**Detecting SSH brute force attacks using machine learning algorithms**

SSH is the widely used protocol. It allows to execute remote commands securely. For example, SSH is indispensable when the task at hand is to configure an instance of Linux server. Although, authentication in SSH can be accomplished using public keys, typically users employ passwords for that purpose. If the password is simple enough, attacker may attempt to perform a password recovery using so called brute force attack. In this situation, attacker will launch consequent connection attempts and try to enter next unused password from a dictionary. In this short article, we try to investigate whether we can spot such attacks using machine learning approaches.

Here, we are going to demonstrate the usage of Support Vector Machine, neural networks, and naïve Bayes classifier. We are going to use real TCP traces and we are going to predict whether SSH connection attempt is a brute force attack or not using aforementioned algorithms.

The structure of this document is the following. First, we are going to discuss the background material related to SVM, Naïve Bayes classifier, and artificial neural networks. Second, we are going to discuss the contents of the dataset. Third, we are going to show how we preprocessed the data and selected the features. And finally, we are going to discuss the results.

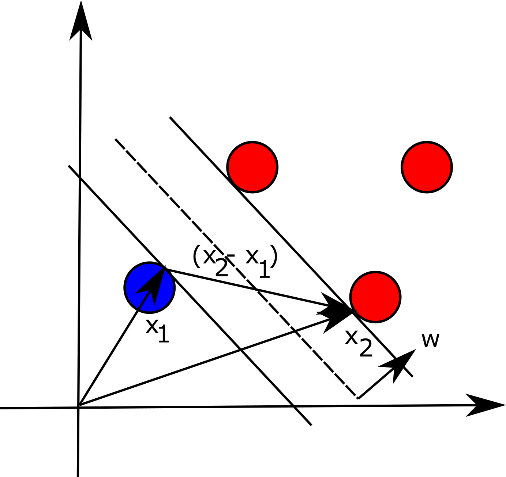
**Algorithms**

The number of machine learning algorithms which can be used for classification is rife: logistic regression, k-nearest neighbors, k-means, SVM, neural networks, naïve Bayes classifier, decision trees and random forests are just few. And in this article, we are going to demonstrate the results for SVM, neural networks and naïve Bayes classifier.

**Support Vector Machine (SVM)**

Support vector machine is a powerful tool, and we believe that every data scientist needs to know the internals of this machine learning algorithm. We are going to discuss the idea behind this classification algorithm in the proceeding paragraphs.

Given a hyperplane **wx**+b=0we can define two additional hyperplanes that are parallel to and equidistant from the given one. Namely, **wx**+b=1and **wx**+b=-1. If we say that we have a set of points (xi, yi)where yi is either one or negative one (depending on the class to which the sample belongs), we state that the following must hold yi(**wx**+b)-1**≥**0, with equality occurring when points are exactly on hyperplanes. We visualize the problem for the 2-D setting in the figure below:



Now if we take two datapoints, one on each hyperplane parallel to and equidistant to **wx+b=0**, **x1** and **x2,** the margin between these two points can be expressed as follows (basically, the difference between the two vectors projected onto unit vector **w**, which is perpendicular to hyperplane that separates the points in different classes):

According to SVM, the goal is to maximize this distance to reduce the classification error. However, maximizing this expression is the same as minimizing the following:

Since we are given constraints in a form yi(**wx**+b)-1**≥**0we can use Lagrange multipliers and construct the following equation:

To minimize this function, we need to find the partial derivatives with respect to vector **w** and b:

If we now plug in the values for **w** into the original equation and reduce it (here we also use the result for the second partial derivative), we obtain (dual optimization problem):

s.t.

We can now solve this maximization problem using Quadratic Programming (QP) solver and obtain Lagrange multipliers. Once this is done, we can find the vector **w** and bias term b. For the last point, we can take a point on the hyperplane:

The derivations that we have presented are good only for linearly separable samples, however, one can use so called kernel trick to classify more complex datasets. We leave out the discussion of the different kernels in here, but we will return to this issue when we will be playing with the empirical data. The advantage of the classifier is that other than the kernel function it does not have any other hyperparameters. Hence, tweaking this classifier is rather straightforward.

**Naïve Bayes classifier**

Naïve Bayesian classifier is a simple approach to classify samples into multiple classes. Assume that we have vector **Y** of classes and vector of features **X**, we can then use the Bayes’ formula to find conditional probabilities as follows:

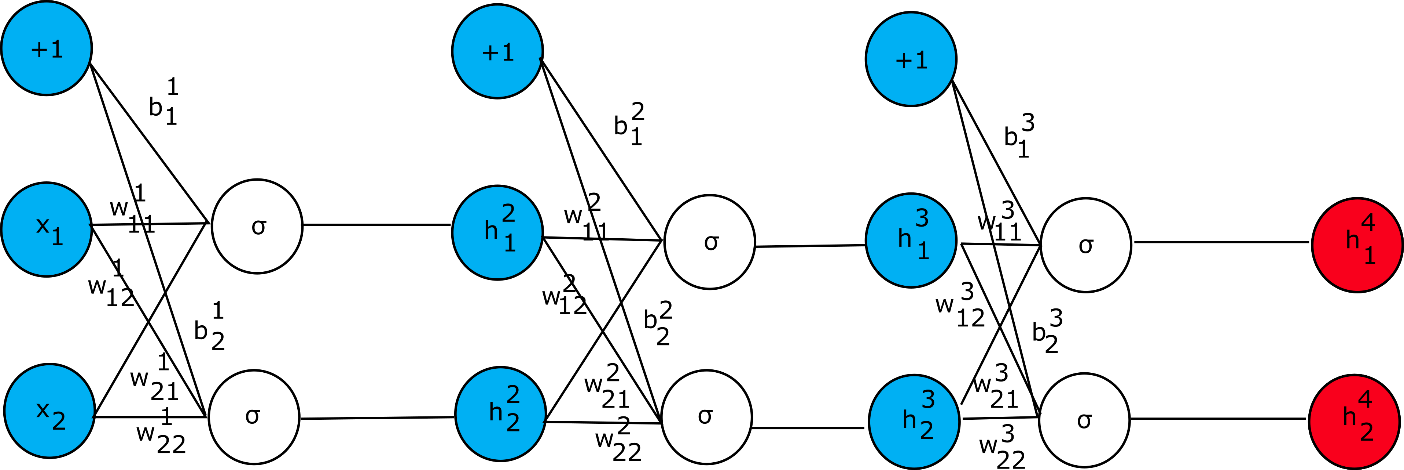
Calculating and is trivial: the first probability can be calculated as a fraction of time class appears in the dataset, and the second probability can be calculated as a fraction of time feature appears in the class . The above classifier assumes that the features appear independently, hence the name of the classifier – naïve. The classification is therefore done as follows (we can ignore the denominator since it does not depend on class and is effectively a constant, i.e., it scales all results in a similar way):

To ease the computation, we can take the logarithm of the function.

**Artificial neural networks**

Neural networks is a powerful tool that is used when the task at hand is to classify the data.

Let’s consider simple neural network that is shown below:



In this example, we consider that the error function is given as: . The goal of the backpropagation is to adjust the weights of the neurons in such a manner that the error function is minimized. Let’s consider first how to adjust the weights of the neural network presented and then we will give the general formula for the gradients.

In the example presented in the next few paragraphs we use sigmoid function . This function has a simple derivative which is convenient during the derivation of backpropagation rules.

Given the network presented in the figure above, let’s consider how different parts should be evaluated (starting from left to right):

Starting from the last layer (on the right) we can calculate the gradient with respect to weights as follows (for now let’s focus what effect does weight have on the error function):

We can denote:

Now in the similar fashion we can derive:

And

We need to continue to derive partial derivatives for all the weights (including the bias terms). For example, computing the partial derivative with respect to and results in the following expressions:

We can observe the pattern in these expressions and so we can generalize. Let’s use the term such that:

Observing that we have recurrency we can restate the above equation as follows:

But, given that by definition is

We can write as

And therefore, we get:

With these equations at hand, we can write the partial derivative as follows:

We now have everything we need to construct the algorithm that updates the weights of the neural network. Basically, a rather trivial approach is to evaluate step by step the neural network (feedforward step) and obtain the output. Next compute the gradients and update the weights. Repeat these steps until the error is below preconfigured threshold. The weight update procedure looks as follows:

Were is a learning rate, which is rather small value such as 0.01. It is the hyperparameter.

When working with neural networks, though, several other hyperparameters should be selected such as the number of hidden layers, the size of the hidden layers, the activation function, the optimizer (such as gradient descent or Adam optimizer), the learning rate parameter (which we described already) and regularization parameter. In our experiments we are going to use the feedforward neural network which consists of one input layer, two hidden layers and one output layer with one node (since we have just two classes and use sigmoid activation function, the value that is less than 0 is going to be class one, and value larger or equal to zero will be class two). For simplicity, the activation function that we are going to use in the example is the sigmoid function. And finally, we are going to use the gradient descent algorithm for the optimization.

**Dataset**

In our experimental work we use ISCX dataset that can be downloaded freely from the Internet. The dataset already contains parsed and labeled data. Thus, there are 5897 samples that are related to SSH-Patator – a Kali Linux tool that is used to perform brute force attacks. Moreover, there are 432074 samples that are related to benign activity. And finally, the dataset contains 7938 samples that are related to FTP brute force attacks.

**Features**

**Results**