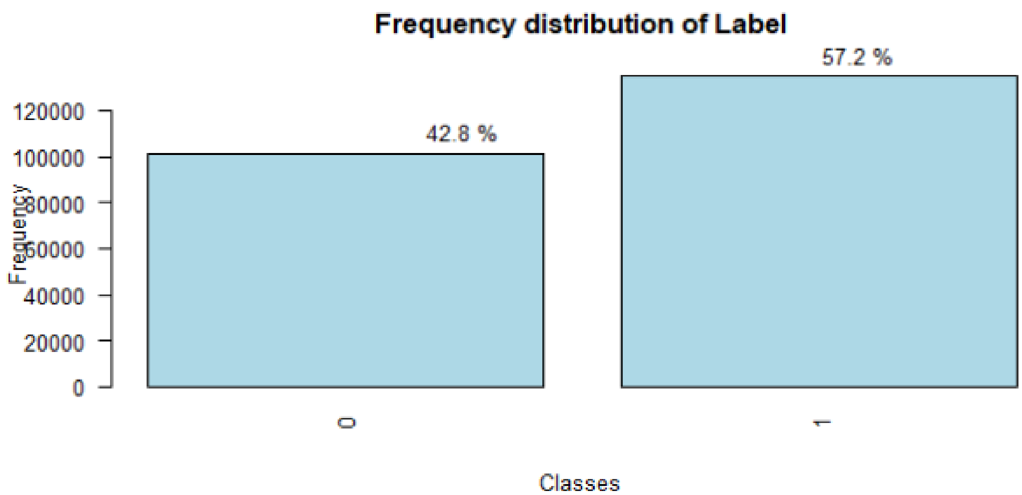


Figure 8: Frequency distribution of response class

5 Model Fitting

We applied a variety of linear and non-linear models for classification as discussed in class. Given that our dataset is not perfectly balanced, **we used Kappa as the classification performance metric**. Kappa is a valuable metric in this context as it adjusts for class imbalance, providing a more accurate evaluation of model performance by considering the agreement between predicted and actual class labels beyond what would be expected by chance.

A. Linear Models

The linear models included Logistic Regression, Linear Discriminant Analysis (LDA), Partial Least Squares Discriminant Analysis (PLSDA), and penalized models. These models were tuned using the training data, and both accuracy and Kappa values were calculated using 3-fold cross-validation to ensure the most conservative and robust results. The table below presents the training and testing Kappa and accuracy values for each of the linear models.

Model	Best Tuning parameter	Training Kappa	Training Accuracy	Testing Kappa	Testing Accuracy
Logistic regression	-	0.7333645	0.8704171	0.7522	0.8828
LDA	-	0.7218282	0.8689752	0.7191	0.8677
PLSDA	ncomp = 4	0.7304395	0.872818	0.7238	0.8722
Penalized model	alpha = 0, lambda = 0.05	0.8079254	0.9084533	0.8059	0.9075

Among the linear models, the penalized model emerged as the best performer, with optimal hyperparameters set to alpha = 0 and lambda = 0.05. This model demonstrated the highest performance in terms of Kappa and accuracy, outperforming the other linear models.

The table below shows the confusion matrix for the Penalized model when predicting on the test set.

	Reference	
Prediction	Legitimate	Phishing
Legitimate	26959	4350
Phishing	11	15839

The model successfully identified 26,959 legitimate URLs and 15,839 phishing URLs, showcasing its strong capability in both categories. However, it also misclassified 4,350 legitimate URLs as phishing (false positives) and failed to detect 11 phishing URLs, labeling them as legitimate (false negatives). The low false-negative count is particularly noteworthy, as missed phishing URLs pose significant security risks. On the other hand, the relatively high false-positive count indicates a trade-off between the model's sensitivity in detecting phishing threats and its specificity in avoiding unnecessary false alarms.

B. Nonlinear Models

For the non-linear models, we utilized Regularized Discriminant Analysis (RDA), Mixture Discriminant Analysis (MDA), Neural Networks, Flexible Discriminant Analysis (FDA), Support Vector Machines (SVM), K-Nearest Neighbors (KNN), and Naive Bayes. Each model was carefully tuned and evaluated to assess its performance on the dataset.

While attempting to include Quadratic Discriminant Analysis (QDA) among the non-linear models, we encountered a limitation: the covariance matrix for the classes did not exist, rendering the implementation of QDA infeasible. However, all other non-linear models were successfully built and tested.

The table below presents the training and testing Kappa and accuracy values for each of the non linear models.

Model	Best Tuning parameter	Training Kappa	Training Accuracy	Testing Kappa	Testing Accuracy
RDA	gamma = 0 and lambda = 0.01	0.7551	0.88392	0.7538	0.8834
MDA	subclasses=19	0.93025	0.96614	0.9188	0.9607
Neural Network	size = 3 and decay = 0	0.99921	0.99902	0.9989	0.9994
FDA	degree = 2 and nprune = 30	0.9970854	0.9956371	0.9986	0.9959
SVM	cost=64, sigma = 0.004992348	0.99923	0.999194	0.9991	0.9996
KNN	k = 3	0.991215	0.99570	0.993	0.9966
Naive Bayes	-	0.8571713	0.9303155	0.8615	0.9325

Among all the linear and non-linear models evaluated, Support Vector Machines (SVM) and Neural Networks emerged as the top 2 best performing models. The SVM model achieved the highest testing Kappa value of 0.9991 with optimal hyperparameters: a cost of 64 and a sigma value of 0.00499. While the Neural Network model achieved a testing Kappa value of 0.9989 with a network size of 3 and a decay parameter of 0.

Since the SVM model is the best model, we draw the confusion matrix table for it when predicting on the test set.

	Reference	
Prediction	Legitimate	Phishing
Legitimate	26962	12
Phishing	8	20177

The model correctly classified 26,962 legitimate URLs and 20,177 phishing URLs, demonstrating high accuracy in both categories. It misclassified only 12 legitimate URLs as phishing (false positives) and failed to detect 8 phishing URLs, labeling them as legitimate (false negatives). This performance highlights the model's excellent balance between sensitivity and specificity, with an impressively low rate of misclassifications. The SVM model's ability to accurately distinguish legitimate from phishing URLs makes it a highly reliable tool for mitigating phishing threats.

C. Top 10 most important predictors

To identify the most influential predictors in our best-performing model, the Support Vector Machine (SVM), we analyzed variable importance and highlighted the top10 contributors.

Only 10 most important variables shown (out of 49)

	Importance
URLSimilarityIndex	100.00
LineOfCode	98.82
NoOfExternalRef	98.40
NoOfImage	96.69
NoOfSelfRef	95.18
NoOfJS	94.92
NoOfCSS	92.25

HasSocialNet	79.50
HasCopyrightInfo	75.59
HasDescription	69.73

Among the top predictors, the URL Similarity Index emerged as the most critical, with an importance score of 100. This variable measures how closely a URL resembles known legitimate or malicious patterns, making it a key factor in distinguishing phishing URLs. The Line of Code variable, with an importance score of 98.82, also played a significant role; it reflects the number of lines in a webpage's code, where larger or more complex codebases can signal legitimate or malicious activities. Additionally, the Number of External References (importance score of 98.40) provided valuable insight by tracking the number of external links on a webpage, which could indicate phishing attempts when links redirect users to malicious sites. The Figure 9 below shows 10 most important predictors according to our best model SVM.

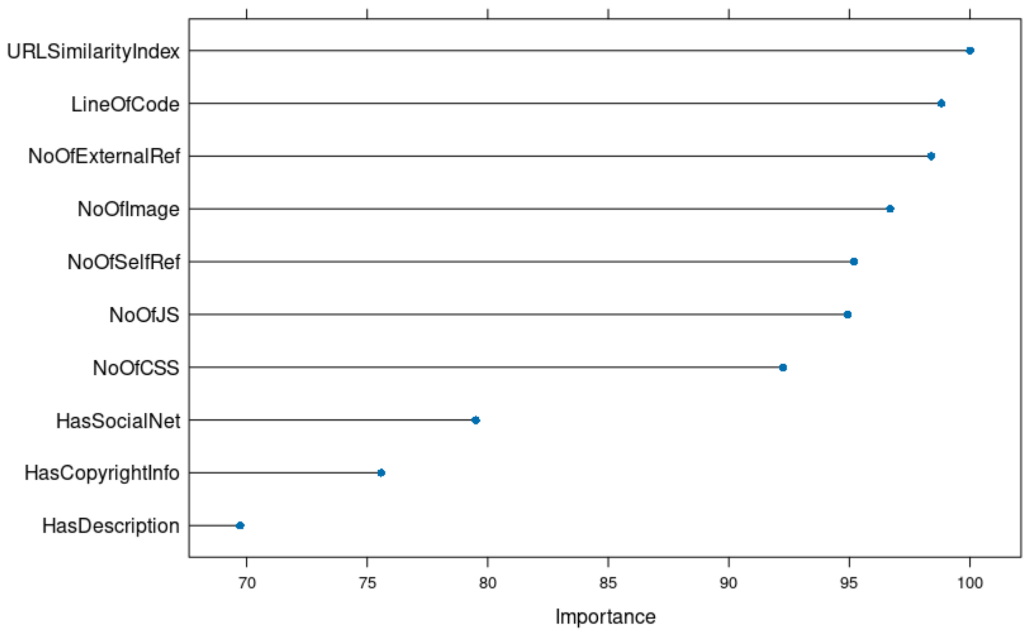


Figure 9: Top 10 most important predictors in SVM

6 Conclusion

In conclusion, our project aimed to identify the best model for detecting whether a URL is phishing or legitimate. Starting with extensive preprocessing, we ensured the data was well-prepared for modeling by addressing skewness, scaling, outliers, and multicollinearity, as well as encoding categorical predictors and handling class imbalance. We explored a wide range of linear and non-linear models, evaluating their performance using kappa and accuracy metrics.

The Support Vector Machine (SVM) model emerged as the best-performing model with a testing kappa score of 0.9991, followed by the Neural Network model with a kappa of 0.9989. Feature importance analysis of the SVM model highlighted key predictors such as the URL Similarity Index, Number of Lines of Code, and Number of External References. These findings demonstrate the robustness of our approach in achieving near-perfect classification accuracy, providing an effective solution for detecting phishing URLs.

7 Literature Review and Comparison

We wanted to know how well we did by comparing our methods and results to other people's methods and results. From (Vajrobol, Gupta, and Gaurav 2024) they got Accuracy with 99.97% by using logistic regression with 5 features which were selected on feature selection based on mutual information. (Alsharaiah et al.2023) got an Accuracy of 98.64% with a framework based on random forest ensemble techniques. (Biswas et al. 2024) got an Accuracy of 94.612 with a hybrid framework. (Usman, and Fong 2023) got an Accuracy of 96.25% with an effective machine learning framework. (Prasad and Chandra 2024) got an Accuracy of 99.24% with USI technique that is proposed that detects visual similarity-based phishing attacks. Gupta et al. (2021) focused solely on URL features and employed a Random Forest algorithm, achieving an impressive accuracy of 99.57% on a dataset of 11,964 records. Pandey and Mishra (2023) explored dominant color features and OCR, achieving an accuracy of 99.13% using a Random Forest algorithm on 6,200 records. Ahammad et al. (2022) employed URL features combined with models like Random Forest, LightGBM, Logistic Regression, and SVM, but their accuracy was limited to 89.5%, likely due to a small dataset of 3,000 records.

Jain et al. (2022) utilized static and site popularity features with algorithms like Logistic Regression, KNN, SVM, and Random Forest, achieving an accuracy of 93.85% on 4,000 records, but their reliance on third-party features limited their performance. Similarly, Alani and Tawfik (2022) used URL and third-party features, achieving an accuracy of 97.5% with multiple machine learning models such as Random Forest, Logistic Regression, and MLP, but their results were constrained by small datasets and reliance on third-party data. Ding et al. (2019) incorporated URL, HTML, and third-party features with Logistic Regression, achieving 98.9% accuracy on 8,659 records.

Sharma and Singh (2022) leveraged webpage HTML code features with TF-IDF and AdaBoost, achieving an accuracy of 98.01% on a dataset of 50,000 records, though their approach depended on a single machine learning model. Nagumwa et al. (2022) combined features derived from DNS, host, and network, using eight machine learning and three deep learning algorithms to achieve 98.42% accuracy on 11,801 records, but their approach was computationally expensive. Sameen et al. (2020) used URL features and a boosting-based approach, achieving 98% accuracy on a large dataset of 100,000 records, though their framework required computationally intensive models. Finally, Rao et al. (2022) utilized HTML code and domain-specific features, employing

Random Forest, SVM, and XGBoost models to achieve 99.34% accuracy on 10,514 records, although their approach was limited by the size of the dataset.

A. Prasad and S. Chandra *Computers & Security 136 (2024) 103545*

Table 1
Comparative summary of related work with proposed work.

Ref.	Features	Detection model	Dataset records	Accu-racy	Limitations
Gupta et al. (2021)	URL features only	Depends on Random forest algorithm	11964	99.57%	Depends solely on URL features, and on single ML algorithm, experimented on small dataset
Pandey and Mishra (2023)	Dominant color features and OCR	Depends on Random forest algorithm	6200	99.13%	Depends solely on single ML algorithm, experimented on small dataset
Ahammad et al. (2022)	URL Features only	RF, DT, Light GBM, LR, and SVM	3000	89.50%	Low prediction performances, experimented on small dataset
Jain et al. (2022)	Static and site popularity features	LR, KNN, SVM, DT, and RF	4000	93.85%	Depends on third party features, low prediction performances, experimented on small dataset
Alani and Tawfik (2022)	URL and third-party features	RF, LR, DT, GNB, and MLP	88646	97.50%	Depends on third party features, low prediction performances
Ding et al. (2019)	URL, HTML and third-party features	Logistic regression	8659	98.90%	Depends on third party features, depends on single ML algorithm, small dataset
Sharma and Singh (2022)	Features from webpage HTML code	TF-IDF and AdaBoost	50000	98.01%	Depends on single ML algorithm, small dataset
Nagunwa et al. (2022)	Features derived from DNS, host, and network	Eight ML and three DL algorithms	11801	98.42%	Computationally expensive model (11 algorithms), small dataset
Sameen et al. (2020)	URL Features only	Boosting-based (2), Ten algorithms	100000	98.00%	Computationally expensive model (10 algorithms)
Rao et al. (2022)	HTML code, and domain specific features	RF, SVM, LR, DT, and XGBoost	10514	99.34%	Experimented on small dataset
Proposed	URL, HTML, and derived features	BernoulliNB, PassiveAggressive, and SGDClassifier	235795	99.79%	Limitation of classifying URLs that download executable file.

8 References

1. PhiUSIIL Phishing URL (Website) dataset from UCI machine learning repository https://archive.ics.uci.edu/dataset/967/phiusiil+phishing+url+dataset
2. M. Alsharaiah, A. Abu-Shareha, M. Abualhaj, L. Baniata, O. Adwan, A. Al-saaidah, M. Oraiqat, A new phishing-website detection framework using ensemble classification and clustering, Int. J. Data Netw. Sci. 7 (2) (2023) 857–864
3. B. Biswas, A. Mukhopadhyay, A. Kumar, D. Delen, A hybrid framework using explainable AI (XAI) in cyber-risk management for defence and recovery against phishing attacks, Decis. Support Syst. 177 (2024) 114102
4. S. Jalil, M. Usman, A. Fong, Highly accurate phishing URL detection based on machine learning, J. Ambient Intell. Humanized Comput. 14 (7) (2023) 9233–9251.
5. A. Prasad, S. Chandra, PhiUSIIL: A diverse security profile empowered phishing URL detection framework based on similarity index and incremental learning, Comput. Secur. 136 (2024) 103545
6. Vajrobol, Vajratiya, Brij B. Gupta, and Akshat Gaurav. "Mutual information based logistic regression for phishing URL detection." Cyber Security and Applications 2 (2024): 100044.

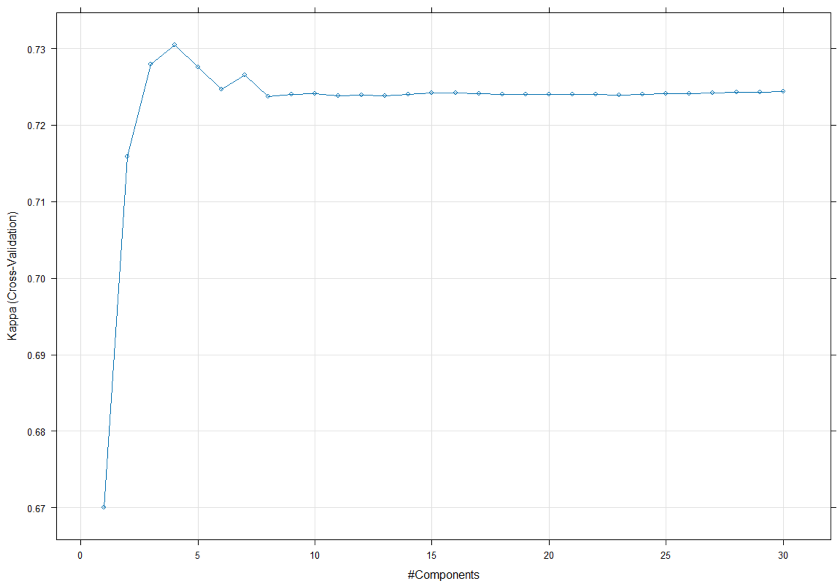
7. Gupta, B.B., Yadav, K., Razzak, I., Psannis, K., Castiglione, A., Chang, X., 2021. A novel approach for phishing URLs detection using lexical based machine learning in a real-timeenvironment.Comput.Commun.175,47–57.https://doi.org/10.1016/j.comcom.2021.04.023
8. Pandey, P., Mishra, N., 2023. Phish-Sight: a new approach for phishing detection using dominant colors on web pages and machine learning. Int. J. Inf. Secur. 1 (11). https://doi.org/10.1007/s10207-023-00672-4.
9. Ahammad, S.H., Kale, S.D., Upadhye, G.D., Pande, S.D., Babu, E.V., Dhumane, A.V., Bahadur, M.D.K.J., 2022. Phishing URL detection using machine learning methods. Adv. Eng. Softw. 173, 103288. https://doi.org/10.1016/j.advengsoft.2022.103288
10. Jain, A.K., Debnath, N., Jain, A.K., 2022. APuML: an efficient approach to detect mobile phishing webpages using machine learning. Wirel. Pers. Commun. 125 (4), 3227–3248. https://doi.org/10.1007/s11277-022-09707-w.
11. Alani, M.M., Tawfik, H., 2022. PhishNot: a cloud-based machine-learning approach to phishing URL detection. Comput. Netw. 218, 109407. https://doi.org/10.1016/j.Comnet.2022.109407.
12. Ding, Y., Luktarhan, N., Li, K., Slamu, W., 2019. A keyword-based combination approach for detecting phishing webpages. Comput. Secur. 84, 256–275. https://doi.org/10.1016/j.cose.2019.03.018.
13. Sharma, B., Singh, P., 2022. An improved anti-phishing model utilizing TF-IDF and AdaBoost. Concurr. Comput., Pract. Exp. 34 (26), e7287. https://doi.org/10.1002/cpe.7287
14. Nagunwa, T., Kearney, P., Fouad, S., 2022. A machine learning approach for detecting fastmflux phishing hostnames. J. Inf. Secur. Appl. 65, 103125. https://doi.org/10.1016/j.jisa.2022.103125.
15. Sameen, M., Han, K., Hwang, S.O., 2020. PhishHaven – an efficient real-time ai phishing URLs detection system. IEEE Access 8, 83425–83443. https://doi.org/10.1109/ACCESS.2020.2991403.
16. Rao, R.S., Umarekar, A., Pais, A.R., 2022. Application of word embedding and machine learning in detecting phishing websites. Telecommun. Syst., 1–13. https://doi.org/10.1007/s11235-021-00850-6.
17. Prasad, Arvind, and Shalini Chandra. "PhiUSIIL: A diverse security profile empowered phishing URL detection framework based on similarity index and incremental learning." Computers & Security 136 (2024): 103545.

Appendix 1: Supplemental Material for Linear Models

Partial Least Squares Discriminant Analysis(PLSDA)

Specific preprocessing - Centering and Scaling

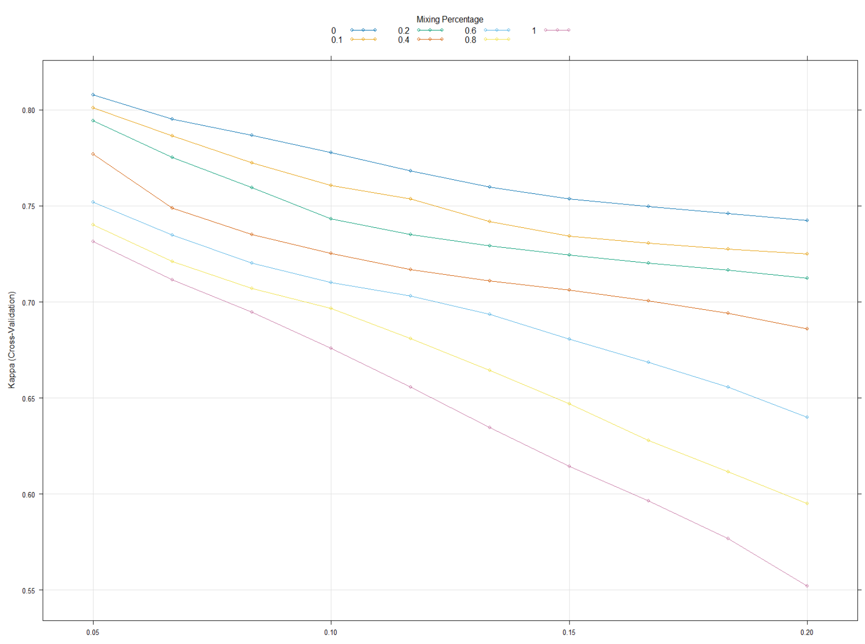
The optimal number of components chosen was 4



Penalized models

Specific preprocessing - Centering and Scaling

The optimal hyperparameters are alpha = 0, and lambda = 0.05

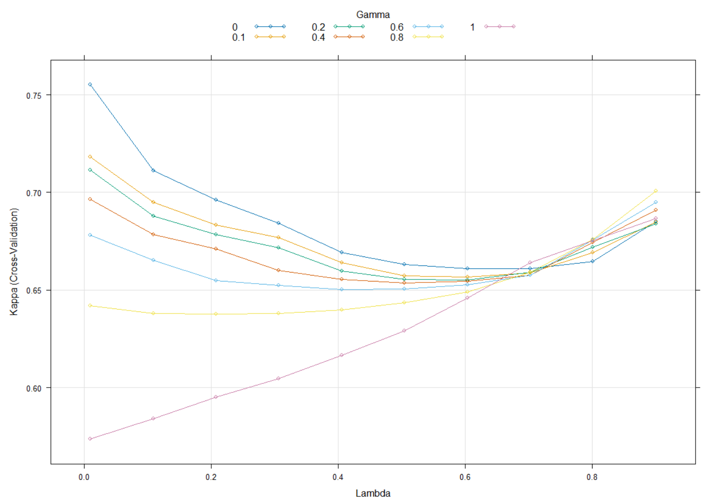


Appendix 2: Supplemental Material for Nonlinear Models

Regularized Discriminant Analysis(RDA)

Specific preprocessing - Remove highly correlated predictors, Center and Scale

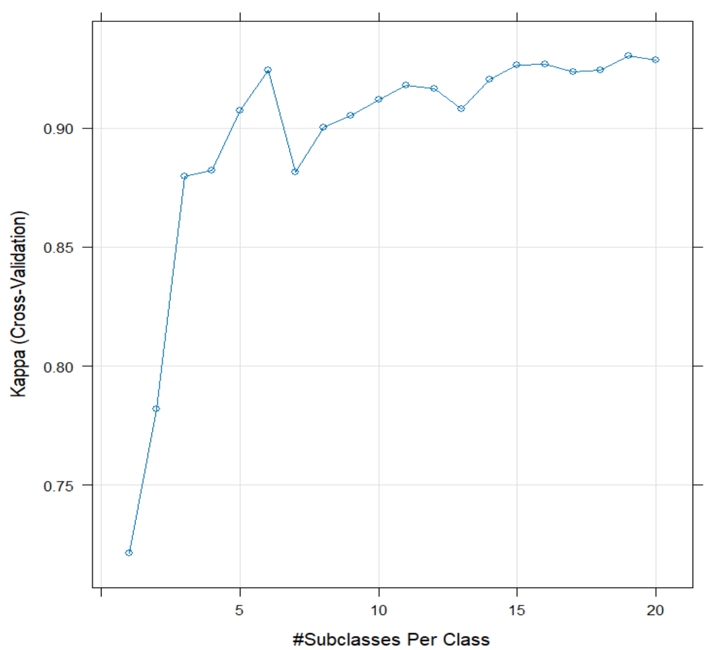
The optimal hyperparameters are gamma = 0, and lambda = 0.01



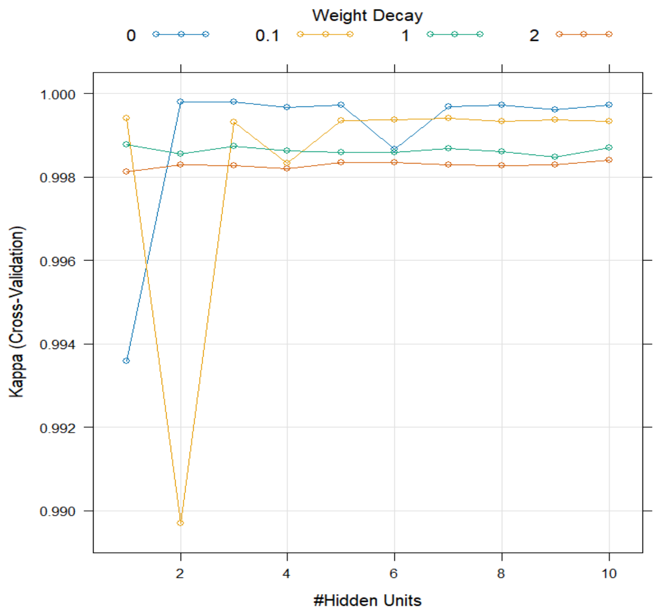
Mixture Discriminant Analysis(MDA)

Specific preprocessing - Remove highly correlated predictors, Center and Scale

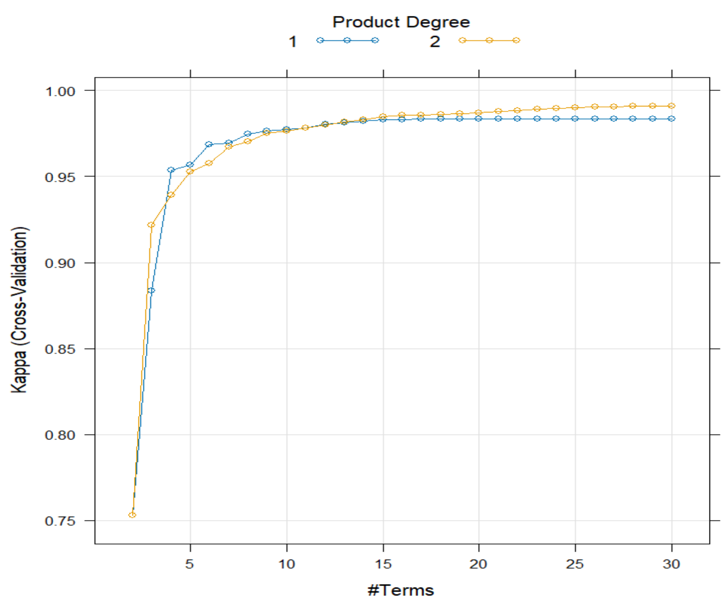
The optimal hyperparameter was subclasses = 19



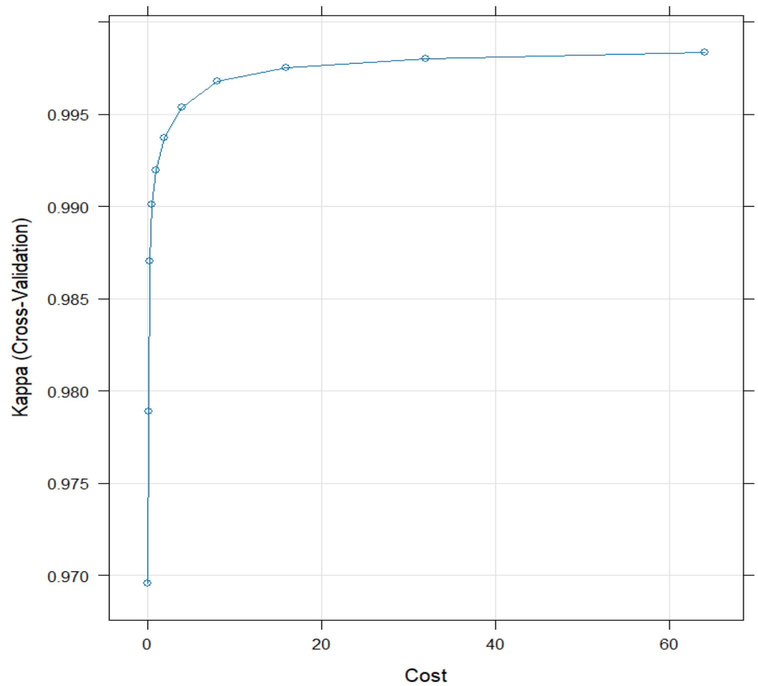
Neural Networks
Specific preprocessing - Remove highly correlated predictors, Spatial Sign transformation, Center and Scale
The optimal hyperparameters are size=3, decay=0



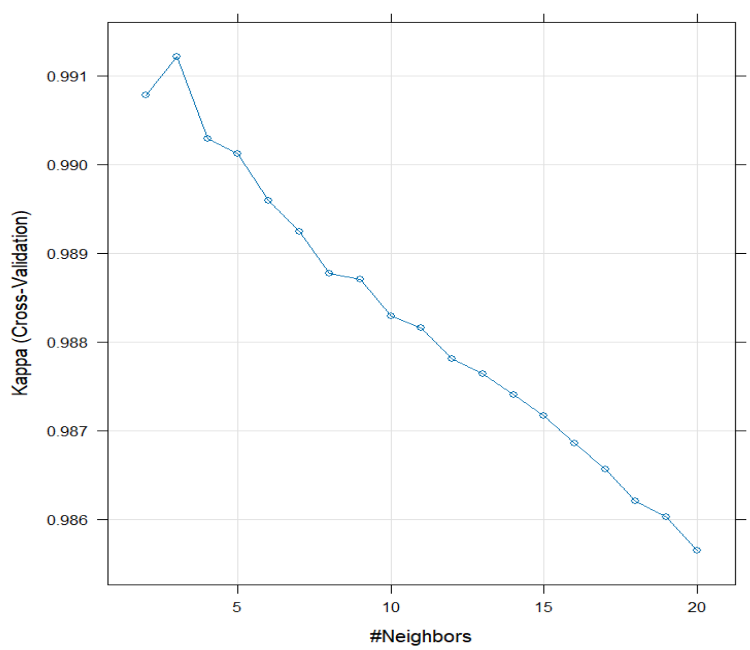
Flexible Discriminant Analysis (FDA)
Specific preprocessing - Remove highly correlated predictors, Spatial Sign transformation
The optimal hyperparameters are degree = 2, and npurne = 30



Support Vector Machines (SVM)
Specific preprocessing - Center and Scale
The optimal hyperparameters are sigma = 0.004992348 and C = 64



K- Nearest Neighbors (KNN)
Specific preprocessing - Center and Scale
The optimal hyperparameter was k = 3



RCode

```
#Detecting phishing URLs

# loading the data set
original_data <- read.csv("D:\\Transfered from
ssd\\Desktop\\PredictiveModeling\\PhiUSIIL_Phishing_URL_Dataset.csv")
# looking the data set
View(original_data)
dim(original_data)
str(original_data)
names(original_data)
original_data$label<-factor(original_data$label)
original_data$label <- ifelse(original_data$label == 0, "phishing", "legitimate")
str(original_data)

###Data preprocessing
# Checking for missing values
total_missing <- sum(is.na(original_data))

# Print the total number of missing values
total_missing
# distribution of each predictors
# Identify numeric columns
numeric_cols <- sapply(original_data, is.numeric)

# Calculate the frequency of each class
freq <- table(original_data["label"])

# Create barplot
barplot(freq,
        main = "Frequency distribution of Label",
        xlab = "Classes",
        ylab = "Frequency",
        col = "lightblue",
        las = 2)  # Rotate x-axis labels

# Calculate percentages
percentages <- round((freq / sum(freq)) * 100, 1)
```

```
# Add percentages to the barplot
text(x = seq_along(freq), y = freq, labels = paste(percentages, "%"), pos = 3)

# Define the columns for numeric and character types explicitly
# Categorizing the columns
numeric_cols <- c(
  "URLLength","DomainLength","URLSimilarityIndex","CharContinuationRate",
"TLDLegitimateProb","URLCharProb","TLDLength","NoOfSubDomain","NoOfObfuscatedCh,
"ObfuscationRatio","NoOfLettersInURL","LetterRatioInURL","NoOfDegitsInURL","DegitRati
oInURL","NoOfEqualsInURL","NoOfQMarkInURL","NoOfAmpersandInURL",
"NoOfOtherSpecialCharsInURL","SpacialCharRatioInURL","LineOfCode","LargestLineLengt",
"DomainTitleMatchScore","URLTitleMatchScore","NoOfURLRedirect","NoOfSelfRedirect","N
oOfPopup","NoOfiFrame","NoOfImage","NoOfCSS",
  "NoOfJS","NoOfSelfRef","NoOfEmptyRef","NoOfExternalRef"
)

factor_cols <- c(
  "IsDomainIP","TLD","HasObfuscation","HasTitle","HasFavicon","Robots","IsResponsive",
  "HasDescription","HasExternalFormSubmit","HasSocialNet","HasSubmitButton",
"HasHiddenFields","HasPasswordField","Bank","Pay","Crypto","HasCopyrightInfo","IsHTTPS
","label"
)
# Check the lists
print(numeric_cols)
print(factor_cols)

# Convert specified columns to numeric
original_data[numeric_cols] <- lapply(original_data[numeric_cols], as.numeric)

# Convert specified columns to character
original_data[factor_cols] <- lapply(original_data[factor_cols], as.character)

str(original_data)

# removing uncessary columns
data <- original_data[ , !(names(original_data) %in% c("URL","Domain", "FILENAME",
"label", "Title"))]
dim(data)
names(data)
```

```
# Adding dummy variables
# Treating "TLD"
unique_values <- unique(data["TLD"])
length(unique_values)
str(data["TLD"])


# adding different classes for TLD
library(dplyr)

classify_tld <- function(tld) {
  tld <- as.character(tld)

  us_country_domains <- c("uk", "de", "fr", "ca", "au", "jp", "it", "es", "nl", "ru", "ch", "se", "dk",
"no", "fi", "ie", "be", "at", "nz", "pl", "br", "mx", "ar", "cl", "za", "in", "cn", "kr", "sg", "hk",
"tw")
  commonly_used_domains <- c("com", "org", "net", "edu", "gov", "mil", "io", "co", "info",
"biz", "me", "tv", "app", "dev")
  numeric_domains <- as.character(0:999)

  if (tld %in% us_country_domains) {
    return("Country Domain")
  } else if (tld %in% commonly_used_domains) {
    return("Common Domain")
  } else if (tld %in% numeric_domains) {
    return("Numeric Domain")
  } else if (startsWith(tld, "xn--")) {
    return("Internationalized Domain")
  } else {
    return("Other")
  }
}

data <- data %>%
  mutate(Classification = sapply(TLD, classify_tld)) %>%
  mutate(
    Internationalized_Domain = as.integer(Classification == "Internationalized Domain"),
    Common_Domain = as.integer(Classification == "Common Domain"),
    Numeric_Domain = as.integer(Classification == "Numeric Domain"),
    Country_Domain = as.integer(Classification == "Country Domain"),
```

```
    Other = as.integer(Classification == "Other")
  ) %>%
  select(-Classification, -TLD)  # Remove the Classification and TLD columns if not needed


#checking for near-zero variance predictors
# considering only catagorical values
cat_data<-data[sapply(data, is.character)] ## for all categorical predictors, need to recall the data
cat_data <- cbind(cat_data,
                  Internationalized_Domain = data[["Internationalized_Domain"]],
                  Common_Domain = data[["Common_Domain"]],
                  Numeric_Domain = data[["Numeric_Domain"]],
                  Country_Domain = data[["Country_Domain"]],
                  Other = data[["Other"]])

dim(cat_data)
#bargraph
# Set up the plot area for multiple graphs
par(mfrow = c(3,3))
# Loop through each categorical column
for (col in names(cat_data)) {
  # Calculate frequencies
  freq <- table(cat_data[[col]])

  # Create barplot
  barplot(freq,
          main = paste("Frequency of", col),
          xlab = "Classes",
          ylab = "Frequency",
          col = "lightblue",
          las = 2)  # Rotate x-axis labels for better readability
}

library(caret)
#near zero variables
nearzero_var<-nearZeroVar(cat_data)

names(cat_data[nearzero_var])

# Remove near-zero variance columns
```

```
cat_data_cleaned <- cat_data[, -nearzero_var]
dim(cat_data_cleaned)
names(cat_data_cleaned)

# bargraph after deleting near-zero var

# Set up the plot area for multiple graphs
par(mfrow = c(3,3))
# Loop through each categorical column
for (col in names(cat_data_cleaned)) {
  # Calculate frequencies
  freq <- table(cat_data_cleaned[[col]])

  # Create barplot
  barplot(freq,
          main = paste("Frequency of", col),
          xlab = "Classes",
          ylab = "Frequency",
          col = "lightblue",
          las = 2)  # Rotate x-axis labels for better readability
}
# changing all to numeric
all_num_data<- data.frame(lapply(data, function(x) as.numeric(as.character(x))))
str(all_num_data)
dim(all_num_data)

# Remove the specified columns
all_num_data <- all_num_data %>%
  select(-IsDomainIP, -HasObfuscation, -HasExternalFormSubmit, -Crypto,
-Internationalized_Domain, -Numeric_Domain)
dim(all_num_data)
#correlation

# Calculate correlation matrix
correlations <- cor(all_num_data, use = "complete.obs")  # Handle missing values

# Print the correlation matrix
print(correlations)

# Convert to a data frame and melt it to long format
```

```
cor_df <- as.data.frame(as.table(correlations))

## To visually examine the correlation structure of the data
library(corrplot)
corrplot(correlations, tl.cex = 0.3,order = "hclust")
# Finding the highly correlated predictors recommended for deletion
highCorr <- findCorrelation(correlations, cutoff = .75)
length(highCorr)
colnames(all_num_data)[highCorr]

#deleting highly correlated data
filtered_data <- all_num_data[, -highCorr]
length(filtered_data)

freq <- table(cat_data[["Numeric_Domain"]])
# Create barplot
barplot(freq,
        main = paste("Frequency of Numeric Domain"),
        xlab = "Classes",
        ylab = "Frequency",
        col = "lightblue",
        las = 2)  # Rotate x-axis labels for better readability

## After deletion plot
correlations2 <- cor(filtered_data, use = "complete.obs")  # Handle missing values
library(corrplot)
corrplot(correlations2, tl.cex = 0.3,order = "hclust")

# let's explore the continious preictors
num_filtered <- filtered_data[, !names(filtered_data) %in% c(
  "IsHTTPS", "HasTitle", "HasFavicon", "Robots", "IsResponsive",
  "HasDescription", "HasSocialNet", "HasSubmitButton", "HasHiddenFields",
"HasPasswordField",
  "Bank", "Pay", "HasCopyrightInfo", "Common_Domain", "Country_Domain", "Other"
)]

dim(num_filtered)

cont_data <- as.data.frame(num_filtered)
```

```
# Display the continuous predictors
print(cont_data)
dim(cont_data)

# Histogram
# Set up the layout for multiple plots
par(mfrow = c(3, 3))  # Adjust the number of rows and columns as needed
for (col in names(cont_data)) {
   x_limits <- range(cont_data[[col]])
   hist(cont_data[[col]],
        main = paste(col),
        xlab = col,
        xlim = x_limits,
        col = "lightblue",
        border = "black",
        breaks = 20)
  }

#skewness
library(e1071)
# Function to calculate skewness for numeric or integer columns
skewness_results <- sapply(cont_data, skewness)

# Print the skewness results
print(skewness_results)


#Center and scale
s_c_data<-scale(cont_data, center = TRUE, scale = TRUE)
# Convert scaled data to a data frame
s_c_data_df <- as.data.frame(s_c_data)
#adding constant
# Find the minimum value in the data frame
min_value <- min(s_c_data_df, na.rm = TRUE)  # Overall minimum value
shift_constant <- abs(min_value) + 1  # Shift constant to ensure positivity

# Shift the data
s_c_data_df_shifted <- s_c_data_df + shift_constant
```

```
#Box cox transformation
library(caret)
xx1 <- preProcess(s_c_data_df_shifted, method = c("BoxCox"))
transformed_data <- predict(xx1, s_c_data_df_shifted)
# Convert transformed data to a data frame
transformed_data <- as.data.frame(transformed_data)

# Histogram after boxcox transformation
# Set up the layout for multiple plots
par(mfrow = c(3, 3))  # Adjust the number of rows and columns as needed
for (col in names(transformed_data)) {
    hist(transformed_data[[col]],
         main = paste(col),
         xlab = col,
         col = "lightblue",
         border = "black",
         breaks = 20)
}

# Function to calculate skewness for numeric or integer columns
skewness_after<- sapply(transformed_data, skewness)

# Print the skewness results after boxcox
print(skewness_after)



#boxplot
# Set up the layout for multiple plots
par(mfrow = c(3, 3))  # Adjust the number of rows and columns as needed

# Loop through each column in the data frame
for (col in names(transformed_data)) {
  boxplot(transformed_data[[col]],
          main = paste(col),
          ylab = col,
          col = "lightblue",
          border = "black")
}
```

```
str(cat_data_cleaned)
cat_data_cleaned <- data.frame(lapply(cat_data_cleaned, function(x)
as.numeric(as.character(x))))

View(transformed_data)
View(cat_data_cleaned)
dim(transformed_data)
dim(cat_data_cleaned)


#applying spatialsign
spatialsign_data<-spatialSign(transformed_data)
spatialsign_data <- as.data.frame(spatialsign_data)
View(spatialsign_data)

#boxplot after spatialsign
# Set up the layout for multiple plots
par(mfrow = c(3, 3))  # Adjust the number of rows and columns as needed

# Loop through each column in the data frame
for (col in names(spatialsign_data)) {
  boxplot(spatialsign_data[[col]],
          main = paste(col),
          ylab = col,
          col = "lightblue",
          border = "black")
}

merged_allfinal<-cbind(spatialsign_data, cat_data_cleaned)
dim(merged_allfinal)
# Step 2: Convert all factor columns in merged_all to numeric
merged_allfinal <- as.data.frame(lapply(merged_allfinal, function(x) {
  if (is.factor(x)) {
    as.numeric(as.factor(x))  # Convert factor to character, then to numeric
  } else {
    x  # Keep other types unchanged
  }
}))

# Step 3: Check the structure of the final data frame
```

```
str(merged_allfinal)
View(merged_allfinal)


#PCA
pcaObject_data <- prcomp(merged_allfinal, center = TRUE, scale. = TRUE)
summary(pcaObject_data)


#scree plot
screeplot(pcaObject_data,
        main = "Scree Plot",
        xlab = "Principal Component",
        ylab = "Variance Explained",
        type = "lines",
        col = "blue",  # Optional: line color
        pch = 19)      # Optional: point type
# Load necessary library
library(ggplot2)


# Calculate variance explained
variance <- pcaObject_data$sdev^2
variance_explained <- variance / sum(variance)

# Create a data frame for plotting
scree_data <- data.frame(PC = 1:length(variance), Variance = variance_explained)

# Draw the scree plot using ggplot2
ggplot(scree_data, aes(x = PC, y = Variance)) +
  geom_line() +
  geom_point() +
  labs(title = "Scree Plot", x = "Principal Component", y = "Proportion of Variance Explained") +
  theme_minimal()

original_data$label<-factor(original_data$label)
str(original_data)
# applying spatial sign transformation on all_num_data
spatialsign_data49<-spatialSign(all_num_data)
spatialsign_data49 <- as.data.frame(spatialsign_data49)
```

```
no_boxcox_data<- cbind(cont_data,cat_data_cleaned)
# applying spatial sign transformation on no_boxcox_data
spatialsign_data44<-spatialSign(no_boxcox_data)
spatialsign_data44 <- as.data.frame(spatialsign_data44)


###############Model building###################
################Linear Models##################
# Logistic regression
#spliting data
# Set the random number seed so we can reproduce the results
set.seed(476)
library(caret)
# use createDataPartition for stratified sampling, p is percentage of training set
trainingRows <- createDataPartition(original_data$label, p = .80, list= FALSE)

trainPredictors <- spatialsign_data49[trainingRows, ]
trainClasses <- original_data$label[trainingRows]

testPredictors <- spatialsign_data49[-trainingRows, ]
testClasses <- original_data$label[-trainingRows]
dim(trainPredictors)
dim(testPredictors)
library(caret)

ctrl <- trainControl(method = "CV",number=3,
                     summaryFunction =defaultSummary,
                     classProbs = TRUE,
                     savePredictions = TRUE)
set.seed(476)
logistic <- train(trainPredictors,
                  y = trainClasses,
                  method = "glm",
                  metric = "Kappa",
                  trControl = ctrl)
logistic
# predicting on test set
predicted<- predict(logistic, newdata= testPredictors)
```

```
#confusion matrix
confusionMatrix(predicted, testClasses)

# LDA
set.seed(476)

# use createDataPartition for stratified sampling, p is percentage of training set
trainingRows <- createDataPartition(original_data$label, p = .80, list= FALSE)

trainPredictors <- spatialsign_data44[trainingRows, ]
trainClasses <- original_data$label[trainingRows]

testPredictors <- spatialsign_data44[-trainingRows, ]
testClasses <- original_data$label[-trainingRows]
dim(trainPredictors)
dim(testPredictors)


LDA <- train(trainPredictors,
        y = trainClasses,
        method = "lda",
        metric = "Kappa",
        trControl = ctrl,
        preProcess = c("center", "scale"))
LDA

# predicting on test set
predicted<- predict(LDA, newdata= testPredictors)

#confusion matrix
confusionMatrix(predicted, testClasses)

#PLSDA
#spliting data
# Set the random number seed so we can reproduce the results
set.seed(476)

# use createDataPartition for stratified sampling, p is percentage of training set
trainingRows <- createDataPartition(original_data$label, p = .80, list= FALSE)
```

```
trainPredictors <- spatialsign_data49[trainingRows, ]
trainClasses <- original_data$label[trainingRows]

testPredictors <- spatialsign_data49[-trainingRows, ]
testClasses <- original_data$label[-trainingRows]
dim(trainPredictors)
dim(testPredictors)

set.seed(476)
plsda <- train(x = trainPredictors,
           y = trainClasses,
           method = "pls",
           tuneGrid = expand.grid(.ncomp = 1:30),
           preProc = c("center","scale"),
           metric = "Kappa",
           trControl = ctrl)
plsda
plot(plsda)

# predicting on test set
predicted<- predict(plsda, newdata= testPredictors)

#confusion matrix
confusionMatrix(predicted, testClasses)

# Penalized Models


glmnGrid <- expand.grid(.alpha = c(0, .1, .2, .4, .6, .8, 1),
                .lambda = seq(.01, .2, length = 10))
set.seed(100)
glmnTuned <- train(x = trainPredictors,
             y = trainClasses,
             method = "glmnet",
             tuneGrid = glmnGrid,
             preProc = c("center", "scale"),
             metric = "Kappa",
             trControl = ctrl)
glmnTuned
plot(glmnTuned)
```

```
# predicting on test set
predicted<- predict(glmnTuned, newdata= testPredictors)

#confusion matrix
confusionMatrix(predicted, testClasses)


####################Non Linear Models#####################
#QDA
set.seed(476)

# use createDataPartition for stratified sampling, p is percentage of training set
trainingRows <- createDataPartition(original_data$label, p = .80, list= FALSE)

trainPredictors <- spatialsign_data44[trainingRows, ]
trainClasses <- original_data$label[trainingRows]

testPredictors <- spatialsign_data44[-trainingRows, ]
testClasses <- original_data$label[-trainingRows]
dim(trainPredictors)
dim(testPredictors)
levels(trainClasses)
library(caret)

ctrl <- trainControl(method = "cv", number = 3,
                summaryFunction =defaultSummary,
                classProbs = FALSE,
                savePredictions = TRUE)

set.seed(476)

qda <- train(x = trainPredictors,
         y = trainClasses,
         method = "qda",
         metric = "Kappa",
         preProc= c("center","scale"),
         trControl = ctrl)
qda
```

```
# predicting on test set
predicted<- predict(qda, newdata= testPredictors)

#confusion matrix
confusionMatrix(predicted, testClasses)

#RDA
rdaGrid <- expand.grid(.gamma = c(0, .1, .2, .4, .6, .8, 1),
                  .lambda = seq(.01, .9, length = 10))
set.seed(476)

rda <- train(x = trainPredictors,
          y = trainClasses,
          method = "rda",
          metric = "Kappa",
          preProc= c("center","scale"),
          tuneGrid = rdaGrid,
          trControl = ctrl)
rda
plot(rda)
# predicting on test set
predicted<- predict(rda, newdata= testPredictors)

#confusion matrix
confusionMatrix(predicted, testClasses)

#MDA
set.seed(476)
mda <- train(x = trainPredictors,
          y = trainClasses,
          method = "mda",
          metric = "Kappa",
          preProc= c("center","scale"),
          tuneGrid = expand.grid(.subclasses = 1:30),
          trControl = ctrl)
mda
plot(mda)
# predicting on test set
predicted<- predict(mda, newdata= testPredictors)
```

```
#confusion matrix
confusionMatrix(predicted, testClasses)

#Neural Network
nnetGrid <- expand.grid(.size = 1:10, .decay = c(0, .1, 1, 2))
maxSize <- max(nnetGrid$.size)
numWts <- (maxSize * (44 + 1) + (maxSize+1)*3)

set.seed(476)
nnetFit <- train(x = trainPredictors,
                 y = trainClasses,
                 method = "nnet",
                 metric = "Kappa",
                 preProc = c("center", "scale"),
                 tuneGrid = nnetGrid,
                 trace = FALSE,
                 maxit = 2000,
                 MaxNWts = numWts,
                 trControl = ctrl)
nnetFit
plot(nnetFit)
# predicting on test set
predicted<- predict(nnetFit, newdata= testPredictors)

#confusion matrix
confusionMatrix(predicted, testClasses)

#FDA
marsGrid <- expand.grid(.degree = 1:2, .nprune = 2:20)
set.seed(476)
fda <- train(x = trainPredictors,
             y = trainClasses,
             method = "fda",
             metric = "Kappa",
             tuneGrid = marsGrid,
             trControl = trainControl(method="CV",number=3))
fda
plot(fda)

# predicting on test set
```

```
predicted<- predict(fda, newdata= testPredictors)

#confusion matrix
confusionMatrix(predicted, testClasses)

set.seed(476)

# use createDataPartition for stratified sampling, p is percentage of training set
trainingRows <- createDataPartition(original_data$label, p = .80, list= FALSE)

trainPredictors <- spatialsign_data49[trainingRows, ]
trainClasses <- original_data$label[trainingRows]

testPredictors <- spatialsign_data49[-trainingRows, ]
testClasses <- original_data$label[-trainingRows]
dim(trainPredictors)
dim(testPredictors)

# SVM
library(kernlab)
sigmaRangeReduced <- sigest(as.matrix(trainPredictors))
svmRGrid<- expand.grid(.sigma = sigmaRangeReduced[1],.C = 2^(seq(-4, 6)))
set.seed(476)
svmRModel <- train(x = trainPredictors,
        y = trainClasses,
        method = "svmRadial",
        metric = "Kappa",
        preProc = c("center", "scale"),
        tuneGrid = svmRGrid,
        fit = FALSE,
        trControl = ctrl)
svmRModel
plot(svmRModel)
# predicting on test set
predicted<- predict(svmRModel, newdata= testPredictors)

#confusion matrix
confusionMatrix(predicted, testClasses)
```

```
#KNN
set.seed(476)
knnFit <- train(x = trainPredictors,
                y = trainClasses,
                method = "knn",
                metric = "Kappa",
                preProc = c("center", "scale"),
                tuneGrid = data.frame(.k = 2:20),
               trControl = ctrl)
knnFit
plot(knnFit)
# predicting on test set
predicted<- predict(knnFit, newdata= testPredictors)

#confusion matrix
confusionMatrix(predicted, testClasses)

#Top 10 most important variables
ImpSim <- varImp(svmRModel, scale = FALSE)
ImpSim
plot(ImpSim, top = 10, scales = list(y = list(cex = .95)))
```