IN3062: Introduction to AI
Exploration of the effectiveness of varying machine learning models on a phishing-based dataset.

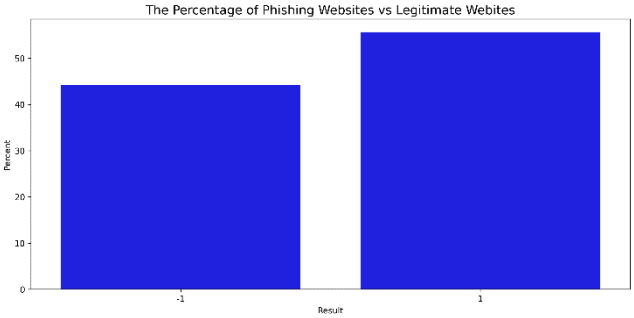
Zainab Mayet
Vijay Kesireddy
Andreas Salcedo-Cespedes
Shiven Saini

Link to github: https://github.com/saini-shiven/IntroToAIGroup1

1. Introduction

The "domain" we chose was that of Cyber Security - specifically "Phishing" and the classification of a website's URL as one that points to a phishing or non-phishing website. Our dataset is a collection of features of various website URLs and whether they're indicative of a phishing website or not. This is represented by one of three values: -1 for "indicative of a phishing website", 0 for "suspicious" and 1 for "indicative of a non-phishing website". These features include classifiers like “having_IPhaving_IP_Address” (If an IP address is used as an alternative of the domain name in the URL, such as “http://125.98.3.123/fake.html”), “double_slash_redirecting” (examining where in the URL a “//” appears, as its position is indicative of if the URL is secure or not), etc.
“Phishing” refers to the practice of sending fraudulent snippets of digital communication (SMS messages, emails, phone calls, etc.) with the purpose of inducing individuals into revealing sensitive personal information – be it anything from passwords to credit card numbers (phishing.org, n.d.). The main questions we’d be exploring in this report would be: *“What features of a URL (if any) are the most useful for predicting if said URL points to a phishing website?”*, as well as *“What is the most effective model for predicting if a URL points towards a phishing website?”*. From these questions, we aim to achieve the production of an array of effective AI models that can generalise our dataset to a high degree of accuracy – as well as figuring out what features are the most important for each model’s generalisation. The outputs we expect differ from model to model (the specific models we’re going to be using will be expanded on more in our “method” section), but the ideal accuracy for each will be anything above 90%.

As said above, an initial run of our dataset showed that all features for each URL (or row) have been scored one of three ways; -1 for “indicative of a Phishing URL”, 0 for “suspicious” and 1 for “Indicative of a non-phishing URL”. We found that “0” didn’t appear nearly as much as “-1” or “1” (due to this, we’ll investigate what removing “0” and replacing it with “-1” will do for our model’s accuracy), and no 0’s were present in the “results” column. We found that out of the 11055 entries present in the dataset, 56% were classed as “non-phishing” whilst 44% were classed as “phishing”. We took this as a good sign, as a close to equal result dataset wouldn’t present many challenges in terms of biases/skews. We also looked through our dataset for any missing values, or repeating rows; we found our data contained neither “NaN”/Non-applicable data (essentially any data that doesn’t fit into the 1, 0, -1 scoring system), nor any repeating/duplicate entries (I.e. rows). Looking through each feature individually (by way of excel data manipulation tools and bolstered by the ‘.value_counts()’ function) we found that the most important feature for classifying something as “non-phishing” was “having_sub_domain” (whether or not the URL pointed to a website under a sub-domain) – out of the 6157 total URLs classified as a non-phishing website, 5630 were scored as “indicative of a non-phishing website”. Adversely, all the 4898 total URLs classified as phishing were scored as “indicative of a phishing website” in the “double_slash_redirecting” feature. Subsequently, we’ll focus on how the exclusion of these two features contribute to our models’ accuracy.



2. Method

One of the methods we decided to use that allows for the solving of regression and classification problems is known as K-Nearest Neighbor (KNN). KNN includes advantages such as the implementation being easy and quite straightforward. Additionally, a model prior to implementation isn't required to be built, thus not needing to tune several parameters or determine further assumptions. KNN is also quite versatile as it can be used for data that is either regression or classification. However, one fall back to this model is its computation time seems to be proportional to the number of features that is acting on. The model will have access to our phishing dataset which would ultimately use it output a result. In this case the model would analyze a new piece of data input and look at its surroundings data to produce a result - i.e, is it phishing or not. For this model the dataset will be kept in its raw form and will analyze the different columns whilst initiating a target, in this case 'result'. The data will need to ensure it is 'fitted' correctly to its data, meaning the line of best fit should run through the data as opposed to it being spread and omitted as a whole. Finally, when the code is tested a prediction table will be displayed and the accuracy will also be shown. Judging by the accuracy we could conclude whether the model is ideal for the dataset and can be used to predict whether a case is phishing or not.

Random Forest is another supervised machine learning algorithm which expands and combines numerous decision trees essentially creating a 'forest'. Once again, this model can be used for both regression and classification problems. Some advantages include its efficiency, simplicity (especially users that are beginners to SML and modelling), accuracy and versatility. A drawback to random forest can be its need for mass amount of memory (thus resulting in a large computation time). Since it uses a variety of decision trees much more memory is required thus decreasing its speed. In conclusion although it is slow it produces increased accuracy in its results. For our project we will have the model take in the phishing dataset and select all columns aside from the target ('result'). Unlike KNN, Random Forest will take into account all pieces of data. Using this, the model should be able to calculate whether the case is phishing or not. Once again, the model produces a line of best fit in accordance with the data. To evaluate this model, we will observe the accuracy, this well give us a clear indication of how useful the model is and much it can be relied on to create predictions

The next supervised machine learning algorithm is Support Vector Machines (SVM) which we used for our classification dataset. There are a variety of advantages for SVM models. Firstly, it is able to establish the best separating hyperplane in order to submit an accurate prediction. It is also ideal for high dimensional data and is relatively memory efficient. Finally, as it is specialized for classification problems it integrates well with our dataset. Disadvantages can include that the model does not work very well with significantly bigger datasets. It may also prove to be not very accurate in comparison to other models as our data contains a large number of overlapping data points. The high margins will require numerous calculations to take place. The task of this model is to take into account two parameters to produce an output, 'phishing' / 'not phishing'. From this task, we will be able to observe the effectiveness of the SVM model when tested against this large dataset. The dataset was prepared by searching for missing or corrupt values and removing them where applicable. We must also experiment with which kernel to use on the model**.** To evaluate this model, we will be assessing its accuracy against both our original dataset and our edited dataset (where all 0's have been replaced with -1's).

Another model used in supervised learning is perceptron. It aids in solving classification problems and helps classify data that's been input. The advantage of perceptron is that it can help solve non-linear problems. It can work with both big and minimal datasets whilst having the accuracy remain constant. It can also rapidly produce a prediction after its training. For muti-layered perceptron models the computations may become quite time consuming - thus, we'll consider only using a single-layer perceptron. On top of this, being

able to predict how the dependent variables affects an independent variable becomes quite tricky. In addition to the previous drawbacks, the model's ability to generalize depends heavily on the training's quality/quantity. This model should be able to produce a high level of accuracy whilst predicting against our test set**.** To prepare the dataset for this model, we performed an initial run of the input values - perceptions are suited to large datasets so not much had to get cut down, we just had to ensure the dataset was clean (without missing/corrupt values)**.** To evaluate this model, we'll have it perform it's training twice; once against the original dataset, and once against a dataset where all "0"s have been replaced with "-1"s - it's effectiveness will be evaluated against its ability to surpass our set success rate of 90%.

Decision Trees used in supervised learning are very popular with classification problems as well as predictions again, being quite ideal for our chosen dataset. Decision trees can be compared to flow charts in the sense that a step-by-step procedure is taken before reaching the next appropriate node. Using decision trees is useful as minimal computation is required to produce a result. It is also suited to categorical variables which our dataset is consistent with. Some disadvantages can include common errors when there are several classes and limited training data. With our dataset we can assume there will be numerous branches as there are various characteristics. This model should be able to perform well with our feature-rich dataset. From this model, we will be able to observe a decision tree being created that can generalize the data and make predictions with decent accuracy. We will also see how the accuracy of predictions is affected by using different hyperparameters such as *gini* and *entropy* splitting criteria. The dataset was prepared by dropping the *id* column as it is unique for every datapoint and checking for any missing values. To evaluate this model, we will generate confusion matrices to visualize the proportion of true predictions against false ones and make use of *sklearn*'s metrics library by outputting the accuracy score for each test.

Naïve Bayes is another supervised machine learning algorithm used for a variety of classification problems. The advantage of naïve bayes is that it isn't very complicated and is very quick in producing a result whilst working well with text classification. With a real-life example naïve bayes isn't very ideal as features are not necessarily independent therefore making this algorithm less accurate. This model should be able to accurately predict whether a data sample is classed as a phishing website or not based on prior knowledge about the class label. From this model we will be able to observe predictions made "naively" with less regard to relations between features. The dataset was prepared by dropping the "id" column and our two most biased columns. Another training and testing process will be done with data that contains -1s replaced with 0 on all features. We have the option of two main variants of the model, Gaussian and Bernoulli. The latter being optimal for binary data and should produce optimal results. To evaluate this model, we will produce a confusion matrix and compare the accuracy of our results to our target of 90%.

3. Results

Perceptron

The perceptron model initially seemed most suited to our data due to its suitability to boolean values (unsw, n.d.) - our dataset is primarily binary values (1 representing a non-phishing URL and -1 representing a phishing URL).
We encoded our data by using all columns and features, splitting the test data and training data to a 1:4 ratio. Training this model pre-standardisation (standardisation being the process of converting data to a common format to enable users to process and analyse it (Sisense, n.d.)) only produced a 55% accuracy (see initial confusion matrix) - due to this, coupled with the performance of our other models, we've set our success criteria to any accuracy above 90%. However, once standardisation and "K-fold" cross validation (the exercise of training several machine learning models on subsets of the training data and evaluating their generalisation on the test data) were applied, accuracy rose to between 90% and 95% for each fold (see appropriate confusion matrix). Note that our "k" count (the number of subsets we used) was 5, and raising that value yielded no significant increase in accuracy. Utilising this model against our edited dataset (i.e., replacing all 0's with -1's) doesn't yield much difference in accuracy, just 1% less accurate.

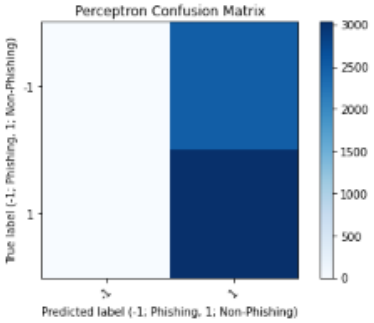


Figure 1: Perceptron confusion matrix pre-standardisation

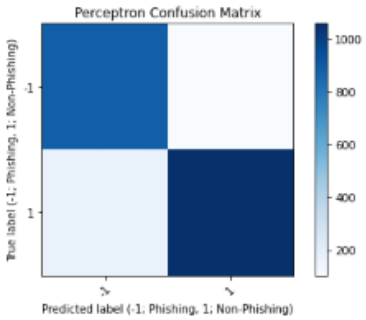


Figure 2:Perceptron confusion matrix post-standardisation with validation

SVM

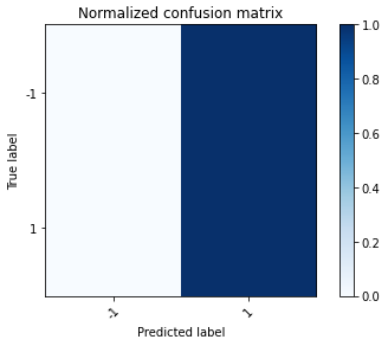


Figure 3: SVM confusion matrix with a linear kernel

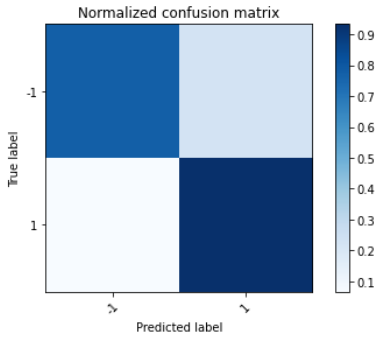


Figure 4: SVM confusion matrix with an RBF kernel

The SVM model seemed one well suited to our data due to its suitability for classification - specifically high dimensionality data (Solarzano, 2019), which our dataset is.
We encoded our data by using all columns and features, splitting the test data and training data to a 1:4 ratio. Originally, we were using a 'linear' kernel on my entire dataset with a test sample of 50%. This only yielded a 55% accuracy, which we've found was mainly due to the size of our dataset; the SVM model is not suited to large datasets with too many overlapping features (Laparna, 2020), which unfortunately is a characteristic of our data. To combat this, we took a slice of our data (reducing 11055 items to 400), and re-ran the model with an 'rbf' (Radial Basis Function) kernel. The sliced data was initially a random sample of our entire dataset, but this greatly reduced our accuracy (making it more like our original 50%). To combat this, we took a slice of data that was more indicative of our complete dataset; the complete dataset's final column (result) had a proportion of 44:56 for 1 and -1 values respectively. The 400 entries we had selected had a proportion of 42:58 of the same values. As these are similar, we decided to press on with the 400 selected values rather than the randomly selected values. This produced a model that had an 86 - 90% output accuracy (see appropriate confusion matrices). Again, we encoded our variables via dropping all columns bar "results" and using that as our X and using the results column as our Y (our

Target). Utilising this model against our edited dataset (replacing 0's with -1's) further improved its accuracy, raising it from ~87% to >90%.

K - Nearest Neighbor

The k nearest neighbor model is ideal to display classified data. This model takes the data closest to the pointer and assigns a group to the variable based on the values nearest to it. During implementation we had set the KNN length to 12, this had produced a mean accuracy of 53.7%. As this result isn't very accurate, we decided to trial run different methods to increase accuracy. Firstly, we began by increasing the KNN length from 12 to 15. We thought by having more data, the machine would be able to take it into account several more factors to produce a greater accuracy, this in-fact worked as the accuracy had improved to 54.3%. We then decided to set the length to 17, seeing if the accuracy could improve even more. However, this decreased the accuracy by 0.2 making it 54.1% in total. Thus, we concluded in keeping the length as 15, as it produced the highest accuracy. As our dataset consists of three possibilities 1 = 'non-phishing' 0 = 'maybe phishing' and –1 = 'phishing', we thought it would be interesting to see whether limiting the possibilities to two could perhaps increase the accuracy. This gave us an accuracy of 54%, if the KNN length was 12 then this would have been quite an improvement - however since the length was set to 15 the accuracy had ultimately decreased. Finally, the last test made was to remove two features from the dataset being 'having_Sub_Domain' and 'double_slash_redirecting'. Overall producing an accuracy of 53.4%, being the lowest, thus concluding that having a slightly higher KNN length can assist in producing the highest accuracy mean as well as highlight the importance of the aforementioned features in classifying a phishing/non-phishing URL.

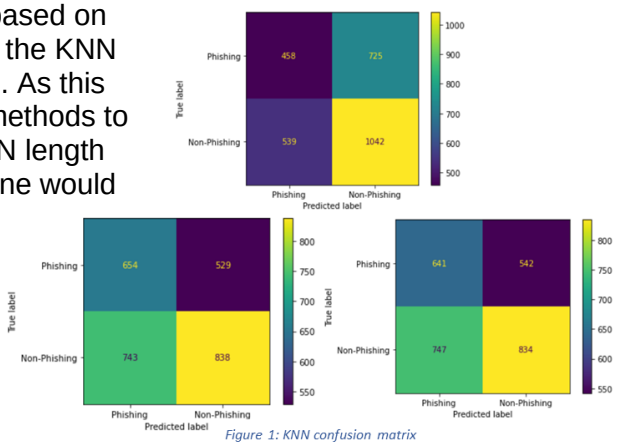


Figure 1: KNN confusion matrix

Random Forest

Random forest model produces some of the highest accuracy figures in comparison to other models. This statement was proven to be true from my implementation and findings. Firstly, this model was implemented with the original dataset which produced an incredibly high accuracy being 96.3%. Similarly to the KNN model we wanted to test whether this accuracy could be increased further. We proceeded to test it with the dataset that omits '0's. To our surprise this resulted in a slightly less accurate mean of 93.8%, implying that this model requires as much data in order to make an accurate verdict. Finally, we tested it against the dataset without the two features 'having_Sub_Domain' and 'double_slash_redirecting', during research stages we concluded that the two features had a big impact in deciding whether a case was phishing or not thus producing least accurate mean (93.5%). This shows that the features do in fact play a role in producing better accuracy means as not having them resulted in a poor output. My findings showed us that this model works best when given all possibilities. The algorithm clearly has procedures in place that lead it to be optimized when as much data as possible is fed into it.

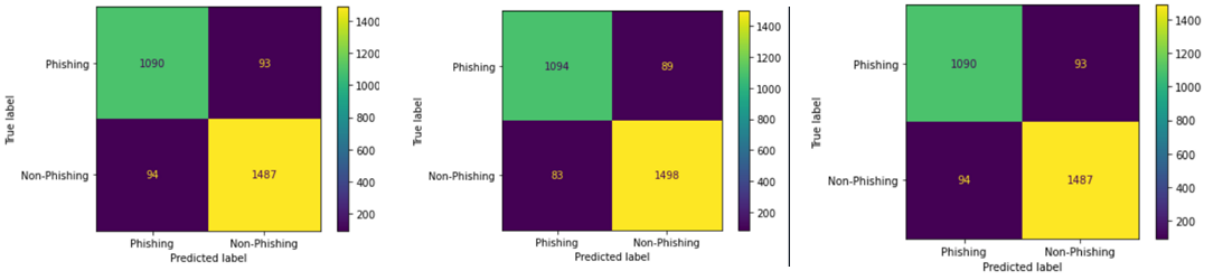


Figure 2: Random Forest confusion matrix

Decision Tree

We initially predicted the decision tree to be among the less effective models for our dataset as although easy to interpret compared to the perceptron or support vector machine models, they can be prone to overfitting if the training data size is not large enough. The model is primarily used for classification rather than predicting continuous values and therefore performed well with our feature-rich dataset.

We trained the model five times using different hyperparameters and encoded input variables. There are two criteria for deciding on how to split a decision tree, entropy and gini, both of which produced very similar results of around a 0.96 accuracy score. Due to the binary nature of our dataset, the standard scaler also produced almost identical results with an accuracy score of 0.963. Combining these configurations with the random splitter as opposed to the default best splitter also produced identical results. Figure 3 shows the confusion matrix most produced during these tests. Dropping having_Sub_Domain and double_slash_redirecting had little effect on the results, with accuracy dropping to 0.933 (figure 4). Testing this model against a dataset where all 0's have been replaced with -1's also yielded an accuracy of 0.933.

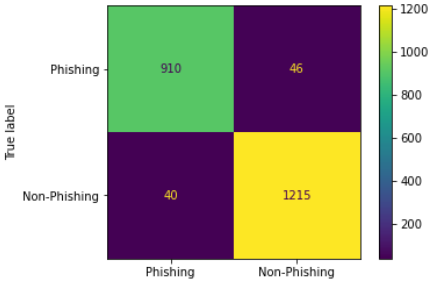


Figure 3: Confusion matrix for decision tree trained with default dataset and confiauaration.

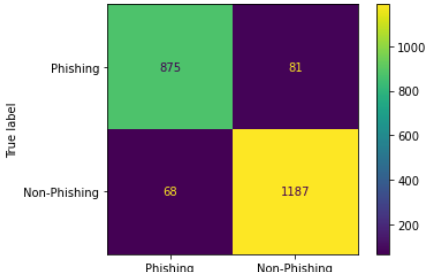


Figure 4: Confusion matrix for decision tree trained with edited dataset.

Naive Bayes

This model was trained four times through the course of our exploration. The Gaussian model was initially used and had an initial accuracy score of 0.58 (See figure 5) which increased to 0.69 (figure 7) by omitting having_Sub_Domain and double_slash_redirecting columns and retraining it.

Out of the two variants, Bernoulli had the highest accuracy score of 0.9 (figure 6). This was expected due to it being suited for binary values. Further tests with this classifier included training it with the edited dataset which consists of only 1s and -1s produced results with no notable change to the accuracy score.

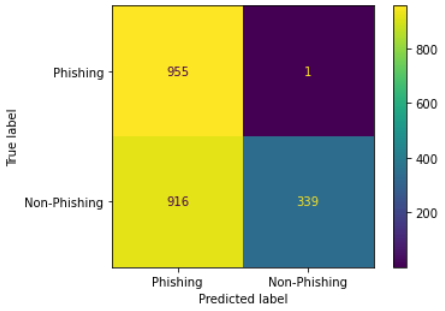


Figure 5: Confusion matrix for Gaussian Naive Bayes with default dataset and configuration.

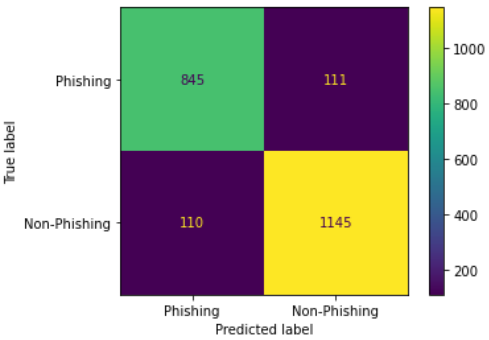


Figure 6: Confusion matrix of Bernoulli Naive Bayes with default dataset

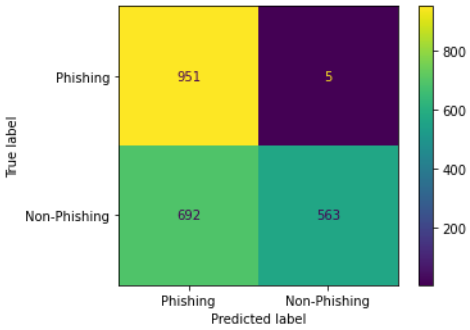


Figure 7: Confusion matrix of Gaussian Naive Bayes with edited dataset

4. Evaluation and Conclusion

Our dataset provided very few formatting challenges - however, the non-continuous nature of the datapoints affected our predictions for certain models, notably the linear regression model (which was expected to perform poorly) achieved our success criteria of 90%, with 92% accuracy without any major data processing beforehand. Most of our experimentation with the data thus centered around dropping whole columns before splitting the dataset into training and validation sets. As previously mentioned, several overlapping features proved to be a problem when trying to determine features to drop and experiment on. We found two optimal columns to drop (*having_Sub_Domain* and *double_slash_redirecting*) which were found to have strong correlations to phishing websites (non-phishing and phishing respectively). This change was also applied alongside standardization attempts by changing all 0s in the dataset to –1s. Notable accuracy improvements were observed in the support vector machine and Naïve Baye models with an 87% to 92% and 59% to 68% increase respectively. In other models, results either did not change or decreased slightly.

In the end, we were left with results of varying degrees of accuracy for each model. Most of our models achieved our set accuracy target of 90%. Among these, the perceptron and support vector machine models which had performed poorly on our initial dataset but worked better in further tests resulting from exploration of our data. The models that performed best with our dataset were the random forest and decision tree methods, both predicting results with 97% and 96% accuracy respectively on the unedited dataset (save for the id column being dropped). These top models center around the splitting of features into categories. Our data, being largely binary in nature, facilitated this process along with the considerable number of features.

Returning to our initial questions, the use of decision trees proved to be the most effective model for predicting whether a URL points towards a phishing website. Given that its ensemble method, the random forest is used to improve accuracy, this can be considered the optimal model to use. Through our subsequent evaluations of the dataset, we identified features relating to aesthetic characteristics of the URL - namely whether a subdomain is present instead of just a public IP address and whether there are double slashes “//” present – were the most important features for identifying what constitutes as a phishing URL. Finally, as mentioned above, testing our models against an edited dataset (replacing 0’s with -1’s) did not produce a notable change in accuracy save for the SVM and Naïve Bayes models.

Finally, due to our 11055 entries, we chose the holdout validation approach for most of our tests. Following the successful 32% increase in accuracy for the perceptron model after applying k-fold cross validation, it would be suggestable to test this approach with other models.

References

Laparna, G. (2020, December 22). *Support vector machine in machine learning*. Retrieved from Geeksforgeeks.org: https://www.geeksforgeeks.org/support-vector-machine-in-machine-learning/

phishing.org. (n.d.). *What is phishing?* Retrieved from Phishing.org: https://www.phishing.org/what-is-phishing

Sisense. (n.d.). *Data Standardization*. Retrieved from Sisense.com: https://www.sisense.com/glossary/data-standardization/

Solarzano, M. (2019, May 16). *Working with high dimensional data*. Retrieved from medium.com: https://medium.com/working-with-high-dimensional-data/working-with-high-dimensional-data-9e556b07cf99

unsw. (n.d.). *Single Layer Tutorial*. Retrieved from http://www.cse.unsw.edu.au/: http://www.cse.unsw.edu.au/~cs9417ml/MLP1/tutorial/singlelayer.htm