CS235 Project Final Report: Detection of Phishing Websites

Hritvik Gupta #1
NetID: hgupt010

Nityash Gautam #2
NetID: ngaut006

Yash Aggarwal #3
NetID: yagga004

Shubham Sharma #4
NetID: sshar180

Siddhant Poojary #5
NetID: spooj003

ABSTRACT

Phishing is a kind of fraud where perpetrator designed lookalike of the the trustworthy website and get sensitive information of user such as login passwords or account information. The phishing website hyperlink is a lookalike of the legitimate link but can break through to user sensitive information from any server. The purpose and goal of this project is to classifies and distinguish those malicious phishing website from the legitimate websites through hyper parameters features such as URl, hyperlink and domain name. Furthermore, we have used multiple machine learning techniques to train and predict on dataset and tries to compare the output on key measures such as accuracy, F1, recall and miss rate. Additionally we have used the implemented model from Python inbuilt Scikit learn library and from the scratch implementation.

KEYWORDS

Classifier, Random Forest, Decision Tree, K-Nearest Neighbour, Multi-Layer perceptron , Single layer perceptron

ACM Reference Format:
Hritvik Gupta #1, Nityash Gautam #2, Yash Aggarwal #3, Shubham Sharma #4, and Siddhant Poojary #5. 2018. CS235 Project Final Report: Detection of Phishing Websites. In *Proceedings of Data Mining Techniques (CS235 F22).* ACM, New York, NY, USA, 8 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Due to their increased undetectability in recent years, phishing websites have become more prevalent and constitute a bigger hazard. Additionally, because of the rise in users, these websites run the danger of disclosing sensitive user data. Phishing websites, then, are imitations of legitimate websites that use the same URL, hyperlinks, and web pages. As a result, this issue is pervasive and the current internet infrastructure is susceptible to it. We can reduce the frequency of cyber phishing attempts by using the practical technique of machine learning.

This study focuses on fundamental machine learning methods for categorizing phishing websites. In order to forecast phishing websites and to determine the most effective way for doing so, we have used a variety of fundamental machine learning methodologies. The four classification models are decision tree, random forest classifier, K-NN , Multi layer perceptron and single layer perceptron. The models are trained using a labeled dataset from the supervised classification. In order to avoid the phishing attacks machine learning models can helps to assist users in taking a more proactive role in thwarting threats and quickly responding to ongoing attacks.

CS235 F22, Fall 2022 quarter,, Riverside, CA

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

2 DATA SET DESCRIPTION

: We are using a dataset of phishing websites offered by the machine learning repository UCI. Each website in the dataset is given a label of -1 if it isn't a phishing website and a label of 1 if it is. The dataset was collected by analyzing a collection of 11000+ websites among which some were used for phishing and others not. For every website which are included in the dataset, 30 attributes are there.

3 PROPOSED METHODS

3.1 Preliminary Method 1: Multilayer Perceptron(MLP)

(1) **Overview**: A Multilayer perceptron network consists of three layers (Input layer, Hidden Layer, and Output Layer). Every layer has a certain number of nodes present in it, and except for the input layer, nodes in every layer use a non-linear activation function. MLP functions on a supervised learning technique known as Backpropagation. MLP can distinguish data that is not linearly separable.
MLPs are neural network models that work as universal approximators, i.e., they can approximate any continuous function.
The multi-layer perceptron (MLP) is another artificial neural network process containing a number of layers. In a single perceptron, distinctly linear problems can be solved but it is not well-suitable for non-linear cases. To solve these complex problems, MLP can be considered. From the network architecture presented in Fig. 5.1, it can be observed that MLP is a feed-forward neural network combined with multiple layers. Generally, it is used in the case of different machine learning purposes.

(2) **Algorithm**: (Follow this link). The Multilayer Perceptron algorithm works a step ahead of the single-layer perception, as the single-layer perceptron is limited to a linearly separable dataset whereas the multilayer perceptron handles no linearly separable datasets with high efficiency.
Multilayer perceptron carries out FIVE steps over a prespecified number of epochs.

Initializing the Weights and Biases: At this stage, the Weights and Biases are randomly initialized, with weights

in the form of a matrix whose size depends upon the architecture of the network created. The biases are generated in the form of an array whose length again is relative to the network architecture.

Feed Forward Operation: Using the weights and biases initialized in the previous step, the feed-forward operation returns the user with the value of the final output of the network, which is further used to calculate the loss. Also, this operation calculates the parameters of the intermediate layers, namely $z^{(l)}$ and $y^{(l)}$.
Here, $z->$ The output of a single perceptron
Here, $y->$ The final output of a single perception after applying the activation function
Here, $l->$ Index of the layer

Calculating the Loss: After obtaining the final output value via forward propagation, the loss is calculated and stored in an array. To calculate the loss, the binary cross-entropy loss is computed

Backpropagation Operation: In the backpropagation operation, gradients with respect to every weight and bias are computed.

Updating the Weights and Biases: Having computed the gradients with respect to every weight and bias in the network, we update the original weights and biases with the goal to minimize loss and eventually reach convergence.
The number of epochs over which the algorithm operates is finalized after training the model through different values of epochs and at what point convergence is observed.

(3) **Math**:

Initializing the Weights and Biases: We initialize the weights for $W^{(1)}$, $W^{(2)}$, and $W^{(3)}$ with random values drawn from normal distribution with zero mean and 0.01 standard deviation. We will initialize bias vectors $\mathbf{b}^{(1)}$, $\mathbf{b}^{(2)}$, and $\mathbf{b}^{(3)}$ with zero values.

Sigmoid Activation Function:

$$\varphi(z) = \frac{1}{1+e^{-z}}$$

Note that derivative of sigmoid is $\varphi'(z) = \varphi(z)(1-\varphi(z))$.

Calculating the Loss: The binary cross entropy loss can be computed by the following equation:

$$Loss_i = -y_i log y_i^{(2)} - (1-y_i) log(1-y_i^{(2)})$$

Here y_i is the true label and $y_i^{(2)}$ is the predicted label. We can average out the loss value for N number of samples as $Loss = \frac{1}{N}\sum_{i=1}^{N} Loss_i$.

Feed Forward Operation: We are using a 3 layer neural network for our approach. The input layer has 30 nodes, followed by 2 hidden layers having 12 and 8 nodes respectively. Each node will have relu activation function. The feed forward operation equations are as follows:

$$z^1 = w^1 x + b^1, y^1 = (z^1)$$

$$z^2 = w^2 y^1 + b^2, y^2 = (z^2)$$

$$z^3 = w^3 y^2 + b^3, y = (z^3)$$

Backpropagation Operation: we can write the gradient of Loss with respect to $W^{(l)}, \mathbf{b}^{(l)}$ for a single sample as

$$\nabla_{W^{(l)}} Loss_i = \delta^{(l)} \mathbf{y}^{(l-1)T},$$

$$\nabla_{\mathbf{b}^{(l)}} Loss_i = \delta^{(l)},$$

where

$$\delta^{(l)} = \nabla_{\mathbf{z}^{(l)}} Loss_i = \nabla_{\mathbf{y}^{(l)}} Loss_i \odot \varphi'(\mathbf{z}^{(l)}).$$

For the the last layer, we can compute $\delta^{(L)}$ by plugging the value of $\nabla_{\mathbf{y}^{(L)}} Loss$ as described above. For the intermediate layers $l < L$, we can write

$$\delta^{(l)} = W^{(l+1)T} \delta^{(l+1)} \odot \varphi'(\mathbf{z}^{(l)}).$$

Once we have the gradients $\nabla_{W^{(l)}} Loss_i, \nabla_{\mathbf{b}^{(l)}} Loss_i$ for all i. We can compute their average to compute the gradient of the total loss function $\frac{1}{N}\sum_{i=1}^{N} Loss_i$ as

$$\nabla_{W^{(l)}} Loss = \frac{1}{N}\sum_{i} \nabla_{W^{(l)}} Loss_i,$$

$$\nabla_{\mathbf{b}^{(l)}} Loss = \frac{1}{N}\sum_{i} \nabla_{\mathbf{b}^{(l)}} Loss_i.$$

Updating Weights and Biases: Updating $W^1, \mathbf{b}^1, W^2, \mathbf{b}^2, W^3, \mathbf{b}^3$ as

$$W^1 \leftarrow W^1 - \alpha \nabla_{W^1} Loss, \mathbf{b}^1 \leftarrow \mathbf{b}^1 - \alpha \nabla_{\mathbf{b}^1} Loss$$

$$W^2 \leftarrow W^2 - \alpha \nabla_{W^2} Loss, \mathbf{b}^2 \leftarrow \mathbf{b}^2 - \alpha \nabla_{\mathbf{b}^2} Loss$$

$$W^1 \leftarrow W^3 - \alpha \nabla_{W^3} Loss, \mathbf{b}^3 \leftarrow \mathbf{b}^3 - \alpha \nabla_{\mathbf{b}^3} Loss$$

α is the learning rate.

(4) **Key Observation**: Initially training the model over 100 epochs resulted in the model getting stuck in a local minimum which hindered the model's performance drastically. Gradually increasing the number of epochs and training the model over them resulted in optimally updating the weights and biases, hence, bringing the model out of the local minima and finally converging.

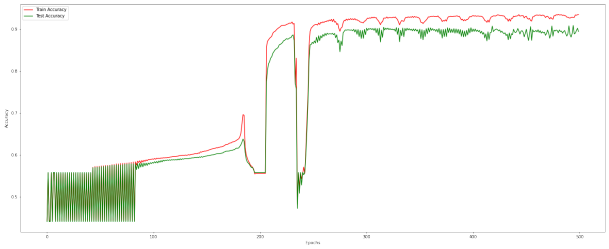


Figure 1: MLP Accuracy over Epochs

(5) **Comparing with scikit learn implementation**: The multilayer perceptron algorithm is implemented on the phishing website dataset by two means. Firstly, an off the shelf Scikit Learn's implementation is utilized. Secondly, an artificial neural network is constructed having the same architecture and parameters as that used in Scikit Learn. Functions to initialize weights and biases for forward propagation, calculating the intermediate layer parameters (outputs at every

layer), and updating the weights going as per the backpropagation algorithm have been constructed.
To visualize the model performance clearly, stratified kFold cross validation is used. The dataset is partitioned into 20 stratified sets using stratified KFold sampling. Further, an average Miss Rate of 0.05 was obtained with standard deviation of 0.03. The scratch-Implementations gave a Miss rate of 0.13, Training accuracy of 93.5%, and testing accuracy of 89%. This discrepancy arises due to two reasons. Firstly, the random initialization of the weights and biases in the self-implementation forced the model to start training with values that required to be updated over large number of epochs, in order to yield the best values for them. Secondly, hyper parameter tuning was also done in the baseline model, where the performance was tested over multiple values of epochs and batch size, which resulted in the better performance.

3.2 Preliminary Method 2: Random Forest Classifier

(1) **Overview**: The Random Forest classifier is made up of several distinct decision trees. The class with the highest votes is predicted by each individual decision tree, and that class is what we forecast. In other words, because each decision tree in a random forest classifier is unrelated to the others, these are referred to as ensemble predictions.
Random Forest ensures the behaviour of each individual tree through a method called bootstrapping aggregation. As Decision trees are very sensitive of the data, Random forest allows each decision tree to randomly sample the data from dataset such as features And by that we can ensure that each feature in phishing website dataset is equally weighted to predict the outcome.Moreover, Random forest is more stable as compared to other models in high dimensional spaces and outliers, Also Random forest is sturdy to overfitting in such cases.

(2) **Algorithm**:

1. *Select randomly M features from the feature set.*
2. *For each x in M*
   a. *calculate the Information Gain*

$$Gain(t,x) = E(t) - E(t,x)$$
$$E(t) = \sum_{i=1}^{c} - P_i \log_2 P_i$$
$$E(t,x) = \sum_{c \in X} P(c)E(c)$$

Where E(t) is the entropy of the two classes, E(t,x) is the entropy of feature x.

   b. *select the node d which has the highest information gain*
   c. *split the node into sub-nodes*
   d. *repeat steps a, b and c to construct the tree until reach minimum number of samples required to split*
3. *Repeat steps 1 and 2 for N times to build forest of N trees*

Source: https://tinyurl.com/4jpzy43w
Figure 2: Random Forest Classifier

(3) **Working**: Random Forest uses the Entropy to determine branching of nodes. In other words, Entropy is used to determine how decision tree node should be branched. Here pi is the relative frequency of class we are observing in the dataset.

$$E(S) = -p_i \sum p_i \quad \text{- (1)}$$

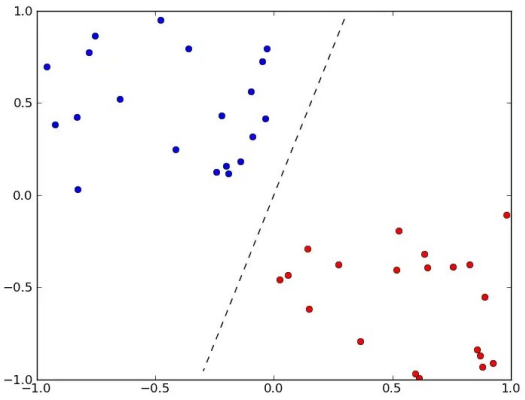
The other significant method that Random forest uses is feature selection or boot strap aggregation to evalute the importance of each node in the decision tree. Impurity in below formula is the entropy. N parameter is the number of rows of data in a decision tree.

$$n_i = \frac{N_t}{N}[impurity - (\frac{N_t(right)}{N_t} * rightimpurity) - (\frac{N_t(left}{N_t} * leftimpurity)] \quad \text{- (2)}$$

(4) **Comparing with scikit learn implementation**: We have used the random forest classifier on phishing website dataset using 2 methods. One is off the shelf baseline method where, we have imported the model using scikit learn and other is constructing the model ground which implements entropy inplace of Gini index to branch nodes in decision tree. To the model performance better on the provided dataset stratified K-fold is used. Data is separated in 5 stratified sets using stratified k-fold sampling. The performance of scikit learn model is 97 percent slightly more as compared to base implemented model which is 95 percent. This discrepancy is due to the fact that scikit learn uses Gini index and our base implemented model uses entropy to branch nodes in decision tree

3.3 Preliminary Method 3: Single Layer Perceptron

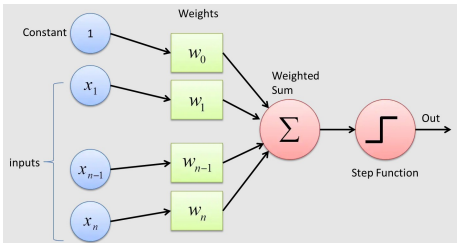
(1) **Overview** The Perceptron algorithm is one of the earliest machine learning algorithms. It is an online, linear, supervised classification algorithm for binary classifiers. An Online classifier learns a decision function based on a hyperplane by processing training points one at a time when training a model. A linear classifier achieves this classification by making a classification decision based on the value of a linear combination of the characteristics. Supervised learning is a machine learning paradigm for problems where the available data consists of labeled examples, meaning that each data point contains features and an associated label. A binary classifier is a function that can determine whether or not an input, represented by a vector of numbers, belongs to one class or the other.



Source: https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53

Figure 3: Linear Binary Classifier

Single-layer perceptrons can learn only linearly separable patterns. One line or hyperplane will divide the data points generating the patterns in a single node for a classification task with some step activation function. The perceptron was designed as a decision function for receiving inputs to produce a binary output. Its structure consists of one or more inputs, a processor, and a single output. Inputs are fed into the processor (neuron) and processed, generating an output.



Source: https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53

Figure 4: Basic Perceptron

(2) **Algorithm**

The perceptron learning algorithm tries to find a separating hyperplane by minimizing the distance of misclassified points to the decision boundary. Firstly, The algorithm is online. This means that instead of considering the entire data set simultaneously, it only looks at one example. It processes that example and then goes on to the next one. Second, it is error-driven. This means that it does not bother updating its parameters so long as it is doing well. The algorithm maintains a "guess" at suitable parameters (weights and bias) as it runs. It processes one example at a time. For a given an example, it makes a prediction. It checks to see if this prediction is correct. If the prediction is correct, it does nothing. Only when the prediction is incorrect does it change its parameters, and it changes them in such a way that it would do better on this example next time. It then goes on to the following example. Once it hits the last example in the training set, it loops back around for a specified number of iterations. The only hyper-parameter of the perceptron algorithm is maxIter, the number of passes to make over the training data. If we make many passes over the training data, the algorithm is likely to over-fit.

```
1:  w_d ← 0, for all d = 1 ... D                          // initialize weights
2:  b ← 0                                                  // initialize bias
3:  for iter = 1 ... MaxIter do
4:    for all (x,y) ∈ D do
5:      a ← Σ_{d=1}^{D} w_d x_d + b                        // compute activation for this example
6:      if ya ≤ 0 then
7:        w_d ← w_d + yx_d, for all d = 1 ... D            // update weights
8:        b ← b + y                                        // update bias
9:      end if
10:   end for
11: end for
12: return w_0, w_1, ..., w_D, b
```

Source: Hal Daumé III . (2017). chapter 4. Algorithm 5 In A Course in Machine Learning.

Figure 5: Perceptron Algorithm

(3) **Working**

Mathematically, an input vector $x = \langle x_1, x_2, ..., x_d \rangle$ arrives. The neuron stores as many weights in the weight vector $w = \langle w_1, w_2, ..., w_d \rangle$. The neuron then computes the sum to get activation. The sum is calculated as

$$a = \sum_{i=i}^{d} w_i * x_i$$

It is generally convenient to have a non-zero threshold after which the activation should fire. This can easily be achieved using a bias term. This updates the activation as

$$a = bias + \sum_{i=i}^{d} w_i * x_i$$

For Prediction we look at the sign of the activation and return 0 or 1 based on the sign. Therefore, for prediction we can use the following equation.

$$out = step(bias + \sum_{i=i}^{d} w_i * x_i)$$

The step function is given as

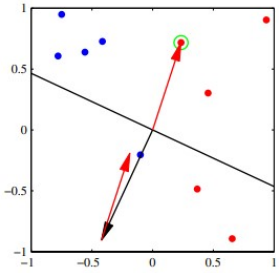
$$\begin{cases} 1 & x \geq 0 \\ 0 & otherwise \end{cases}$$

Finally, we can use this prediction to calculate gradient and update the weights using gradient descent and some learning rate (α).

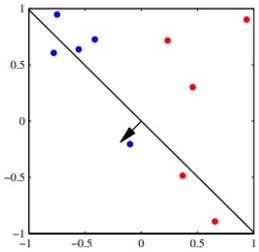
$$w_i = w_i - \alpha * grad$$
$$b_i = b_i - \alpha$$

This process is repeated for several epoch till the maxIters are reached. We can optimize the process an terminate early by setting a limit on the step size. If step size goes below a certain

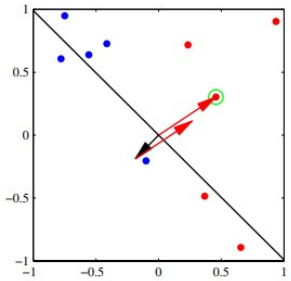
value say 10^{-5} we can terminate the program. Another way to optimize if to look into an allowed value of tolerance. In case there is not a significant improvement in error from last epoch say 10^{-7}, then we can look to terminate the program



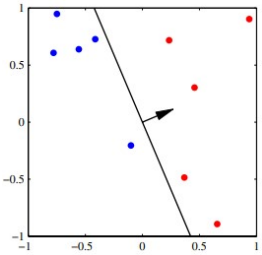
(a) Find a mislabeled point and update decision boundary accordingly



(b) Get the updated boundary



(c) Find a mislabeled point and update decision boundary accordingly



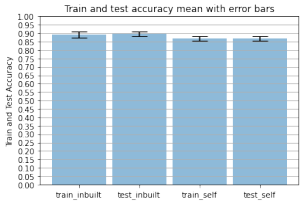
(d) Get the updated boundary

Source: BISHOP, C. H. R. I. S. T. O. P. H. E. R. M. (2016). chapter 4. Figure 4.7 In Pattern recognition and machine learning. essay, SPRINGER-VERLAG NEW YORK.

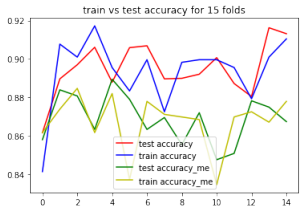
Figure 6: Working of perceptron algorithm

(4) **Comparison with scikit learn**

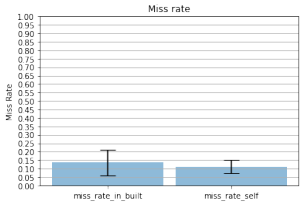
We tested our own perceptron implementation with the in-built sklearn's perceptron implementation after similar initialization and same parameters and after running both the models for 15-fold cross validation we can conclude the there is not a lot of discrepancy between the implementations. The average test train accuracy for in-built implementation was calculated to be 89.3% ± 1.3% and for self implementation to be 86.7% ± 1.3%. Further the in-built implementation had a miss rate of 13% ± 7% while the self implementation had a miss rate of 11% ± 3%.



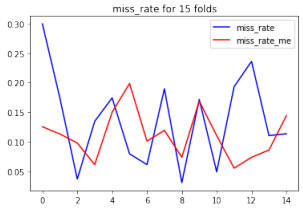
(a) Test-Train accuracy line plot for all folds



(b) Miss rate line plot for all folds



(c) Find a mislabeled point and update decision boundary accordingly



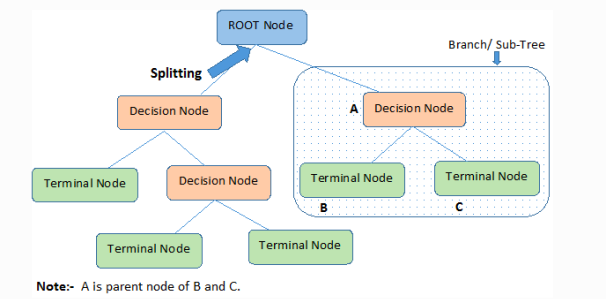
(d) Miss rate bar graph to summarize the results

Figure 7: Comparison with sklearn implementation

3.4 Preliminary Method 4: Decision Tree Classifier

(1) **Overview**: Decision Trees are supervised learning methods that can be used to solve either classification or regression

problems, however, they are typically chosen for Classification problems. It is a tree-structured classifier, where internal nodes stand in for the dataset's features, branches for the decision-making processes, and leaf nodes for the result. Each leaf node of a decision tree is assigned a class or label that represents the best suitable target value. Samples are classified by moving them from the tree's root to the leaf.
The decision tree algorithm can be represented geometrically as a collection of orthogonal hyperplanes, each of which is orthogonal to a feature axis. This is due to the fact that a decision tree divides the data based on a feature value, and this value would stay constant for each decision boundary.



Source: https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html

Figure 8: Decision Tree

(2) **Working**: A decision tree is used to partition the data space into dense regions and sparse regions which depends completely on the target variable. A binary tree can be split in two ways: binary or multiway. This algorithm will keep on splitting the tree node until the data is sufficiently homogeneous. In the end, a decision tree is returned that can be utilized to generate the optimally categorized predictions. Decision trees aim to make optimal splits between nodes which will optimally categorize the data.
In order to split the Categorical Target Variable Gini Impurity and Information Gain are used. It counts how much the entropy has changed in relation to the independent variables. In the decision tree it tells us how important a given attribute of the feature vectors is. It explains the significance of the trait. It is computed using a factor known as Entropy.

$$Information\,Gain = 1 - Entropy$$

Entropy comes from the concept of information theory. The higher the entropy the more the information content. It provides information about how much uncertainty or unpredictability there is in the data.

$$Entropy = \sum_{i=1}^{C} -p_i * \log_2(p_i)$$

Entropy is calculated for each child node in a separate split. Calculate the entropy of each split as the weighted average entropy of child nodes. Choose the split that has the most information gain or lowest entropy.

(3) **Algorithm**

GenDecTree(Sample S, Features F)
Steps:

1. ***If*** *stopping_condition(S, F) = true* ***then***
   a. *Leaf = createNode()*
   b. *leafLabel = classify(s)*
   c. ***return*** *leaf*
2. *root = createNode()*
3. *root.test_condition =findBestSpilt(S,F)*
4. *V = {v | v a possible outcomecfroot.test_condition}*
5. ***For each*** *value v Є V:*
   a. S_v *= {s | root.test_condition(s) = v and s Є S};*
   b. *Child = TreeGrowth* (S_v, F);
   c. *Add child as descent of root and label the edge {root → child} as v*
6. ***return*** *root*

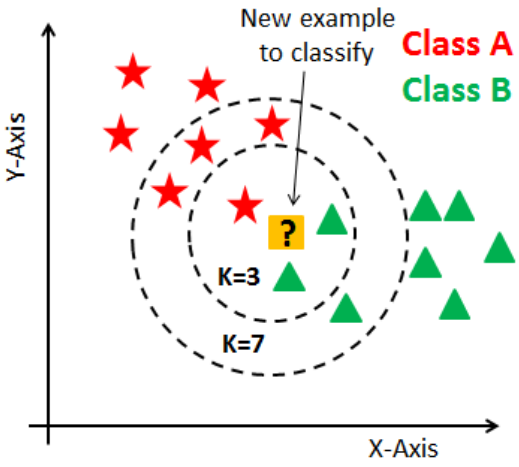
Source: https://www.researchgate.net/figure/Pseudocode-of-Decision-Tree-Algorithm$_{fig1_3}$38528758

Figure 9: Decision Tree

(4) **Comparing with scikit learn implementation**: The self-implemented Decision Tree classification algorithm is used to detect whether a website is phishing or not. In order to evaluate the model, we are using an off-the-self baseline model from scikit-learn and compare both the implementation performance measure. we compared Accuracy, Precision, Recall, and F1 score for both implementations along with area under ROC curve for 5-fold.
The off-the-shelf Model Decision Tree Classification has an average test accuracy of 95% with precision, recall and f1 score respectively as 0.94 0.94 0.94. The miss classification rate was approximately 5% and the mean ROC AUC is .96 with a standard deviation of ±1. The self-implemented Model Decision Tree Classification has an average test accuracy of 95.3% with precision, recall and f1 score respectively as 0.95 0.95 0.95. The miss classification rate was of approximately 6.4% and the mean ROC AUC is .95 with a standard deviation of ±1. Comparing both the result shows that both the model are working similar to each other with negligible discrepancy.

3.5 Preliminary Method 5: K-Nearest Neighbours Classifier

(1) **Overview**: K-Nearest Neighbours algorithms are supervised and non-parametric learning classifying methods that are used to make classifications and predictions on the basis of the proximity of individual data points. They are lazy-learning algorithms and do not make any assumptions about underlying data. Close proximity neighbors are data points that have the minimum distance from our selected data point.



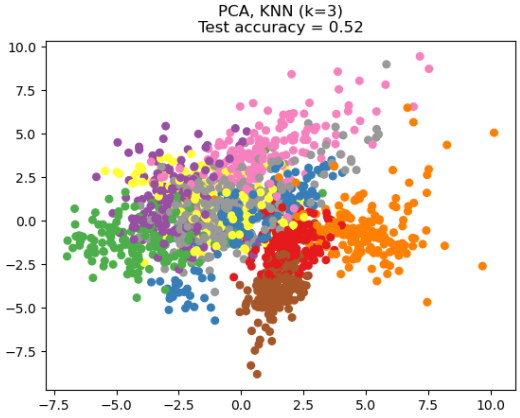
Source: https://www.kdnuggets.com/2020/11/most-popular-distance-metrics-knn.html

Figure 10: K-nearest neighbor

'K' is the number of data points that we are taking into consideration while using the algorithm. Choosing the right distance metric for a data set is crucial in ensuring that the algorithm works well. Thus, It is important to consider the K-value and distance metric during the implementation of the algorithm since they are two important factors for consideration.

(2) **Algorithm**: The working of KNN Algorithm can be described as follows:
1) Find the optimal value of K for the data set.
2) Select the appropriate distance metric for the K number of Neighbours.
3) Calculate the K-nearest neighbors according to the distance metric selected.
4) Enumerate the count of data points in each cluster.
5) Allocate the new data points to the cluster, where the number of neighbors is maximum.

(3) **Working**: KNN uses the number of neighbors provided to it. We have selected k=5 for our model as it provided us with the optimum mean accuracy. A feature representation using Principal Component Analysis (PCA) is performed on the data to find the combination of attributes (Principal components/ Feature space directions) that account for the greatest variance.



Source: https://scikit-learn.org/stable/auto$_e$*xamples*/*neighbors*/*plot*$_n$*ca*$_d$*im*$_r$*eduction.html*

Figure 11: Principal Component Analysis Representation

Furthermore, we select the Manhattan (CityBlock) Distance metric for our model. The formula for it is as follows:

$$d = \sum_{i=1}^{n} |x_i - y_i|$$

(4) **Comparing with SciKit Learn implementation**: We have used the KNN classifier on our dataset of Phishing Websites. Using SciKit Learn, we imported the Off-the-shelf baseline method of KNeighboursClassfiers. On the other hand, We implemented KNN from scratch. We used the PCA feature representation because it gave us the most accurate mean values alongside with Manhattan Distance metric. Data is separated into 10 stratified sets using Stratified k-fold sampling.
The performance of the sci-kit Learn model is 94.68% percent. The Scratch KNN Model gave us a mean accuracy of 94.75%. No Major discrepancy was seen during the implementation of the models.

4 EXPERIMENTAL EVALUATION

:

Below are the performance evaluation metrics for scratch implementation of each algorithm :

- **Perceptron**:

METRICS	Baseline Implementation	Self Implementation
Precision	0.99	0.95
Recall	0.82	0.90
F1-Score	0.90	0.92
Miss Rate	0.013	0.05

- **MultiLayer Perceptron**:

METRICS	Baseline Implementation	Self Implementation
Precision	0.9	0.89
Recall	0.9	0.89
F1-Score	0.89	0.88
Miss Rate	0.09	0.13

- **Random Forest**

METRICS	Baseline Implementation	Self Implementation
Precision	0.94	0.96
Recall	0.95	0.97
F1-Score	0.94	0.97
Miss Rate	0.04	0.02

- **Decision Trees**

METRICS	Baseline Implementation	Self Implementation
Precision	0.94	0.95
Recall	0.94	0.95
F1-Score	0.94	0.95
Miss Rate	0.05	0.06

- **K-Nearest Neighbour**

METRICS	Baseline Implementation	Self Implementation
Precision	0.95	0.95
Recall	0.94	0.95
F1-Score	0.95	0.95
Miss Rate	0.05	0.05

5 DISCUSSION & CONCLUSIONS

We will conclude with the observations gathered above of the performance of each algorithm to classify a website as PHISHING or NOT PHISHING. MISS RATE will be considered the prime inter-algorithm evaluation factor.

This is because, as per the scope of this project, we are classifying websites as phishing or not phishing. In the scenario where our model classifies a PHISHING website as NOT PHISHING has to be penalized more than otherwise.

Hence, for the evaluation of OFF-THE-SHELF implementation, the PERCEPTRON algorithm presents the lowest Miss Rate (0.013). Therefore, the PERCEPTRON Algorithm shows the best performance in the baseline implementation.

For the evaluation of BASELINE implementation, the RANDOM FOREST algorithm presents the lowest Miss Rate (0.02). Therefore, the RANDOM FOREST Algorithm shows the best performance in the baseline implementation.

6 REFERENCES

(1) SK Learn Perceptron Implementation
(2) SK Learn Multilayer Perceptron Implementation
(3) SK Learn Decision Tree Classifier Implementation
(4) SK Learn K-Neighbors Classifier Implementation
(5) SK Learn Random Forest Classifier Implementation
(6) Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review, 65(6), 386–408.
(7) Quinlan, J. R. 1986. Induction of Decision Trees. Mach. Learn. 1, 1 (Mar. 1986), 81-106
(8) Classification and Regression Trees Leo Breiman, Jerome Friedman, Charles J. Stone, R.A. Olshen (1984)
(9) Rosenblatt, Frank. x. Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms. Spartan Books, Washington DC, 1961
(10) Neural networks. II. What are they and why is everybody so interested in them now?; Wasserman, P.D.; Schwartz, T.; Page(s): 10-15; IEEE Expert, 1988, Volume 3, Issue 1
(11) Haykin, Simon (1998). Neural Networks: A Comprehensive Foundation (2 ed.). Prentice Hall
(12) Beyer, Kevin; et al. "When is "nearest neighbor" meaningful?" (PDF). Database Theory—ICDT'99. 1999: 217–235.
(13) Aggarwal, C. C. (2016). Data Mining: The textbook. Springer.
(14) BISHOP, CHRISTOPHER M. (2016). Pattern recognition and machine learning. SPRINGER-VERLAG NEW YORK.
(15) Mohri, M., Rostamizadeh, A. and Talwalkar, A. (2018). Foundations of Machine Learning. The MIT Press.
(16) Starmer, J. (2022). The Statquest Illustrated Guide to Machine Learning!!! StatQuest.
(17) Hastie, T., Friedman, J., and Tisbshirani, R. (2017). The elements of Statistical Learning: Data Mining, Inference, and prediction. Springer.