Website Phishing COMS3007 Project

Group Members

Dino Anastasopoulos: 1900661

Timothy Walters: 1855167

Razeen Gani: 1842173

Reece James Peters: 1924514

Introduction

Dataset Information, Sample Data Points and Description of Attributes

In a world where cybercrime (and particularly website phishing) is an indefatigable and often undetectable issue, we have elected to apply a number of machine learning algorithms onto a dataset that may be used to determine the legitimacy of a website.

The dataset consists of 9 attributes and 1353 data points. The goal is to predict the classification, that is whether the website is legitimate, suspicious or phishy.

Sample Data points:

	SFH	PopUpWindow	SSL_Final_State	Request_URL	URL_of_Anchor	Web_Traffic	URL_Length	Age_Of_Domain	IP_Address	Class
0	1	-1	1	-1	-1	1	1	1	0	0
1	-1	-1	-1	-1	-1	0	1	1	1	1
2	1	-1	0	0	-1	0	-1	1	0	1
3	1	0	1	-1	-1	0	1	1	0	0
4	-1	-1	1	-1	0	0	-1	1	0	1
...	...	...	...	...	...	...	...	...	...	...
1348	-1	-1	-1	-1	-1	-1	0	1	0	1
1349	-1	0	1	0	-1	0	0	1	0	-1

The attributes and their corresponding values are as follows:

- **Server Form Handler:**
  Once the user submitted his information; the webpage will transfer the information to a server so that it can process it. Normally, the information is processed from the same domain where the webpage is being loaded. Phishers resort to make the server form handler either empty or the information is transferred to somewhere different than the legitimate domain.
  - 1: otherwise
  - 0: SFH redirects to different domain
  - -1: SFH is 'about: blank' or empty

- **PopUp Window:**
  Usually authenticated sites do not ask users to submit their credentials via a popup window.
  - 1: otherwise
  - 0: rightClick alert showing
  - -1: rightClick disabled

- **Fake HTTPs protocol/SSL final:**
  The existence of HTTPs protocol every time sensitive information is being transferred reflects that the user is certainly connected with an honest website. However, phishers may use a fake HTTPs protocol so that users may be deceived, so it is recommended to check that the HTTPs protocol is offered by a trusted issuer.

- 1: use of https and trusted issuer and age ≥ 2 years
- 0: using https and issuer is not trusted
- -1: otherwise

- **Request URL:**
  A webpage usually consists of text and some objects such as images and videos. Typically, these objects are loaded into the webpage from the same server of the webpage. If the objects are loaded from a domain other than the one typed in the URL address bar, the webpage is potentially suspicious.
  - 1: request URL < 22%
  - 0: 22% ≤ request URL < 61%
  - -1: otherwise

- **URL of Anchor:**
  Similar to the URL feature, but here the links within the webpage may point to a domain different from the domain typed in the URL address bar.
  - 1: URL anchor % < 31%
  - 0: 31% ≤ URL anchor < 67%
  - -1: otherwise

- **Web Traffic:**
  Legitimate websites usually have high traffic since they are being visited regularly. Since phishing websites normally have a relatively short life, they have low web traffic.
  - 1: Web Traffic > 150 000
  - 0: 150 000 ≥ Web Traffic ≥ 50 000
  - -1: otherwise

- **URL Length:**
  Phishers hide the suspicious part of the URL to redirect the information submitted by users or redirect the uploaded page to a suspicious domain.
  - 1: URL Length < 54
  - 0: 54 ≤ URL Length ≤ 75
  - -1: URL Length > 75

- **Age of Domain:**
  Websites that have an online presence of less than 1 year, can be considered risky.
  - 1: age ≤ 6 months
  - -1: otherwise

- **IP Address in URL:**
  Using an IP address in the domain name of the URL is an indicator someone is trying to access the personal information
  - 1: otherwise
  - 0: IP Address exists in URL

- **Classification:**
  - 1: Legitimate website

- 0: Suspicious website
- -1: Phishy website

As a general rule of thumb, an attribute's value correlates with its classification. That is:

1 -> Legit, 0 -> Suspicious, -1 -> Phishy

Data Formatting and Processing

Our data was processed and prepared for use as follows:

We created an empty Pandas data frame (design matrix) with headings, 'SFH', 'PopUpWindow', 'SSL_Final_State', 'Request_URL', 'URL_of_Anchor', 'Web_Traffic', 'URL_Length', 'Age_Of_Domain', 'IP_Address' and finally 'Class' for each of the attribute columns. The data for each of these columns (which is pre-encoded) was then loaded in from a text file 'data.txt', split by a comma for each value within a line of text in the text file (see figure 3.2 for the populated design matrix). The data was then split into training, test and validation datasets which were further converted from panda data frames to NumPy arrays to implement analysis and operations on the respective data points and their values. With regards to splitting the data into training, test and validation sets, the Scikit class 'sklearn.model_selection' and function 'train_test_split' were used giving a training dataset size of 811 and test and validation dataset sizes of 271 each.

Further normalization techniques were not required as all attributes already share a common scale.

(Citation: Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.)

Machine Learning Algorithms Used

Logistic Regression

Design Brief

The first algorithm our team implemented was logistic regression. Due to the well-defined, highly discrete and categorical nature of our target variables, using logistic regression (with gradient descent and without regularization) was an easy choice since we're using different web attributes to predict the legitimacy of a website, which implied the use probabilistic outputs to make our predictions. We found that logistic regression lends itself very well to multi-classification tasks. Therefore, we applied gradient descent (the next paragraph gives an in-depth explanation of the hyperparameter values and why) with the concept of the "1 vs all" property to obtain three binary models (to represent our 3 target variables). These 3 models had their own respective sets of theta values which simplified the classification process tremendously. Finally, the classification occurred through the SoftMax function which made very intuitive sense as it multiplied each data row through each set of theta values from the binary models, obtained probabilities and classified according to the highest of the three probabilities for each data row.

Hyperparameters & Results

The 2 main hyperparameters used in this algorithm were the tolerance and learning rate. Tolerance is the value that the difference between the old and new must be greater than in order to continue the gradient descent. After doing a multitude of test runs using different values for the tolerance and learning rate, we concluded that we got the best possible performance and total test accuracy (least total test error) using the values of tolerance = 0.001 and learning rate = 0.001. While values greater than 0.001 for the tolerance and learning rate would also give very similar results, they would result in either model 2 and 3 having a low iteration count with fewer updates to the theta parameters which led to marginally lower test data accuracy and slightly higher errors. Finally, very small values (<0.001) of tolerance and learning rate, would result in the gradient descent looping almost infinitely, taking extremely long to update the theta parameters to their optimal values which was very inefficient and redundant. With the values of tolerance and learning rate = 0.001, we obtained very promising results and visuals. A training accuracy of 82%, a validation accuracy of 81% and a test accuracy of 80%. Furthermore, for binary models 1 and 2, the errors would decrease iteratively until they reached an extremely small value which was a promising observation. Model 3's error would increase over time but very marginally which may have been the cause of the 18-20% misclassification rate/error (See figures 1.1,1.2,1.3 for the error plot under tolerance = 0.001 and learning rate = 0.001.). The training, test and validation accuracies and errors are all displayed below along with a confusion matrix for the predictions on the test data where the learning rate = 0.001 and the tolerance = 0.001. (See figures 2.1,2.2,2.3 and 3.1). To conclude, if someone else were working with this data, I'd recommend that they try to minimise the errors across all 3 binary models as much as possible to get a higher accuracy, possibly by implementing regularization to this logistic regression algorithm or to use a Clustering algorithm which may work well on this data as it is highly discrete.

Diagrammatic Information

The following is under a tolerance of 0.001, and with a learning rate of 0.001

Error Plots

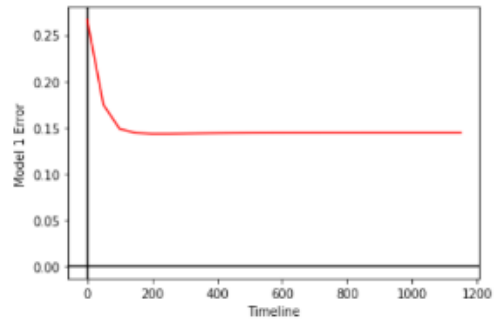


Fig 1.1 (Model 1)

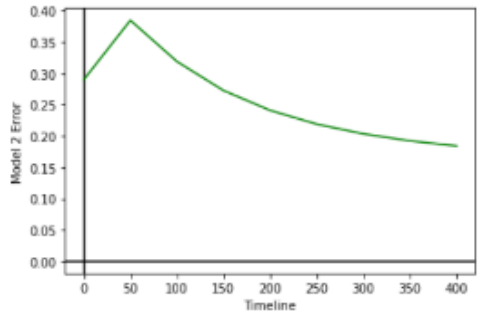


Fig 1.2 (Model 2)

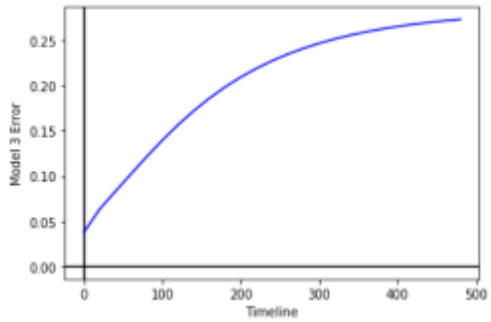


Fig 1.3 (Model 3)

Accuracy, Error and Data Analysis

```
True predictions for 0,1,-1
6
290
370
Total Correct Predictions Added Up  666

False prediction 1 when value was 0 or -1
31
32
False prediction 0 when value was 1 or -1
5
15

False prediction -1 when value was 0 or 1
27
35

Number of prediction data points: 811
Overall accuracy:  82.12083847102343
Total Misclassification Rate/Error:  17.879161528976574
```

Fig 2.1 (Training Data Analysis (Accuracy, Error, etc))

```
True predictions for 0,1,-1
1
105
114
Total Correct Predictions Added Up  220

False prediction 1 when value was 0 or -1
12
16
False prediction 0 when value was 1 or -1
0
5

False prediction -1 when value was 0 or 1
7
11

Number of prediction data points: 271
Overall accuracy:  81.18081180811808
Total Misclassification Rate/Error:  18.81918819188192
```

Fig 2.2 (Validation Data Analysis (Accuracy, Error, etc))

```
True predictions for 0,1,-1
0
92
126
Total Correct Predictions Added Up  218

False prediction 1 when value was 0 or -1
9
17
False prediction 0 when value was 1 or -1
2
7

False prediction -1 when value was 0 or 1
10
8

Number of prediction data points: 271
Overall accuracy:  80.44280442804428
Total Misclassification Rate/Error:  19.55719557195572
```

Fig 2.3 (Testing Data Analysis (Accuracy, Error, etc))

	Predicted		
	0	1	-1
actual 0	0	9	10
actual 1	2	92	8
actual -1	7	17	126

Fig 3.1 (Confusion Matrix from Testing Data)

Naive Bayes

Design Brief

For our second algorithm we chose to implement Naive Bayes. Most importantly, it is a multi-class classification algorithm. Based on thorough research we found that having an assumption of the classes of the data being conditionally independent (even though this was not the case with our data), the algorithm would perform very well with our real data. We decided to go with the Gaussian Naïve Bayes approach since its performance is claimed to be best under assumptions of normal distribution as well. With the setup of our standard deviation and mean for each class, our data tended towards a normal distribution for each class which was very good news that our results were going to be promising using the Gaussian approach! Since our outcomes were certain that it could either be legitimate, suspicious or phishy, no use of Laplace smoothing was needed for our Naïve Bayes implementation due to the extremely discrete nature of our data.

Implementation

Our approach for Gaussian Naïve Bayes was done with 6 steps. Using this step-by-step approach (Brownlee, 2020), equipped us extremely well when it was time to debug, if any, any errors were experienced with the performance of our algorithm.

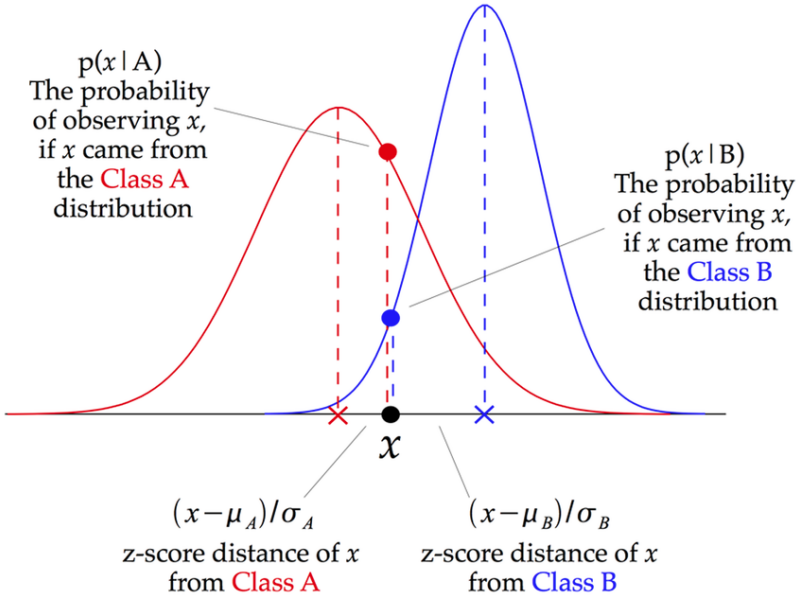
The steps are as follows:

1) Separate Data By Class.
    a. We implemented a **class_divide(train)** function that passes training data as an argument.
    b. This **class_divide(train)** function returned 3 matrices that consisted of the 3 classes our dataset was confined to.
2) Summarize Dataset.
    a. We created a **mean(Class)** function that returns the mean given a matrix of a respective class.
    b. Lastly a **std_dev(Class)** function that returns the standard deviation given a matrix of a respective class.
3) Summarize Data By Class.
    a. Using the previous steps functions, we could populate 2 matrices per class, that being one for mean and the other for standard deviation.
4) Gaussian Probability Density Function.
    a. We made a function **calc_probability(data_point, mean, stdev)** which returns the probability using the Probability Density Function when a data point, mean and standard deviation are passed within the arguments.

$$p(x_i|y_j) = \frac{1}{\sqrt{2\pi\sigma_j^2}} e^{-\frac{(x_i-\mu_j)^2}{2\sigma_j^2}}$$

(Machine Learning Algorithms - Second Edition, 2020)

5) Class Probabilities.
    a. We made use of the function **class_probabilities(class1, class2, class3, row)** which passed all 3 classes from training data and a row/observation from unseen data as arguments – in our case this was either a test or validation data row. This function multiplied all the probabilities of features for a respective row/observation together to get the shape of the Gaussian Distribution graph and returned all 3 probabilities for a respective row/observation.
    b. Lastly a **class_prediction(class1, class2, class3, row)** function which passed all 3 classes from training data and a row from unseen data as arguments – in our case this was either test or validation data. This function returned the class a respective row would most likely be part of by assigning the class with the highest probability outcome.
    c. Visually this is represented as follows:



(Smoothness without Smoothing: Why Gaussian Naive Bayes Is Not Naive for Multi-Subject Searchlight Studies, 2013)

In this visual representation, observation 'x' would classified with 'Class A' since its probability is higher than 'Class B'

6) Now for the real deal: Gaussian Naïve Bayes Prediction
    a. A function **naive_bayes_calculation(class1, class2, class3, data)** was made to bring together all steps 1-5 in one. This function passed all 3 classes from training data and a data matrix from unseen data as arguments – in our case this was either test or validation data. The function returns a matrix of our model's predictions.
    b. The **accuracy (class1, class2, class3, data)** function passes 3 classes from training data and a data matrix from unseen data as arguments – in our case this was either test or validation data. This function returns the models accuracy based off the true dataset values. This value is expressed as a percentage and since this is a closed form solution approach instead of iterative with parameters that are tweaked over time, 1-percentage accuracy will yield the error of our model.

Our Findings

In order to find a reasonably reliable accuracy and error value for our test/validation with training data, we had to run the model several times. We found that the average findings above 25 iterations seemed to be 83.72% accuracy and 16.28% error against test data, and 85.87% accuracy and 14.13% error against validation data.

Summary of our model using training data against test data with a confusion matrix:

```
Model Accuracy..
84.50184501845018
Error..
15.498154981549817

True predictions for 0,1,-1
4
106
119

Total Correct Predictions
229

False prediction 1 when value was 0 and -1
8
10

False prediction 0 when value was 1 and -1
3
2

False prediction -1 when value was 0 and 1
6
13
```

	Predicted		
	0	1	-1
actual 0	4	8	6
actual 1	3	106	13
actual -1	2	10	119

Summary of our model using training data against validation data with a confusion matrix:

```
Model Accuracy..
84.87084870848709
Error..
15.129151291512912

True predictions for 0,1,-1
4
91
135

Total Correct Predictions
230

False prediction 1 when value was 0 and -1
3
9

False prediction 0 when value was 1 and -1
6
2

False prediction -1 when value was 0 and 1
9
12
```

	Predicted		
	0	1	-1
actual 0	4	3	9
actual 1	6	91	12
actual -1	2	9	135

A few limitations to our method include no use of logarithmic probabilities which could mitigate floating point underflow issues. Furthermore, no use of multiple density functions such as Multinomial naïve Bayes, Bernoulli naïve Bayes or Semi-supervised parameter estimation in order to get a more well-rounded estimation of the model's performance and see how the model reacts with different assumptions about the distribution of attribute values and/or their relationship with the class value. (Brownlee, 2020)

Artificial Neural Network (Bonus Algorithm)

Design Brief

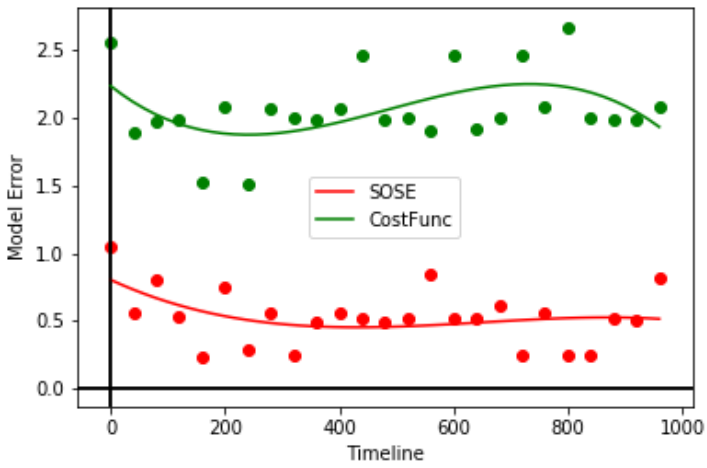
Our group's main efforts were geared towards the first two algorithms. However, we were also very interested in Neural Networks and decided to implement a simple one using our data. The network consisted of 9 inputs (corresponding to each attribute), 3 outputs (corresponding to either phishy, suspicious, or legitimate), and a hidden layer consisting of 6 neurons (the mean number of inputs and outputs)

Implementation

The data was split into 60% training data and 40% testing data, as we found that this ratio yielded the highest accuracy. No validation data was allocated as we implemented no biases. The model was trained by initializing all weights as random, and then sending the data through the network before calculating errors and adjustments and finally back-propagating.

Experimentation

This process was initially iterated 1000 times and we determined from its error (plotted below) that the optimal number of iterations was around 300.



The data's attributes and classes consist of the same values so there was no need to normalize the data. However, we attempted to change the phishy labels(-1) to 0, suspicious(0) to 0.5, and legit(1) remained as 1, as we thought that this more accurately scales the data to be either weakly (0-0.5) or strongly (0.5 – 1) correlated to legitimacy. Although, this gave us a problem in that all phishy attributes seemed to have no effect on the changing of weights rather than a negative effect as we needed, so we decided to leave the values as -1, 0 and 1

Shortcomings.

Our data set had insufficient records classified as "suspicious" and this resulted in our model being more prone to classifying as phishy or legitimate, and unable to classify examples as suspicious.

The training errors were a little bit all over the place because on every iteration we were only changing the weights. If we had added biases and regularization, the error would've decreased more

normally. However, they did gradually decrease over time. We also noticed that an excessive number of iterations does not have any benefits, and results in overfitting

Strengths

Despite the simplicity of our model, the accuracy was astoundingly high. Our final accuracy after testing was consistently between 70% and 80%. A few runs did yield accuracy below 40% but we are certain that this was due to the model being trained and tested using unevenly split data. The confusion matrix for one of our runs is shown below.

		True		
		Phishy	Suspicious	Legit
Predicted	Phishy	274	25	60
	Suspicious	0	1	0
	Legit	11	15	156

Model Accuracy: 79.52029520295203

Increasing the accuracy would be possible by adding optimizations such as momentum, and adding regularization and biases to the backpropagation - all of which we plan to add in the future.

Bibliography

Scikit-Learn:

http://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html

Dataset Information:

http://fadifayez.com/wp-content/uploads/2017/11/Phishing-detection-based-Associative-Classification-data-mining.pdf

Numpy:

https://numpy.org/doc/stable/reference/

Pandas:

https://pandas.pydata.org/pandas-docs/stable/reference/frame.html

Brownlee, J., 2020. Naive Bayes Classifier From Scratch In Python. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/naive-bayes-classifier-scratch-python/ [Accessed 25 June 2020].

O'Reilly Online Learning. 2020. Machine Learning Algorithms - Second Edition. [online] Available at: https://www.oreilly.com/library/view/machine-learning-algorithms/9781789347999/e2ab1a62-018d-46b8-a0f1-1c4a792284de.xhtml [Accessed 25 June 2020].

Research Gate. 2013. Smoothness Without Smoothing: Why Gaussian Naive Bayes Is Not Naive For Multi-Subject Searchlight Studies. [online] Available at: https://www.researchgate.net/figure/Illustration-of-how-a-Gaussian-Naive-Bayes-GNB-classifier-works-For-each-data-point_fig8_255695722 [Accessed 25 June 2020].