Project Report: Website Phishing Detection – Harshad Kamble

1. Introduction

Phishing is a prevalent cybercrime where attackers masquerade as legitimate entities to extract sensitive personal information from victims. This project aims to analyze a dataset containing features of URLs to classify them as phishing (malicious) or legitimate. Using machine learning techniques, we identify patterns and build predictive models to detect phishing websites.

---

2. Objective

- To build predictive models to detect phishing websites using machine learning techniques.
- Perform deep Exploratory Data Analysis (EDA).
- Check distribution of all the variables.
- Make Q-Q (Quantile-Quantile) plots are used to visually assess how well a dataset's distribution matches a theoretical distribution (normal_distribution).
- Plot correlation and note findings.
- Calculate Variance Inflation Factor(VIF) and note insights.
- Apply PCA on it, If applicable.
- Build a classification based model using 10 different ML algorithms and share the findings.
- Do hyperparameter tuning.
- Use different metrics that can be used for evaluation of a model & note findings
- Plot confusion matrix and share your findings

3. Exploratory Data Analysis (EDA)

Dataset Overview

The dataset comprises the following features:

1. `url_length`: Length of the URL.
2. `n_dots`: Count of '.' characters.
3. `n_hyphens`: Count of '-' characters.
4. `n_underline`: Count of '_' characters.
5. `n_slash`: Count of '/' characters.
6. `n_questionmark`: Count of '?' characters.
7. `n_equal`: Count of '=' characters.
8. `n_at`: Count of '@' characters.
9. `n_and`: Count of '&' characters.
10. `n_exclamation`: Count of '!' characters.
11. `n_space`: Count of spaces.
12. `n_tilde`: Count of '~' characters.
13. `n_comma`: Count of ',' characters.
14. `n_plus`: Count of '+' characters.
15. `n_asterisk`: Count of '*' characters.
16. `n_hastag`: Count of '#' characters.
17. `n_dollar`: Count of '$' characters.
18. `n_percent`: Count of '%' characters.
19. `n_redirection`: Count of redirections in the URL.
20. `phishing`: Target variable (1 = Phishing, 0 = Legitimate).

Checked for null values in data, Checked data is imbalanced or not.

Data columns are **not-null and balanced.**

Key EDA Insights:

1) Distribution of Variables:

Plotted Histogram to check Distribution of Variables

Key Observations and likely interpretations from the chart:
Highly Skewed Distributions for all features: The distribution is heavily skewed to the right (positively skewed). This means that features are relatively short, with a long tail of less frequent, much longer URL feature' frequency.

---

2) Q-Q Plots:

Used to evaluate normality of numeric features.

Q-Q (Quantile-Quantile) plots are used to visually assess how well a dataset's distribution matches a theoretical distribution, most commonly the normal distribution.

The primary purpose was to see if the counts of various characters in URLs were normally distributed. In all the Q-Q plots ("n_hyphens", "n_dots", "n_slash", "n_questionmark", "n_at", "n_equal", "n_exclamation", "n_space", "n_plus", "n_asterisk", "n_hashtag", "n_dollar", "n_percent", "n_comma", and "n_redirection"), **a consistent pattern emerges:**

Non-Normal Distribution: The data for each character count **is not normally distributed**. This is evident from the significant deviations of the blue dots from the red line in each plot.

Right Skew (Positive Skew): The data is consistently right-skewed. This indicates that most URLs have a small number (often zero) of these characters, but there's a long tail of URLs with a considerably higher count.

Discrete Nature: The plots show a stepped pattern, confirming that these are count variables, taking only whole number values.

The Q-Q plots clearly showed that the character counts are not normally distributed, which is important information for choosing appropriate statistical methods or machine learning models.

Visualizing Skewness and Discrete Nature: Q-Q plots effectively visualize the right skew and the discrete nature of the data, which is harder to see in simple histograms.

---

3) Correlation Matrix

Based on the correlation Matrix findings are:-

- **url_length has a noticeable positive correlation with phishing**: This is a key finding.
- Longer URLs are more likely to be associated with phishing.
- n_at has a moderate positive correlation with phishing: The presence of the "@" symbol is also somewhat indicative of phishing.
- n_equal also has a positive correlation with phishing
- n_percent also has a positive correlation with phishing
- n_redirection also has a positive correlation with phishing
- **Correlations among character counts:** There appear to be some moderate to strong positive correlations among the character count features themselves (e.g., n_dots, n_hyphens, n_slash). This is expected because URLs that are longer will tend to have more of these characters.

---

4)VIF Analysis:

VIF is a measure of multicollinearity. It quantifies how much the variance of an estimated regression coefficient is increased due to multicollinearity

Here, Correlation matrix likely revealed some moderate to strong correlations between certain URL features (especially the character counts). This is called multicollinearity. And Multicollinearity can cause problems for some machine learning models (particularly linear models like linear regression or logistic regression) because:

Unstable Coefficients: It becomes difficult to determine the individual effect of each predictor variable on the outcome. The coefficients in the model become very sensitive to small changes in the data.
Reduced Interpretability: It becomes harder to interpret the model and understand the relationships between the features and the target variable.
So, VIF analysis performed.

VIF values are interpreted as follows:

VIF = 1: No multicollinearity. 1 < VIF < 5: Moderate multicollinearity, may not require action. VIF ≥ 5: High multicollinearity, likely requires action. VIF ≥ 10: Very high multicollinearity, definitely requires action.

Based on VIF results:

- url_length (4.78), n_equal (7.26), and n_and (6.08) have VIF values that suggest moderate to high multicollinearity. These features are likely correlated with other features in your dataset.
- The value of n_equal is of particular concern and should be addressed.

- The rest of the features have VIF values below 5, suggesting they have relatively low multicollinearity.

5) Dimensionality Reduction

Principal Component Analysis (PCA)

PCA is a dimensionality reduction technique.

After identifying multicollinearity with VIF, PCA can be used to address this issue by creating new, uncorrelated features (called principal components) that capture most of the variance in the original data.

After performing PCA, we get output as:-
Explained variance ratio: [0.19597369 0.11471021 0.06621965 0.06092402 0.05603545]

Interpretation of PCA Results:
This tells us how much variance in the original data is explained by each principal component:

First Principal Component: Explains 19.6% of the variance.
Second Principal Component: Explains 11.5% of the variance.
Third Principal Component: Explains 6.6% of the variance.
Fourth Principal Component: Explains 6.1% of the variance.
Fifth Principal Component: Explains 5.6% of the variance.

---

6) Model Building

Algorithms Applied:

1. Logistic Regression
2. Decision Trees
3. Random Forest
4. Gradient Boosting
5. XGBoost
6. Support Vector Machines (SVM)
7. k-Nearest Neighbors (kNN)
8. Naive Bayes
9. Adaboost
10. Neural Networks

Key Findings:

Performing of Models:

Model Performance Summary:

Logistic Regression: Accuracy = 0.8559152677857714

Decision Tree: Accuracy = 0.882494004796163

Random Forest: Accuracy = 0.8912869704236611

Gradient Boosting: Accuracy = 0.8829936051159073

AdaBoost: Accuracy = 0.8723521183053558

XGBoost: Accuracy = 0.8945843325339728

Naive Bayes: Accuracy = 0.7156274980015987

Neural Network: Accuracy = 0.8884892086330936

SVM (Linear): Accuracy = 0.7855715427657873

KNN: Accuracy = 0.8815947242206235

Insights from result:-
Tree-Based Models Perform Best: The tree-based models (Random Forest, XGBoost, Gradient Boosting, and Decision Tree) **generally achieve the highest accuracy**, ranging from approximately 88% to 89.5**%. XGBoost shows the best performance** among them, with an accuracy of around 89.5%.

Neural Network Performs Well: The Neural Network also achieves a good accuracy of approximately 89%, comparable to the tree-based models.

KNN Performs Respectably: KNN achieves an accuracy of around 88%, which is quite good but slightly lower than the top performers.

Logistic Regression and SVM (Linear) Perform Moderately: These linear models achieve accuracies around 85-86%, which is reasonable but lower than the non-linear models.

Naive Bayes Performs Significantly Worse: Naive Bayes has the lowest accuracy, around 71.5%. This suggests that the assumptions of Naive Bayes (feature independence) are likely violated in your dataset.

Among all of them **XGBoost shows the best performance.**

On this XGBoost model performed, Hyperparameter Tuning

---

7) Hyperparameter Tuning

Default hyperparameters are often not optimal for every dataset. **Hyperparameter Tuning** aims to find the settings that give the best performance on our specific data.

Grid Search and Random Search

- Performed hyperparameter tuning XGBoost models.
- And reduced the effect of overfitting

Key Improvements and Explanations:

- **Broader Parameter Grid:** Increased the number of values for most hyperparameters, especially `n_estimators`, `max_depth`, `learning_rate`, `gamma`, `reg_alpha`, and `reg_lambda`. This allows for more extensive exploration of the parameter space. I also added `min_child_weight`.
- **Logarithmic Scale for `learning_rate`:** Using `np.logspace` creates a logarithmic scale for the learning rate, which is often more effective than a linear scale. Learning rate values are often more impactful at the lower end of the scale.
- **Stratified K-Fold Cross-Validation:** Using `StratifiedKFold` ensures that each fold of the cross-validation has the same proportion of classes as the original dataset. This is crucial for imbalanced datasets (which is common in phishing detection).
- **Increased `n_iter`:** Increasing `n_iter` increases the number of random parameter combinations tried, which improves the chances of finding better hyperparameters.
- **XGBClassifier Initialization:** Initialize the `XGBClassifier` *before* passing it to `RandomizedSearchCV`. This makes the code cleaner and allows you to set parameters that you *don't* want to tune (like `random_state`, `n_jobs`, and `eval_metric`).
- **Verbose Output:** The `verbose=1` parameter in `RandomizedSearchCV` provides output during the search process, so you can track its progress.
- **Evaluation on Test Set:** *Crucially*, the code now evaluates the best model found by `RandomizedSearchCV` on your *test set*. This gives you a more realistic estimate of the model's performance on unseen data. The `best_score_` attribute of `RandomizedSearchCV` shows the average cross-validation score on the *training data*, which can be overly optimistic.
- **Classification Report:** Added a classification report to show precision, recall, and f1-score along with accuracy. This gives a more comprehensive evaluation of the model's performance, especially in imbalanced datasets

---

6. Model Evaluation

Different Evaluation Metrics can be used and Why They Are Used:

1) Accuracy: The most intuitive metric, it measures the overall correctness of the model's predictions: (True Positives + True Negatives) / Total Predictions.

Why Use It: Easy to understand and provides a general sense of model performance.

Limitations: Can be misleading on imbalanced datasets (where one class has many more samples than the other). A model that always predicts the majority class can achieve high accuracy even if it performs poorly on the minority class.

2) Precision: Measures the proportion of true positives among all predicted positives: True Positives / (True Positives + False Positives).

Why Use It: Important when the cost of false positives is high. In phishing detection, a false positive means a legitimate website is flagged as phishing, which can be very disruptive for users.

Focus: Minimizing false positives.

3) Recall (Sensitivity or True Positive Rate): Measures the proportion of true positives that were correctly identified: True Positives / (True Positives + False Negatives).

Why Use It: Important when the cost of false negatives is high. In phishing detection, a false negative means a phishing website is not detected, which can lead to significant harm for users.

Focus: Minimizing false negatives.

4) F1-Score: The harmonic mean of precision and recall: 2 * (Precision * Recall) / (Precision + Recall).

Why Use It: Provides a balanced measure that considers both precision and recall. Useful when you need to find a trade-off between minimizing false positives and false negatives.

5) ROC-AUC (Receiver Operating Characteristic - Area Under the Curve): Measures the ability of the model to distinguish between the two classes (phishing and legitimate) across different classification thresholds.

Why Use It: Provides a good overall measure of performance, especially when you need to compare models or when the class distribution is imbalanced. It is less sensitive to class imbalance than accuracy.

Interpretation: An AUC of 0.5 means the model performs no better than random chance, while an AUC of 1.0 means perfect classification.

Then calculated all metrices and the result is:

```
Logistic Regression Metrics:
Precision: 0.8654797230464887
Recall: 0.7174091281770976
F1-Score: 0.7845188284518828
ROC-AUC: 0.9329875445261893

Decision Tree Metrics:
Precision: 0.8615317414094351
Recall: 0.808554249795026
F1-Score: 0.8342027350909348
ROC-AUC: 0.9154206120608457

Random Forest Metrics:
Precision: 0.8588777219430486
Recall: 0.8408034982235584
F1-Score: 0.8497445104267366
ROC-AUC: 0.9558660562521848

Gradient Boosting Metrics:
Precision: 0.8376764386536374
Recall: 0.8433998360207707
F1-Score: 0.8405283943892143
ROC-AUC: 0.9544516363938155

AdaBoost Metrics:
Precision: 0.846299258397557
Recall: 0.7952992620934681
F1-Score: 0.8200070447340613
ROC-AUC: 0.9489856297994347

XGBoost Metrics:
Precision: 0.8621696801112656
Recall: 0.8470893686799672
F1-Score: 0.8545629997242901
ROC-AUC: 0.9616748161970362

Naive Bayes Metrics:
Precision: 0.8864068441064639
Recall: 0.25485105220005466
F1-Score: 0.395881978348546
ROC-AUC: 0.9034718213269507

Neural Network Metrics:
Precision: 0.8402461867808403
Recall: 0.8581579666575567
F1-Score: 0.8491076257436452
ROC-AUC: 0.9574643737755312

SVM (Linear) Metrics:
Precision: 0.9163456246560264
Recall: 0.45504236130090187
F1-Score: 0.6081081081081081

KNN Metrics:
Precision: 0.8439933259176863
Recall: 0.8294616015304728
F1-Score: 0.8366643694004134
ROC-AUC: 0.9187943488936826
```

Interpretation of Your Results:

XGBoost and Random Forest generally perform best: XGBoost shows the highest ROC-AUC (0.962) and a good balance between precision (0.862), recall (0.847), and F1-score (0.855), indicating strong overall performance. Random Forest also performs very well, with an ROC-AUC of 0.956 and a good F1-score of 0.850. Neural Network is also comparable to

Decision Tree: A decent F1 score and ROC-AUC but slightly lower than Random Forest and XGBoost**.**

Gradient Boosting and AdaBoost: Perform reasonably well, but slightly behind XGBoost and Random Forest.

Naive Bayes: Has very low recall (0.255), meaning it misses a large proportion of phishing websites (high false negatives). While precision is high (0.886), the low recall makes it unsuitable for phishing detection where minimizing false negatives is crucial.

SVM (Linear): Has very high precision (0.916) but very low recall (0.455). This means it is very cautious about labeling a website as phishing (few false positives) but misses many actual phishing websites (many false negatives). This is also not desirable for phishing detection.

KNN: Performs reasonably well, but not as good as the tree based models.

Key Findings and Recommendations:

XGBoost and Random Forest are strong candidates: Given their high ROC-AUC and good balance of precision and recall, XGBoost and Random Forest are the most promising models.

Consider the Trade-off Between Precision and Recall: Depending on your specific needs, you might prioritize precision or recall. If minimizing false positives is paramount (e.g., you don't want to block legitimate websites), you might favor a model with higher precision. If minimizing false negatives is more important (e.g., you want to catch as many phishing sites as possible), you might prioritize recall. The F1-score helps you balance these two.

ROC-AUC is a Robust Metric: The high ROC-AUC values for several models suggest that they are generally good at distinguishing between phishing and legitimate websites.

Naive Bayes and SVM are not suitable: Due to their low recall, Naive Bayes and SVM are not well-suited for this particular problem.

7) Confusion Matrix

Result of confusion matrix of all variables

```
Summary of Model Performance:
Model                Accuracy Precision Recall   F1-Score
Logistic Regression  0.8559   0.8586    0.8266   0.8381
Decision Tree        0.8825   0.8775    0.8668   0.8716
Random Forest        0.8913   0.8841    0.8806   0.8823
Gradient Boosting    0.8830   0.8735    0.8746   0.8741
AdaBoost             0.8724   0.8661    0.8560   0.8606
XGBoost              0.8946   0.8875    0.8845   0.8859
Naive Bayes          0.7156   0.7910    0.6180   0.6050
Neural Network       0.8885   0.8787    0.8821   0.8803
SVM (Linear)         0.7856   0.8365    0.7156   0.7303
KNN                  0.8816   0.8733    0.8706   0.8719
```

Based on the summary of model performance , key observations are:

XGBoost is the Top Performer: XGBoost achieves the highest accuracy (0.8946) and F1-score (0.8859), along with very competitive precision (0.8875) and recall (0.8845). This indicates that XGBoost provides the best overall balance between minimizing false positives and false negatives for your phishing detection task.

Random Forest is a Close Second: Random Forest performs very closely to XGBoost, with a slightly lower accuracy (0.8913) and F1-score (0.8823). However, it still demonstrates strong performance and could be a suitable alternative if computational resources are a concern (Random Forest can sometimes be faster to train than XGBoost, though XGBoost is often optimized).

Neural Network Performs Well: The Neural Network also achieves good results, with an accuracy of 0.8885 and an F1-score of 0.8803. Its recall (0.8821) is slightly higher than Random Forest's, but its precision (0.8787) is slightly lower. This suggests a slightly different trade-off between false positives and false negatives.

Gradient Boosting, Decision Tree, and KNN Offer Reasonable Performance: These models achieve accuracies in the 0.88 range and F1-scores in the 0.87 range. While they are decent performers, they are outperformed by XGBoost, Random Forest, and the Neural Network.

AdaBoost Performs Moderately: AdaBoost has an accuracy of 0.8724 and an F1-score of 0.8606. Its performance is noticeably lower than the top three models.

Naive Bayes is Not Suitable: Naive Bayes performs significantly worse than all other models, with a low accuracy (0.7156) and a very low F1-score (0.6050). This confirms that the feature independence assumption of Naive Bayes is not met in your dataset, making it unsuitable for this task.

SVM (Linear) Shows a Trade-off: The Linear SVM has a reasonable accuracy (0.7856) but demonstrates a clear trade-off between precision (0.8365) and recall (0.7156). It prioritizes minimizing false negatives (high recall) at the cost of increasing false positives (lower precision). This trade-off might not be ideal for most phishing detection scenarios.

---

7. Conclusion

This project successfully built a machine learning pipeline to detect phishing websites. Key takeaways include:

- Select XGBoost as the Primary Model: Based on these results, XGBoost is the recommended model for your phishing detection project due to its superior overall performance across all key metrics.
- Consider Random Forest or Neural Network as Alternatives: If computational resources are a significant constraint or if minimizing false negatives is absolutely paramount, Random Forest or the Neural Network could be considered as alternatives.
- Do Not Use Naive Bayes or SVM: These models are not suitable for this task due to their low overall performance or undesirable trade-offs between precision and recall.
- Feature engineering and dimensionality reduction are essential for optimizing model performance.
- Future work could involve real-time URL classification and integration with cybersecurity tools.

8. References

- Hannousse, A., & Yahiouche, S. (2021). “Web page phishing detection.” Mendeley Data, V3.
- Vrbančič, G. (2020). “Phishing Websites Dataset.” Mendeley Data, V1.
- Documentation and insights derived from the "Data Science Project" notebook.