Importance of Feature Selection in Phishing Attack Detection

1 INTRODUCTION

Among cybercrimes, phishing can be considered one of the most successful and prevalent. Legislation encourage financial entities to educate their customers, and at the same time to implement ways to prevent their customers be fooled by this attack [1]. This fact creates the necessity of phishing attack detection applications.

Phishing is misrepresentation where the criminal uses social engineering to appear as a trusted identity. To steal data, including financial data, from users, the perpetrators make active use of fake letters from banks and payment systems, infected web pages and standard malware They leverage the trust to gain valuable information; usually details of accounts, or enough information to open accounts, obtain loans, or buy goods through e-commerce sites [2].

The nature of the webpage can be categorized according to characteristics of the URL, reputation of the domain, age of the domain among others [3]. In previous papers the researchers have used artificial neural networks to classified data and predict if a webpage is illegitimate or not getting 91.3% accuracy [4]. They also describe heuristics to consider whether a feature makes a page suspicious or not, this motivates further study of the features [5].

The main goal of this project is evaluating whether a deep neural network is suitable for this application and whether this classifier would outperform classical models of machine learning. This project also looks for the best set of features for the application. An additional goal is to evaluate whether the data would be successfully spited with an unsupervised learning model.

2 PROBLEM DEFINITION AND ALGORITHM

The dataset is available in UCI Machine Learning repository. The related work for this data uses neural network for classification using only a subset of the variables available on the dataset. This rises the question whether there is minimal subset of variables to achieve a minimum accuracy for this domain. In addition, the publication of the previous work was on 2014, four years later is it possible to design a more accurate classifier with the libraries available now?

The goal of this project is to select the best commination of set of features and classification method for our domain. While achieving this goal, the following methodologies will put on practice:

- Iterative search of the algorithm that best fits the domain. Starting with an assessment of the the suitability of a linear classifier, following with the use of non-linear classifiers.
- Use of partition of data set on: training (60%), validation (20%), and test (20%).
- The two points discussed above will be applied to different subsets of variables within the dataset.
- The model will be selected according to the accuracy get in the validation set.

This project has been an opportunity to explore:

- Deep neural networks.
- Compare the performance of deep neural networks with non-linear support vector machine classifiers.
- Scikit-learn (machine learning library for the Python programming language)
- Keras (the python deep learning library)
- Use of expert's knowledge about features to select the subset of variables.
- EM and k-means clustering

2.1 Task Definition

First, define which subsets of features or variables were used. The subset containing all variables is called sel_01. In addition, expert's assessment of features should be considered in [3], [4], [5]. The reference [4] builds a classifier using a subset of 17 variables, referred in this report as sel_02. The references [3] and [5] explain about correlation among the features or variables in the dataset, the suggested subset will be called sel_04, and slight variation of variables from he variables on sel_04 will de referred as sel_03. Finally, a minimal set of variables with high correlation with the output will be referred as sel_05. The next table shows the variables for each selection

sel_01	***sel_02***	***sel_03***
1. having_IP_Address { -1,1 }	1. having_IP_Address	1. having_IP_Address
2. URL_Length { 1,0,-1 }	2. URL_Length	2. URL_Length
3. Shortining_Service { 1,-1 }	3. having_At_Symbol	3. Prefix_Suffix
4. having_At_Symbol { 1,-1 }	4. Prefix_Suffix	4. having_Sub_Domain
5. double_slash_redirecting { -1,1 }	5. having_Sub_Domain	5. HTTPS_token
6. Prefix_Suffix { -1,1 }	6. HTTPS_token	6. Request_URL
7. having_Sub_Domain { -1,0,1 }	7. Request_URL	7. URL_of_Anchor
8. SSLfinal_State { -1,1,0 }	8. URL_of_Anchor	8. age_of_domain
9. Domain_registeration_length { -1,1 }	9. Links_in_tags	***9.*** web_traffic
10. Favicon { 1,-1 }	10. SFH	
11. port { 1,-1 }	11. Abnormal_URL	
12. HTTPS_token { -1,1 }	12. Redirect	
13. Request_URL { 1,-1 }	13. RightClick	
14. URL_of_Anchor { -1,0,1 }	14. popUpWidnow	
15. Links_in_tags { 1,-1,0 }	15. age_of_domain	
16. SFH { -1,1,0 }	16. DNSRecord	
17. Submitting_to_email { -1,1 }	17. web_traffic	
18. Abnormal_URL { -1,1 }		
19. Redirect { 0,1 }		
20. on_mouseover { 1,-1 }	***sel_04***	***Sel_05***
21. RightClick { 1,-1 }	1. having_IP_Address	1. SSLfinal_State
22. popUpWidnow { 1,-1 }	2. URL_Length	2. age_of_domain
23. Iframe { 1,-1 }	3. Prefix_Suffix	
24. age_of_domain { -1,1 }	4. having_Sub_Domain	
25. DNSRecord { -1,1 }	5. SSLfinal_State	
26. web_traffic { -1,0,1 }	6. Request_URL	
27. Page_Rank { -1,1 }	7. URL_of_Anchor	
28. Google_Index { 1,-1 }	8. age_of_domain	
29. Links_pointing_to_page { 1,0,-1 }	9. web_traffic	
30. Statistical_report { -1,1 }		

Second, preprocess data. The data available in the repository is in the format artff. For convenience the file was transformed to .csv and divided in 3 parts:

- 60% for training
- 20% for validation
- 20% for test

Third, train a Naive Bayes classifier and and set the accuracy of this classifier on the validation set as baseline for the accuracy that the non-linear classifiers should give.

Fourth, train and compare 2 non-linear classifiers:

- SVM with radial basis function kernel classifiers were train for each subset of variables and the error on the validation set as reported. This SVM was implemented using
- Deep neural networks. The task consists on storing the model that improves the epoch in which the accuracy has the best result

Fifth, select the model with highest accuracy and give a recommendation.

Sixth, check if the data can be split using EM and k-means clustering

2.2 Algorithm Definition

The Naive Bayes classifiers was trained with WEKA algorithm.

The SVM classifiers was built and trained for each subset of variables with the Scikit-learn library

```
from sklearn.svm import SVC
clf = SVC()
clf.fit(X_train, y_train)
y_pred_svm = clf.predict(X_val)
```

The deep neural networks were designed with the next considerations:

- The hidden layers will have as activation function the rectified linear unit (relu).
- The output layer will have a unique node with the sigmoid function as activation function.

These deep neural networks were built using keras library. For each subset of variables, the next steps were followed:

- Start with a single layer, set the number of nodes equal to the number of variables in the subset.
- Try adding a new layer of at least twice the number of nodes that the first layer.
- Set a limit for the number of iterations. Initialize the variable containing the maximum accuracy on the validation set: max_acc
- Train the neural net. At the end of each iteration over all samples in training data, check if the accuracy for that iteration is more than max_acc. If so, save the weights for the iteration and update max_acc.
- Until no improvement in accuracy is achieved, try adding another layer with less units, and train the net in the way described above.

The code to train the neural network and test its performance is on file model5.py. Note that I am manually indicating the number of layers and nodes in each layer.

An extract of the code

```
"""Define the architecture of the neural net
"""
model3 = Sequential()
input_len = len(selection)
input_len_iter = (input_len,)
# Add an input layer
model3.add(Dense(2*input_len, activation='relu', input_shape = input_len_iter))
# Add hidden layers
model3.add(Dense(90, activation='relu'))
model3.add(Dense(60, activation='relu'))
model3.add(Dense(40, activation='relu'))
model3.add(Dense(20, activation='relu'))
model3.add(Dense(10, activation='relu'))
#Add an output layer
model3.add(Dense(1, activation='sigmoid'))

"""Train the neural net, saving to a file the model with highest accuracy
"""
model3.compile(loss='binary_crossentropy', optimizer='adam',
              metrics=['accuracy'])
checkpoint = ModelCheckpoint(filepath, monitor='val_acc',
              verbose=1, save_best_only=True, mode='max')
callbacks_list = [checkpoint]
model3.fit(X_train, y_train,
           epochs = 200,
           batch_size = 1,
           callbacks = callbacks_list,
           verbose = 1,
           validation_data = (X_val, y_val))

"""Recover the neural net with the highest accuracy on VALIDATION dataset
"""
model_selected = load_model(filepath)
y_pred_val = model_selected.predict_classes(X_val)

"""Print the accuracy on validation set
"""
print('Accuracy on validation set:')
score_val = model_selected.evaluate(X_val, y_val,verbose=0)
print(score_val)
```

The EM and k-means clustering was done with WEKA

3 Experimental Evaluation

3.1 Methodology

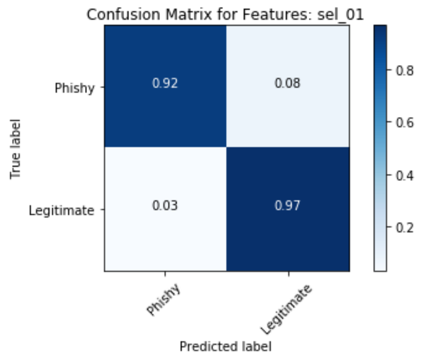
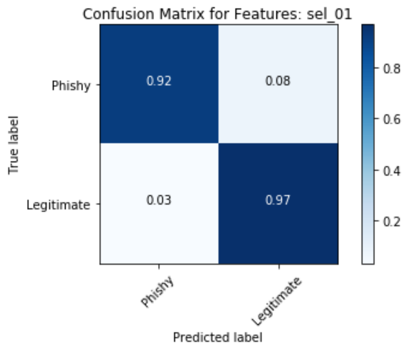
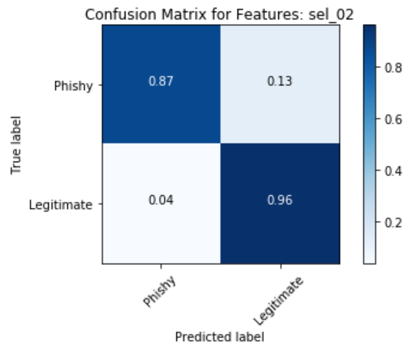
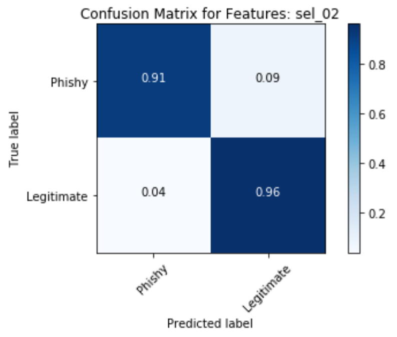
As I mention before, the evaluation criteria to select the final model is accuracy on the validation set. I will make a comparation among the models I get and select the one with the highest. I strongly believe that this will outperform the values of accuracy that the model in [4] got (91.3%) with subset sel_02 and a neural net of only one layer.

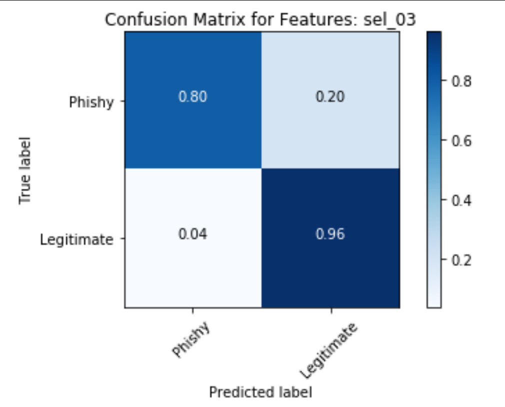
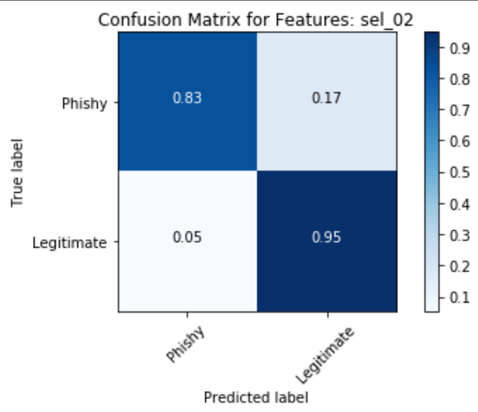
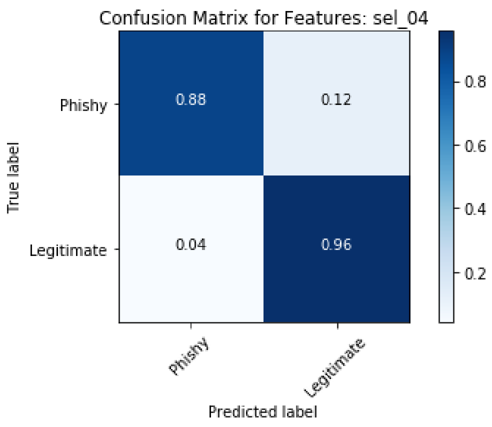
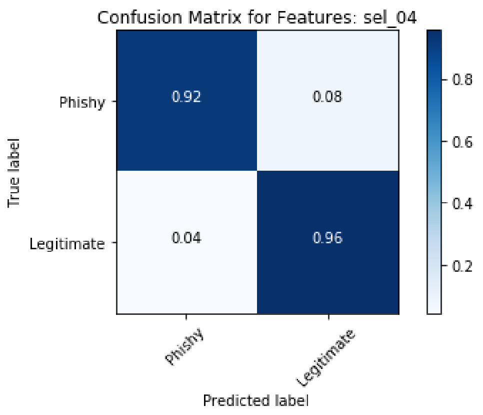
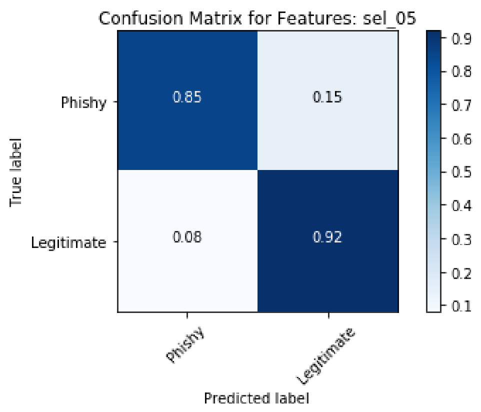
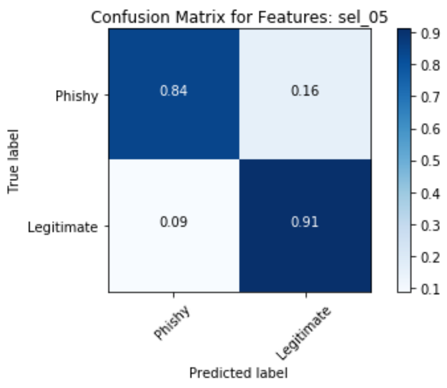
3.2 Results

The table summarize the results with Naive Bayes classifier in WEKA. Those values will the lower bound on accuracy for the non-linear classifiers to be considered as a possible candidate.

Subset of Variables	Accuracy on Validation set
sel_01	92.85%
sel_02	89.64%
sel_03	87.29%
sel_04	91.36%
sel_05	89.01%

The table below summarizes the results with the non-linear classifiers. The the neural nets the number of nodes in each layer is stated.

	SVM (Radial basis function kernel)	**Deep Neural Net**
Sel_01	 Accuracy: 94.66%	 Accuracy: 96.599% Arch: 60,60,40,20,10,1
Sel_02	 Accuracy: 91.3%	 Accuracy: 93.99% Arch: 30,50,20,10,1

Sel_03	 Accuracy: 88.33%	 Accuracy: 89.47% Arch: 9,50,20,10,1
Sel_04	 Accuracy: 92.6%	 Accuracy: 94.06 % Arch: 18,40,20,10,1
Sel_05	 Accuracy: 89.7 %	 Accuracy: 94.06 % Arch: 4,1

The values of accuracy in the validation set shown clearly that the best model for our classification problem is the deep neural net for subsets: sel_01, sel_02, sel_03, and sel_04.

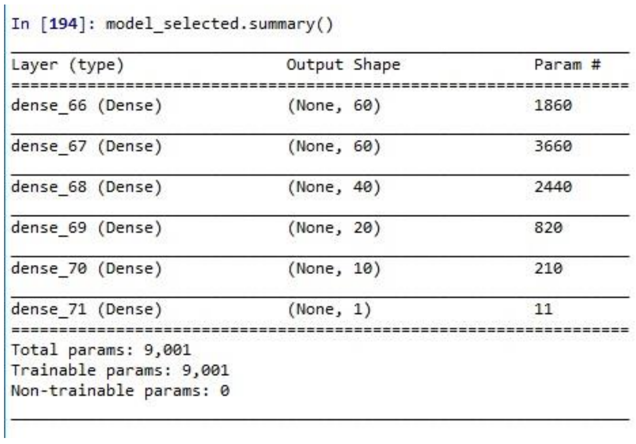
The neural net for sel_02 not only performs better than the SVM classifier I implemented for the same subset, but better than the classifier with a single hidden layer implemented in reference [4].

The final selection is the model that gives the best accuracy (96.599%). This is the deep neural net with subset sel_01 of variables. The accuracy on test set was 97.2% . The architecture is:

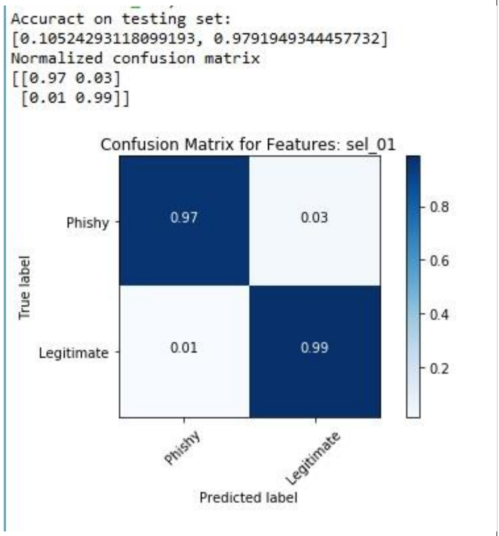
- Layer 1: 60 ‘relu’ nodes
- Layer 2: 60 ‘relu’ nodes
- Layer 3: 40 ‘relu’ nodes
- Layer 4: 20 ‘relu’ nodes
- Layer 5: 10 ‘relu’ nodes
- Layer 6: 1 ‘sigmoid’ node

In the following figures the I print the results for the selected model which was saved in: 'sel_01_weights_model_02.best.hdf5'

'sel_01_weights_model_02.best.hdf5'



```
In [194]: model_selected.summary()

Layer (type)                 Output Shape              Param #
=================================================================
dense_66 (Dense)             (None, 60)                1860

dense_67 (Dense)             (None, 60)                3660

dense_68 (Dense)             (None, 40)                2440

dense_69 (Dense)             (None, 20)                820

dense_70 (Dense)             (None, 10)                210

dense_71 (Dense)             (None, 1)                 11
=================================================================
Total params: 9,001
Trainable params: 9,001
Non-trainable params: 0
```



```
Accuracy on testing set:
[0.10524293118099193, 0.9791949344457732]
Normalized confusion matrix
[[0.97 0.03]
 [0.01 0.99]]
```

The accuracy results for clustering were:

Results for 2 clusters	k-means	EM- clustering
sel_01	52.58 %	92.61 %
sel_02	54.45 %	88.60 %
sel_03	64.28 %	87.42 %
sel_04	81.63 %	90.90 %
sel_05	63.67 %	54.46 %

We can see any of the results for clustering would classify better than the supervised learning methods.

3.3 DISCUSSION

We can clearly see the importance of the parameter highlighted SSLfinal_State. Lets look at the results of sel_03 and sel_04, both share the same features, but SSLfinal_State. It is only one variable the difference, and it implies a big improvement on the accuracy of classifying with and without considering that variable. Therfore, sel_04 is a better subset than sel_03.

The neural net with the second-best performance was the one using sel_04. This is surprising because it uses half of the variables sel_02 uses but gives better accuracy.

My hypothesis about the deep neural net outperforming non-linear SVM and the neural net with only one hidden layer in this domain was confirmed. Neural nets perform better for sel_01, sel_02, sel_03 and sel_04.

4 Related Work

Previous studies [4] showed that with a neural net of one hidden layer of thee nodes the validation set accuracy was 91.3% on epoch 1000. The results show that a deep neural network can outperform this the previous result on this domain.

5 Future Work

[4] shows an interesting approach on self-structuring a neural net. The model proposed by the authors started with a minimal hidden layer (1 node). Then this hidden layer self-structure itself by adding a node at the end of the training, while the evaluation is not good enough. Based on the previous approach I would automate the process of adding layer to my deep neural net.

Although I shown the confusion matrices in the results, I did not use them to select the model. However, the phishing detection problem an alert application, and alerts are evaluated with confusion matrix and F1 scores. I would like to do this comparation.

6 Conclusion

The non-linearity of the deep neural networks proved outperform the classical methods for the application of phishing detection.

The best set of features is sel_01, and the second sel_04. The best classifier for the application is a deep neural network.

My final recommendations for developing a phishing detection application is:

- Set as minimum data required sel_04, and If it is not possible for the application to get the 30 variables (sel_01), apply the deep neural net model implemented for sel_04 for the classification task.
- Whenever the application can get the complete data, use the deep neural net model implemented for sel_01.

We will always need labeled data on training the phishing detection application model because clustering does not provide enough accuracy.

BIBLIOGRAPHY

[1] Neil Daswani, Christoph Kern, and Anita Kesavan. Foundations of Security: What Every Programmer Needs to Know. Apress .Berkeley, CA ,USA, 2007

[2] Ross Anderson. Security Engineering: A Guide to Building Dependable Distributed Systems, Second Edition. John Wiley & Sons. 2008

[3] Mohammad, Rami, McCluskey, T.L. and Thabtah, Fadi Abdeljaber (2012) An Assessment of Features Related to Phishing Websites using an Automated Technique. In: International Conference For Internet Technology And Secured Transactions. ICITST 2012. IEEE, London, UK, pp. 492 497. ISBN 978 1 4673 5325 0

[4] Mohammad, Rami, Thabtah, Fadi Abdeljaber and McCluskey, T.L. (2014) Predicting phishing websites based on self-structuring neural network. Neural Computing and Applications, 25 (2). pp. 443 458. ISSN 0941 0643

[5] R. B. Basnet, A. H. Sung and Q. Liu, “Rule-Based Phishing Attack Detection,” in Proceedings of the International Conference on Security and Management-SAM'11, Las Vegas, NV, USA, 2011.

[6] Phishing Websites Features. This document is part of the dataset in UCI

[7] OWASP