Personal Challenge Research

Advancing Network Security: Evaluating Machine Learning Approaches for Intrusion Detection in the NSL-KDD Dataset

Mohammed Nasser Alshukaili
ICT, Fontys University of Applied Sciences

AI for Society, Semester 7

2024

Table of Contents

1. Introduction

Information system security has become critical in a time when digital transformation is at the core of society progress. Cyber-attacks are one of the biggest threats to our digital society because of the increase in internet connectivity and the widespread use of digital devices, which have not only made it possible to access information and services never before possible but have also revealed vulnerabilities. In order to detect these threats, intrusion detection systems (IDS) are essential because they keep an eye on network traffic and look for unusual activity or potential breaches. More sophisticated solutions are required as traditional intrusion detection systems (IDS) are unable to keep up with the increasing complexity and sophistication of cyber threats.

In many different industries, artificial intelligence (AI) and machine learning (ML) have become revolutionary technologies that provide fresh approaches to challenging issues. Innovative approaches are presented in cybersecurity, specifically in the development of intrusion detection systems (IDS), to improve detection capabilities and adjust to the ever-changing nature of cyber threats. The cybersecurity community has widely adopted the NSL-KDD dataset, an enhanced version of the KDD'99 dataset, for training and assessing IDS models. This dataset serves as a benchmark for researchers and practitioners to evaluate the effectiveness of their solutions.

Using the NSL-KDD dataset, this study attempts to investigate how machine learning techniques can be applied to improve intrusion detection. This study aims to support

ongoing efforts to secure the internet against malicious activities by examining different machine learning models and their efficacy in detecting various forms of network intrusions. The emphasis on AI and ML highlights the potential of these technologies in enhancing IDS capabilities in addition to being in line with current cybersecurity trends.

This work aims to find practical approaches to enhance intrusion detection by analysing the NSL-KDD dataset in detail and using modern facilities machine learning models. This study intends to offer important insights into the creation of stronger and more flexible cybersecurity defences by addressing the weaknesses of current IDS solutions and investigating the advantages of AI and ML.

2. Problem Description

The ongoing increase in cyberattacks presents a serious threat to network system security globally. The identification and mitigation of such threats has been greatly aided by the use of traditional intrusion detection systems (IDS). Unfortunately, they frequently suffer from high false positive rates, poor adaptability to novel or developing attack vectors, and high maintenance costs. These drawbacks highlight the requirement for more resilient and dynamic systems that can adjust to the ever-changing landscape of cybersecurity threats.

Because machine learning (ML) can learn from data and make intelligent decisions based on patterns and anomalies found within that data, it presents a promising solution to these problems. Nevertheless, there are some difficulties with using ML for intrusion detection. The quality of the data used for training and testing, the selection of pertinent features, and the selection of algorithms that are resistant to the advanced tactics used by attackers all have a significant role in how effective ML-based IDS is.

The NSL-KDD dataset, while a significant improvement over its predecessor, the KDD'99 dataset, still presents challenges that need to be addressed. These include imbalances between different classes of network traffic, outdated attack samples that may not represent newer types of cyberattacks, and a lack of labelled data for emerging threats. Furthermore, while there is a plethora of machine learning models available, identifying the most effective ones for intrusion detection requires extensive testing and validation.

This research aims to tackle these issues by applying various machine learning techniques to the NSL-KDD dataset to determine the most effective models for detecting different types of network intrusions. The goal is to enhance the accuracy, efficiency, and

adaptability of IDS, thereby improving the cybersecurity posture of network systems against an ever-evolving array of cyber threats.

3. Scope

The goal of this project is to investigate how machine learning methods can be used to detect intrusions inside the NSL-KDD dataset. This study's scope includes several important areas:

1- Data Analysis: To comprehend the NSL-KDD dataset's structure, feature distribution, and the traits of the network intrusions it contains, we will perform an in-depth exploratory data analysis (EDA).

2- Machine Learning Models: A number of models that are well-known for working well in classification tasks will be assessed. This covers both more advanced techniques like Support Vector Machines (SVM) and Neural Networks, as well as more conventional models like Decision Trees and Random Forests.

3- Evaluation of Performance: The study will concentrate on evaluating these models' performance in terms of F1 score, accuracy, precision, and recall. The assessment will assist in determining the best models for detecting the different kinds of intrusions that are shown in the dataset.

4- Methodological Restrictions: The research will be carried out with a cognizance of the inherent limitations in the NSL-KDD dataset, such as its representativeness of actual network traffic and the dynamic character of cyberthreats.

5- Practical Applications: The research will not address the deployment of these models in an actual network environment, even though its goal is to provide theoretical understanding and useful insights into the application of ML for intrusion detection.

This targeted strategy will enable a thorough investigation of machine learning's potential to improve intrusion detection systems, offering insightful information about the advantages and disadvantages of existing approaches.

4. Research Questions

4.1 Main Question

How can Artificial Intelligence and Machine Learning be leveraged to improve the detection of network intrusions and enhance cybersecurity defences using the NSL-KDD dataset?

4.2 Sub Questions

1- What are the fundamental AI and ML concepts and techniques essential for developing an effective intrusion detection system?
2- How does Exploratory Data Analysis (EDA) contribute to understanding the NSL-KDD dataset, and what insights can it provide to inform the development of a more accurate intrusion detection model?
3- What are some effective machine learning models for intrusion detection in cybersecurity, and how do they compare in terms of accuracy, performance, and scalability when applied to the NSL-KDD dataset?
4- What are the practical challenges of implementing AI/ML-based intrusion detection systems in real-world cybersecurity environments?

5. Research Methodology

In this research, the methods will be sourced from the CMD Methods website (https://cmdmethods.nl/).

Question	Strategy	Methods
1- What are the fundamental AI and ML concepts and techniques essential for developing an effective intrusion detection system?	Library	Literature study
2- How does Exploratory Data Analysis (EDA) contribute to understanding the NSL-KDD dataset, and what insights can it provide to inform the development of a more accurate intrusion detection model?	Library, Showroom	Literature study, community research, Peer Review
3- What are some effective machine learning models for intrusion detection in cybersecurity, and how do they compare in terms of accuracy, performance, and scalability when applied to the NSL-KDD dataset?	Library, Lab, Showroom	Literature study, Design pattern research, community research, Best, good & bad practice, Usability Testing, Peer Review
4- What are the practical challenges of implementing AI/ML-based intrusion detection systems in real-world cybersecurity environments?	Library, Lab	Literature study, Design pattern research, Best, good & bad practice, Security test.

6. Approach and Planning

6.1 Approach

I will employ the Scrum methodology, consisting of four sprints. Each sprint will last four weeks.

6.2 Sprint Planning

Sprint	Week	Global Planning
0	2-4	- Personal Project Proposal
1	5-7	- Answer sub-question 1: (What are the fundamental AI and ML concepts and techniques essential for developing an effective intrusion detection system?)
2	8-10	- Answer sub-question 2: (How does Exploratory Data Analysis (EDA) contribute to understanding the NSL-KDD dataset, and what insights can it provide to inform the development of a more accurate intrusion detection model?)
3	11-13	- Answer sub-question 3: (What are some effective machine learning models for intrusion detection in cybersecurity, and how do they compare in terms of accuracy, performance, and scalability when applied to the NSL-KDD dataset?)
4	14-16	- Answer sub-question 4: (What are the practical challenges of implementing AI/ML-based intrusion detection systems in real-world cybersecurity environments?)

7. Results

7.1 What are the fundamental AI and ML concepts and techniques essential for developing an effective intrusion detection system?

7.1.1 Artificial Intelligence (AI)

Artificial intelligence (AI) is the capacity of a computer or robot under computer control to carry out operations typically performed by intelligent things. The phrase is commonly used to describe the work of creating artificial intelligence systems that possess like a human ability in areas like reasoning, meaning-finding, generalization, and experience-based learning. (Copeland, 2024)

7.1.2 Machine Learning (ML)

Machine learning is a subset of artificial intelligence (AI). It includes image recognition systems, self-driving cars, and products like Amazon's Alexa.
This involves using data and algorithms to emulate the way humans learn enabling machines to do precise predictions, classifications, or the extraction of insights driven by data.
Machine learning is about teaching a computer by feeding it lots of data enabling it to guess outcomes, spot patterns, or sort data. There are three kinds of machine learning: supervised, unsupervised, and reinforcement learning. (Staff, 2023)

7.1.2.1. Supervised learning

Supervised learning is a type of machine learning that is expected to be the most used in businesses, as per Gartner, a business consulting company.
This method is being used by providing the models past data. The machine gets both the input and expected output. Then, it processes these to make its future outputs as accurate as possible. Methods like neural networks, decision trees, and linear regression are common in supervised learning.
In supervised learning, the machine is guided, or "supervised," during its learning. It's given labelled data as the outcome it needs to learn, and other data as input features to work with. (Staff, 2023)

7.1.2.2. 2. Unsupervised learning

Unsupervised learning, unlike supervised learning, doesn't rely on labelled training sets. Instead, it allows the machine to discover less apparent patterns in the data by itself. Algorithms commonly used in unsupervised learning include Hidden Markov models, k-means clustering, hierarchical clustering, and Gaussian mixture models.

Prediction modelling is one of the many uses for unsupervised learning. It is frequently used for association (finding the rules that connect these groups) and clustering, which groups objects according to their characteristics. Organizing inventory based on manufacturing or sales metrics, classifying customers based on their purchasing patterns, and identifying connections within customer data (such as patterns in the purchases of goods) are a few real-world examples. (Staff, 2023)

7.1.2.3. Reinforcement learning

One kind of machine learning that closely resembles how people learn is called reinforcement learning. This method involves an algorithm, or "agent," that learns through interactions with its surroundings and the provision of positive or negative rewards. Q-learning, deep adversarial networks, and temporal difference are important algorithms in this field.
For example, in a game where we want a car to reach the finish line as soon as possible, we would let the car explore the field by itself and reward it when going the correct direction and take away the rewards when going the wrong direction. This lets the model do the correct job without providing it with any data. (Staff, 2023)

Several machine learning models can be especially useful when analyzing Zeek logs for anomaly detection. Zeek logs are a rich source of network data, and the choice of model depends on the specific aspects of network traffic that is being captured. Here are some models that are commonly used:

7.1.3 Decision Trees

Decision trees are a type of supervised machine learning algorithm that is used for classification and regression tasks. They make decisions based on asking a series of questions.

Structure of a decision trees

- Root Node: This is where the tree starts.
- Splitting: dividing the node into 2 or more sub-nodes based on certain conditions.

- Decision/Internal Node: After the first split, each sub-node becomes a decision/Internal node and can be split further.
- Leaf Node: Last nodes that have no further splits.

The diagram (Figure 1 Structure of a Decision Tree) clearly visualizes each component of a decision tree, showing how data is segmented at different levels based on diverse conditions until a decision is reached at the leaf nodes.

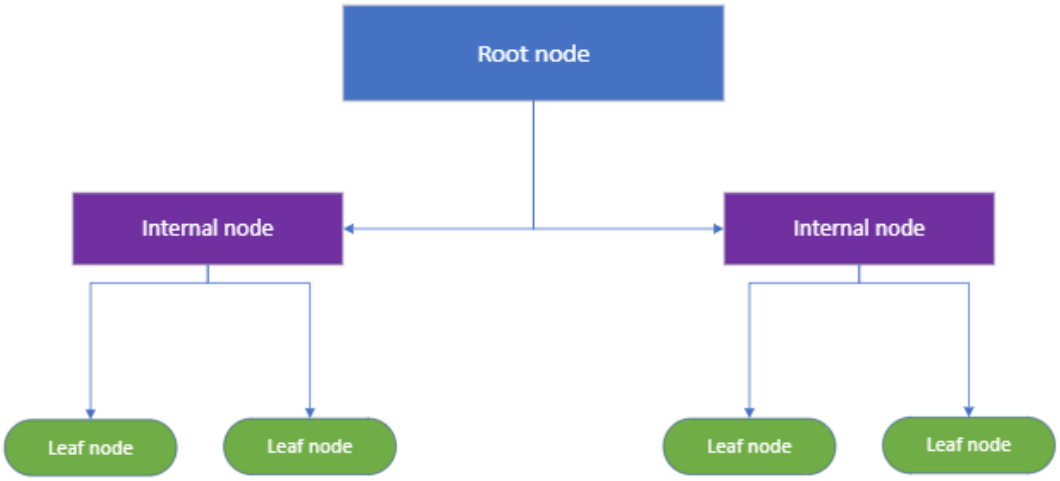


Figure 1 Structure of a Decision Tree.

How to choose the best attribute at each node

There are many techniques for choosing the best attribute at each node in decision tree models. Information gain and Gini impurity are two widely used methods.

These techniques assess the effectiveness of each potential next split by categorizing data into classes.

To understand how these methods work, it is essential to start with the concept of entropy. Entropy is a measure of the impurity in a set of data. It helps to determine how a decision tree should split the data at each node. (What Is a Decision Tree | IBM, 2023)

7.1.4 Numpy

In order to start working with machine learning models, I need to learn Numpy. Numpy is a library in Python that simplifies working with arrays.

Here are some of the things that I learned about Numpy:

numpy.where: To find a value inside an array.

numpy.ones((x,x)): To generate a matrix (x columns, x rows).

numpy.zeros((6,6)): This creates 6*6 matrix all zeros.

numpy.max(x): Get the maximum value in an array.

x.transpose(): To change from a row vector to a column vector.

numpy.sum(x): Get the sum of x

(NumPy: The Absolute Basics for Beginners — NumPy v1.26 Manual, 2023)

7.1.5 Tensorflow

TensorFlow is an open-source library for machine learning, flexible tools for building and training models to derive insights and predictions from data.
I will go through some of the basics in order to start designing my own models to analyze zeek traffic.

This is how to create a Tensor with multiple float variables:

```
x = tf.Variable(initial_value=[10., 20., 30.], name='float_tensor')
```

In order to add a new dimension to a Tensor, we can add [] to the values:

```
x = tf.Variable([[10, 20, 30]], dtype=tf.float32 )
```

Move x to 3 dimensions:

```
x = tf.Variable([[[10, 20, 30]]], dtype=tf.float32,
name='tf_float_variable' )
```

Let's print the value of x:

```
x = tf.Variable([[[10, 20, 30]]], dtype=tf.float32, name='tf_float_variable' )
x
```

```
<tf.Variable 'tf_float_variable:0' shape=(1, 1, 3) dtype=float32, numpy=array([[[10., 20., 30.]]],
dtype=float32)>
```

This is a 'scalar' or 'rank-0' tensor. A scalar consists of just one value and does not have any 'axes'.

```
rank_0_tensor = tf.constant(4)
print(rank_0_tensor)
```

```
tf.Tensor(4, shape=(), dtype=int32)
```

"A 'vector' or 'rank-1' tensor is similar to a list of values. It has one axis:"

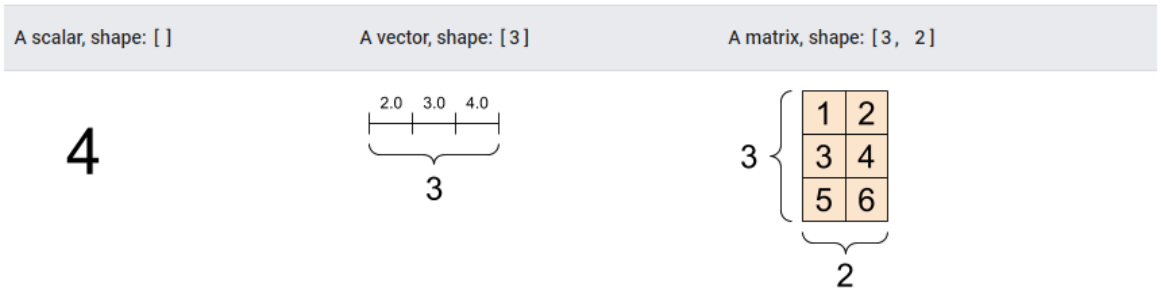
```
rank_1_tensor = tf.constant([2.0, 3.0, 4.0])
print(rank_1_tensor)
```

```
tf.Tensor([2. 3. 4.], shape=(3,), dtype=float32)
```

A 'matrix' or 'rank-2' tensor is characterized by having two axes:

```
rank_2_tensor = tf.constant([[1, 2],
                             [3, 4],
                             [5, 6]], dtype=tf.float16)
print(rank_2_tensor)
```

```
tf.Tensor(
[[1. 2.]
 [3. 4.]
 [5. 6.]], shape=(3, 2), dtype=float16)
```



A tensor can be converted into a NumPy array using either np.array or the tensor.numpy method:

```
np.array(rank_2_tensor)
```

```
array([[1., 2.],
       [3., 4.],
       [5., 6.]], dtype=float16)
```

Some of the previous images contain the word shape, what is shape?
Tensors have shapes:
Shape: The count of elements along each axis of a tensor.
Rank: The total number of axes in a tensor. Scalars are rank 0, vectors rank 1, and matrices rank 2.
Axis or **Dimension**: A specific dimension within a tensor.
Size: The overall quantity of items in the tensor, determined by multiplying the elements of the shape vector.

```
rank_4_tensor = tf.zeros([3, 2, 4, 5])
```

```
print("Type of every element:", rank_4_tensor.dtype)
print("Number of axes:", rank_4_tensor.ndim)
print("Shape of tensor:", rank_4_tensor.shape)
print("Elements along axis 0 of tensor:", rank_4_tensor.shape[0])
print("Elements along the last axis of tensor:", rank_4_tensor.shape[-1])
print("Total number of elements (3*2*4*5): ", tf.size(rank_4_tensor).numpy())
```

```
Type of every element: <dtype: 'float32'>
Number of axes: 4
Shape of tensor: (3, 2, 4, 5)
Elements along axis 0 of tensor: 3
Elements along the last axis of tensor: 5
Total number of elements (3*2*4*5):  120
```

tf.zeros can be used to create Tensors that contain zeros with the provided shape.

More about shapes.

```
# Shape returns a `TensorShape` object that shows the size along each axis
x = tf.constant([[1], [2], [3]])
print(x.shape)
```

```
(3, 1)
```

```
# You can convert this object into a Python list, too
print(x.shape.as_list())
```

```
[3, 1]
```

A tensor can be reshaped into a different shape. The tf.reshape function is efficient and cost-effective, as it doesn't require duplicating the original data.

```
# You can reshape a tensor to a new shape.
# Note that you're passing in a list
reshaped = tf.reshape(x, [1, 3])
print(x.shape)
print(reshaped.shape)
```

```
(3, 1)
(1, 3)
```

(SecurityNik, 2023)

7.1.6 Linear Regression

Linear Regression is a fundamental algorithm in data science. It is used to predict the association between two variables based on the presumption of a linear link between the independent and dependent variables. Its goal is to find the best-fit line that reduces the total squared differences between the predicted and actual values.

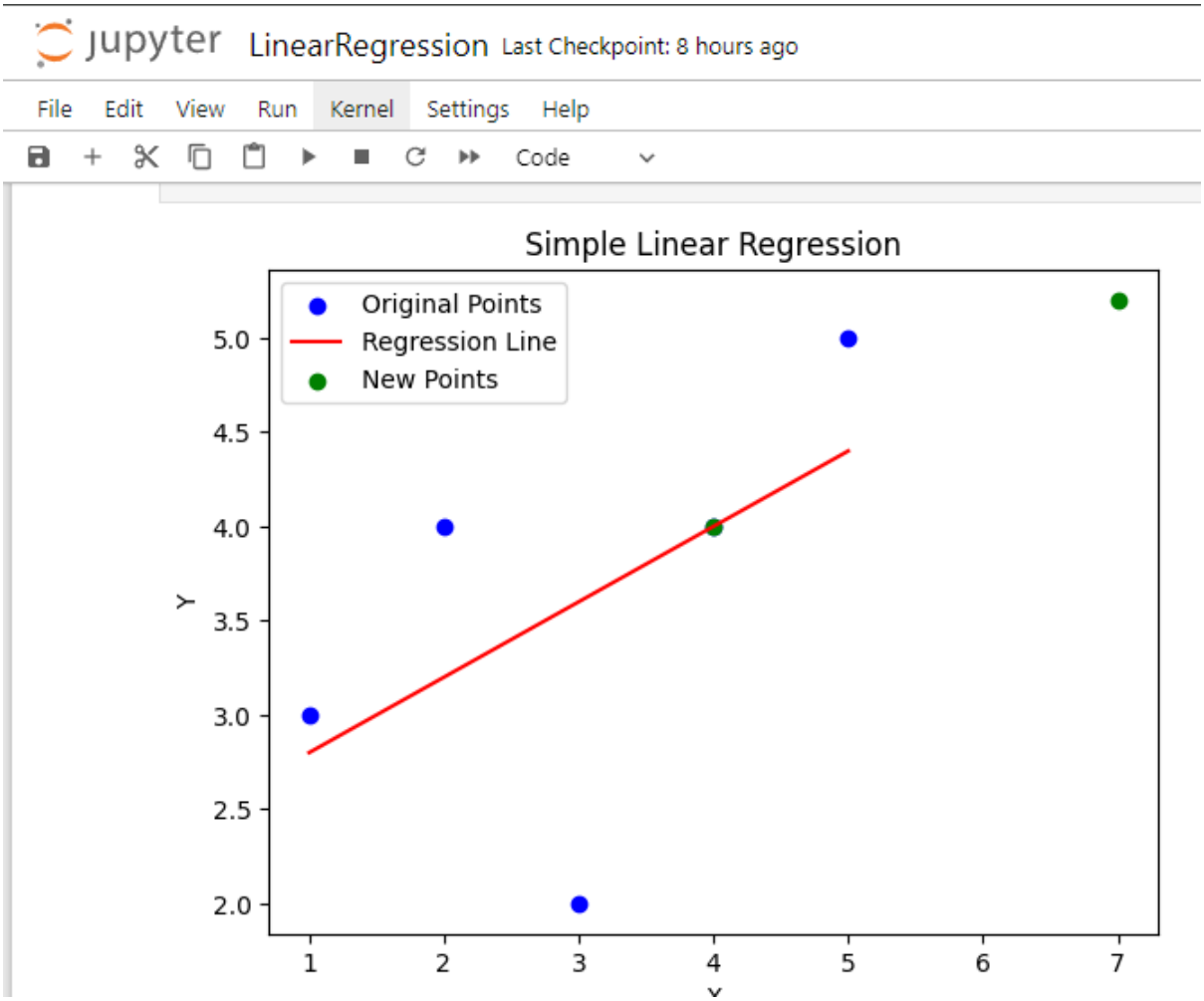


Figure 2 A LinearRegression output that predicts the y value for x values using a line

The blue dots are the training data that was provided to the function, based on the training data, this function creates a line that minimize the destination between the line and the dots. Then, th e line will be used to predict new values.

As you can see, the line was created based on the training data, and after we try to predict new values, it puts them on the line (green dots).

Here is the code for this output:

```
[14]: import numpy as np
      import matplotlib.pyplot as plt

      def simple_linear_regression(X, Y):
          # Calculate the slope (m) and intercept (b) for y = mx + b
          m, b = np.polyfit(X, Y, 1)

          def predict(new_x):
              # Predict new y values using the linear model
              return m * new_x + b

          def visualize(new_x, new_y):
              # Visualize the original points, the regression line, and new points
              plt.scatter(X, Y, color='blue', label='Original Points')  # Original points
              plt.plot(X, m*X + b, color='red', label='Regression Line')  # Regression line
              plt.scatter(new_x, new_y, color='green', label='New Points')  # New points
              plt.xlabel('X')
              plt.ylabel('Y')
              plt.title('Simple Linear Regression')
              plt.legend()
              plt.show()

          return predict, visualize

      # Example usage
      X = np.array([1, 2, 3, 4, 5])
      Y = np.array([3, 4, 2, 4, 5])

      predict, visualize = simple_linear_regression(X, Y)

      # Predict new points
      new_X = np.array([4, 7])
      new_Y = predict(new_X)

      # Visualize the results
      visualize(new_X, new_Y)
```

Figure 3 The code for the LinearRegression model

7.1.7 Support Vector Machines

Support vector machines (SVMs) is a famous supervised ML method that is being used for classification and outliers' detection.

One of the advantages of Support vector machines is that it works with low and high dimensional spaces.

SVM focuses on identifying a hyperplane that optimally separates two classes. SVM is similar to logistic regression, however it is important to highlight that their approach differ fundamentally.

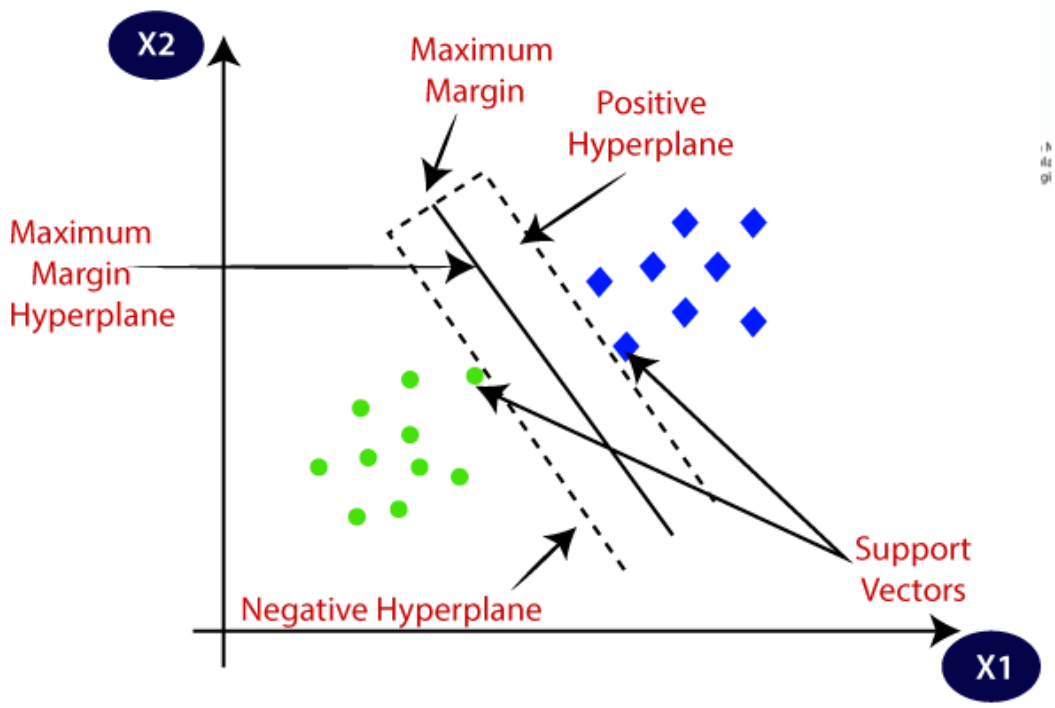
Which hyperplane does it select? There can be millions of hyperplanes can classify the objects into 2 categories. So, how does SVM know which is the best?

SVM picks the best hyperplane by finding the maximum margin between the hyperplanes. This means that it selects the maximum distance between 2 objects (classes).
SVM algorithms can be categorized into two types:

- **Linear SVM:** This is used when the dataset can be linearly separable, meaning that the data points can be classified into two categories with a single straight line.
- **Non-Linear SVM:** This is used when the dataset is not linearly separable, meaning the data points cannot be divided into two classes using a straight line in a two-dimensional view, we use Non-Linear SVM.

The main terms in SVM are:

- **Support Vectors:** These are the closest data points to the hyperplane in an SVM mode. The position of the hyperplane is mainly influenced by these points.
- **Margin:** This is the gap between the hyperplane and the nearest data points (the support vectors).



https://www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/

Bias: It can be referred to the errors in a machine learning algorithm. High bias means bad algorithm as it is more likely to give wrong predictions.
Variance: High variance means that the algorithm is not consistent with multiple datasets, in some datasets it has low bias, and in others it has high bias.

Cross validation: This helps to choose the best machine learning method for a specific purpose. It allows us to compare different machine learning methods and get a sense of how well they will work. When trying to choose the best ML method, we train 80% of the dataset and 20% to test the accuracy of the dataset. However, how to know which piece of data to train and which to test? Cross validation solves this problem by training and testing all the pieces in the dataset and return the best fit.
Kernel tricks: It transforms the dataset into a higher dimension where a hyperplane can effectively separate the classes. (Saini, 2023)

Now, to understand SVM more, I will write simple Python code that uses SVM.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets
```

```
#Generating random dataset
X, y = datasets.make_blobs(n_samples=100, centers=2, random_state=6)
```

```
# Step 2: Training an SVM Model
# We use a linear kernel for simplicity.
clf = svm.SVC(kernel='linear', C=1000)
clf.fit(X, y)
# Step 3: Visualization
plt.scatter(X[:, 0], X[:, 1], c=y, s=30, cmap=plt.cm.Paired)

# Plot the decision boundary
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()

# Create grid to evaluate model
xx = np.linspace(xlim[0], xlim[1], 30)
yy = np.linspace(ylim[0], ylim[1], 30)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
Z = clf.decision_function(xy).reshape(XX.shape)

# Plot decision boundary and margins
ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,
           linestyles=['--', '-', '--'])

# Highlight support vectors
ax.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1], s=100,
           linewidth=1, facecolors='none', edgecolors='k')
plt.show()
```

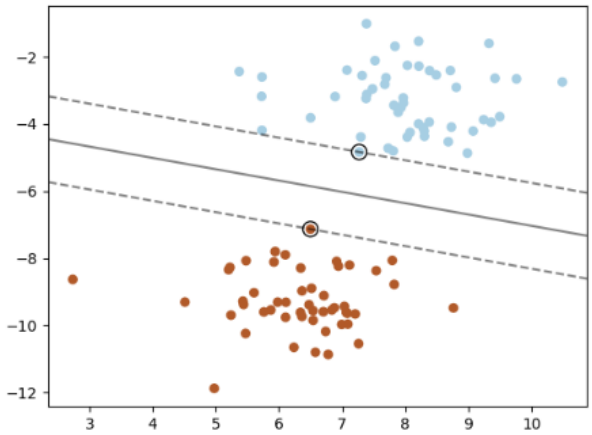
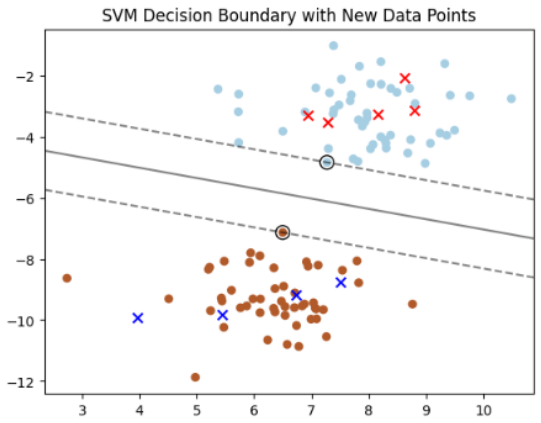


Figure 4 The code for an SVM model that classifies data into 2 categories.

This is Jupyter notebook that implements a very basic SVM model. The plot shows that the model creates a hyperplane based on the distance between the closest data points.

Now, I want to make the model classifies (predicts) new data points.
I added a new array that contains new data points that I want the model to classify:

```
[25]: new_data_points = np.array([[8.80150168, -3.12894105], [8.62901496, -2.0702682], [7.28781069, -3.51982446],
                                  [8.1640321, -3.25401191], [6.9349555, -3.31489601], [6.7391796, -9.18618687],
                                  [5.43568715, -9.81066104], [7.49956603, -8.74346612], [3.96998252, -9.9049186]])

      # Predicting classes for the new data points
      predicted_labels = clf.predict(new_data_points)

      # Plot the original data
      plt.scatter(X[:, 0], X[:, 1], c=y, s=30, cmap=plt.cm.Paired)

      # Plot decision boundary and margins
      ax = plt.gca()
      xlim = ax.get_xlim()
      ylim = ax.get_ylim()
      xx = np.linspace(xlim[0], xlim[1], 30)
      yy = np.linspace(ylim[0], ylim[1], 30)
      YY, XX = np.meshgrid(yy, xx)
      xy = np.vstack([XX.ravel(), YY.ravel()]).T
      Z = clf.decision_function(xy).reshape(XX.shape)
      ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5, linestyles=['--', '-', '--'])

      # Highlight support vectors
      ax.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1], s=100, facecolors='none', edgecolors='k')

      # Plot new data points with predicted labels
      for i, label in enumerate(predicted_labels):
          plt.scatter(new_data_points[i, 0], new_data_points[i, 1], color='red' if label == 0 else 'blue', s=50, marker='x')

      plt.title("SVM Decision Boundary with New Data Points")
      plt.show()
```




```
[ ]:
```

Figure 5 Adding new data points to the model to see how it classifies them

As seen above, the model predicts the location of the new data points and place them in their correct class.

7.1.8 Conclusion

Understanding the nature of entropy and information gain, as illustrated through the decision tree model, allows cyber security professionals to implement effective anomaly detection systems. With the right approach to training and the application of concepts like entropy in supervised learning, machine learning models can be trained to recognize and flag unusual patterns in network logs that could indicate a security threat.

Furthermore, the integration of machine learning within big datasets would enhance the capability to automate the detection of anomalies in network traffic. By employing models such as decision trees, support vector machines, and others, organizations can effectively respond to potential cyber security incidents.

7.2 How does Exploratory Data Analysis (EDA) contribute to understanding the NSL-KDD dataset, and what insights can it provide to inform the development of a more accurate intrusion detection model?

7.2.1 What is EDA?

Data scientists utilize exploratory data analysis (EDA) to examine datasets, highlighting their key features frequently using data visualization techniques.
EDA helps determine the best methods for processing data sources to extract the necessary insights, making it easier for data scientists to find patterns, spot anomalies, test theories, or confirm presumptions.
EDA's primary goal is to assist in examining data before drawing any conclusions. It can assist in locating glaring errors, better understanding data patterns, spotting outliers or unusual occurrences, and discovering intriguing correlations between the variables. Exploratory analysis is a tool that data scientists can use to make sure the results they generate are reliable and relevant to any intended business objectives. By ensuring that stakeholders are asking appropriate inquiries, EDA also benefits them. Standard deviations, categorical variables, and confidence intervals are among the topics that EDA

can assist with. The features of EDA can be applied to more complex data analysis or modeling, such as machine learning, after it is finished, and conclusions have been drawn.

7.2.2 Exploratory data analysis tools

With EDA tools, you can perform a range of statistical procedures and methods, such as:

- methods for dimension reduction and clustering that help to visualize high-dimensional data with lots of different variables.
- presentation of individual attribute visualizations from the raw data set combined with summary statistics.
- Multivariate visualizations are useful for understanding and mapping the relationships between various data fields.
- K-means clustering is an unsupervised learning clustering technique in which data points are grouped into K groups, or the total number of clusters, according to how far they are from the centroid of each group. The data points that fall into the same category are those that are closest to a given centroid. Pattern recognition, picture compression, and market segmentation are three common applications of K-means clustering.
- Data and statistics are used by predictive models, like linear regression, to forecast outcomes.

(What Is Exploratory Data Analysis? | IBM, 2023)

7.2.3 Understanding the NSL-KDD Dataset

7.2.3.1. Dataset Overview

Tavallaee et al. (2009) state that the NSL-KDD dataset is a publicly accessible resource that was created from the previous KDD Cup99 dataset. An inaccurate assessment of Automated Intrusion Detection Systems (AIDS) resulted from a statistical analysis of the Cup99 dataset, which revealed important problems that significantly impact intrusion detection accuracy.

The primary issue with the KDD dataset, as analyzed by Tavallaee et al. (2009), is the substantial number of duplicate packets present. Their analysis of both the training and testing sets revealed that approximately 78% and 75% of network packets, respectively, were duplicates.

Because there are a lot of duplicate instances in the training set, machine learning techniques may be biased toward typical cases and unable to learn from the irregular instances, which frequently pose more serious risks to computer systems.
Tavallaee et al. removed duplicate records from the KDD Cup'99 dataset in 2009 in order to create the NSL-KDD dataset, which addresses the issues found in that dataset. There are 125,973 records in the NSL-KDD training dataset and 22,544 records in the test dataset. Because of its manageable size, the NSL-KDD dataset can be used for research purposes without the need for random sampling, which has resulted in consistent and comparable results across studies. The NSL-KDD dataset has 41 attributes and includes 22 training intrusion attacks. Of these, a full set of features for analysis in intrusion detection research are provided by the 19 attributes that describe the nature of connections within the same host and the 21 attributes that relate to the characteristics of the connection. (Saylor Academy, 2023)

7.2.3.2. Feature Composition

To address the "Feature Composition" of the NSL-KDD dataset for intrusion detection systems, we dive into the types of features included in the dataset, their data types, and their relevance to identifying potential security threats. Here is a closer look at the NSL-KDD dataset's feature composition:

7.2.3.3. Types of Features in NSL-KDD

The NSL-KDD datasets includes 42 features (columns) including a label feature that categorizes each connection as either normal or an attack. The types of attacks are also subdivided into four categories:

- DoS (Denial of Service)
- R2L (Remote to Local)
- U2R (User to Root)
- Probe

All the features in the dataset can be categorized into three main types:

1- Basic Features: Basic Features encompass attributes extracted directly from packet headers, such as connection duration, protocol type, service requested, and flag status, representing core network connection qualities easily identifiable from network traffic.

2- Content Features: Content Features analyze packet payloads for anomalies, such as failed login attempts, crucial for detecting U2R and R2L attacks involving abnormal data transmissions.

3- Traffic Features: These features track the connections where the same host is trying to connect to the same service. This can be helpful when trying to detect DoS attacks.

7.2.3.4. Data Types of Features

- Numerical Features: Most features in the dataset are numerical.
- Categorical Features: Some features are categorical, representing types of protocols (e.g., tcp, udp, icmp), services (e.g., http, ftp, telnet), and network connection status (e.g., SF, S1, REJ). These features often require preprocessing, to be used effectively in machine learning models.

7.2.3.5. Identifying Key Features

The dataset contains over 40 columns, making it important to choose only the key features that are relevant to determining if a particular connection is an attack or benign.

Using my knowledge of cybersecurity, I carefully looked over each column in the dataset and chose a few features that seemed relevant to my objective. The selected features and their descriptions are listed below:

- ‘duration’: The length of the connection.
- ‘protocol_type’: The type of the used protocol like tcp, udp, etc.
- ‘service’: The network service like http, telnet, ssh, etc.
- ‘flag’: The status of the connection like S0, S1, etc.
- ‘src_bytes’: The number of the data bytes from source to destination.
- ‘dst_bytes’: The number of data bytes from destination to source.
- ‘logged_in’: This is binary column where 1 means a successfully login and 0 otherwise.
- ‘is_host_login’: This is binary column where 1 means the login belonged to the "host" list and 0 otherwise.
- ‘is_guest_login’: 1 if the login is a "guest" login and 0 otherwise.
- ‘attack’: There is a column in the dataset that says whether that connection is normal or a type of an attack. There are many types of attack mentioned. Therefore, I will categorize all the type of attacks as only one value (attack) to make it a binary column (normal & attack).

7.2.3.6. Identifying Key Features using Feature Selection

Hans mentioned that it is good to use my experience to pick the important features. However, he advised me to try and implement Feature Selection method to find the best possible features in my dataset.

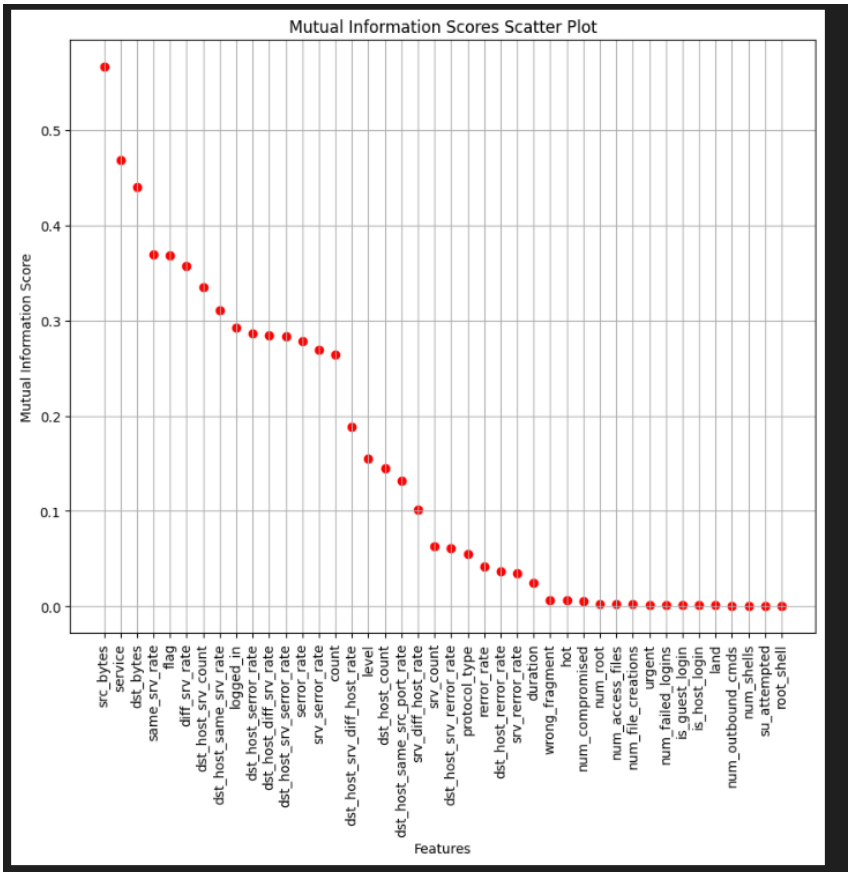
First, I will use mutual information classifier to evaluate feature importance. Then I will use SelectKBest to select the number of features that I want.

```
from sklearn.feature_selection import mutual_info_classif
mutual_info = mutual_info_classif(x_train, y_train)
mutual_info = pd.Series(mutual_info)
mutual_info.index = x_train.columns
mutual_info = mutual_info.sort_values(ascending=False)
mutual_info
```

✓ 28.0s

```
src_bytes                    0.566396
service                      0.468618
dst_bytes                    0.440064
same_srv_rate                0.369244
flag                         0.368950
diff_srv_rate                0.357635
dst_host_srv_count           0.335226
dst_host_same_srv_rate       0.310637
logged_in                    0.292657
dst_host_serror_rate         0.286156
dst_host_diff_srv_rate       0.284316
dst_host_srv_serror_rate     0.283900
serror_rate                  0.278472
srv_serror_rate              0.269268
count                        0.264724
dst_host_srv_diff_host_rate  0.188634
level                        0.155320
dst_host_count               0.144749
dst_host_same_src_port_rate  0.132137
srv_diff_host_rate           0.101122
srv_count                    0.063164
dst_host_srv_rerror_rate     0.061359
protocol_type                0.055364
rerror_rate                  0.042157
dst_host_rerror_rate         0.036913
```

Here is a scatter plot that shows the most relevant features:



Then, I use SelectKBest to choose the features and assign them into new variable:

```
# Selecting the top 15 most informative features
from sklearn.feature_selection import SelectKBest
sel_five_cols = SelectKBest(mutual_info_classif, k=15)
sel_five_cols.fit(x_train, y_train)
x_train.columns[sel_five_cols.get_support()]
```

✓ 27.9s

```
Index(['service', 'flag', 'src_bytes', 'dst_bytes', 'logged_in', 'count',
       'serror_rate', 'srv_serror_rate', 'same_srv_rate', 'diff_srv_rate',
       'dst_host_srv_count', 'dst_host_same_srv_rate',
       'dst_host_diff_srv_rate', 'dst_host_serror_rate',
       'dst_host_srv_serror_rate'],
      dtype='object')
```

```
col=['service', 'flag', 'src_bytes', 'dst_bytes', 'logged_in', 'count',
       'serror_rate', 'srv_serror_rate', 'same_srv_rate', 'diff_srv_rate',
       'dst_host_srv_count', 'dst_host_same_srv_rate',
       'dst_host_diff_srv_rate', 'dst_host_serror_rate',
       'dst_host_srv_serror_rate']
x_train=x_train[col]
x_test=x_test[col]
```

✓ 0.0s

I faced an issue with the selected features; the machine learning models only handle numeric values, yet some of my features are in string format. In search of a solution, I discovered a technique known as One-hot encoding, which I will explore in the following chapter.

7.2.4 One-hot encoding

7.2.4.1. What is Categorical Data?

Categorical data refers to variables that carry label values instead of numerical ones. Normally, the values are in a fixed set.
A “pet” variable might include options like “dog” and “cat”.
A “color” variable could offer choices such as “red”, “green”, and “blue”.
A “place” variable might list rankings like “first”, “second”, and “third”.

7.2.4.2. The Problem with Categorical Data

Some algorithms are meant to work with categorical data straight out of the box. Decision trees, for example, have the ability to learn directly from categorical data. But a lot of machine learning algorithms aren't designed to handle label data in its original form. They demand that the variables be presented in numerical form for both the input and the output.

7.2.5 How to Convert Categorical Data to Numerical Data?

7.2.5.1. Integer Encoding

In the first phase of preparing categorical data for machine learning models, each unique category is assigned a specific integer. For example, the assignment could be set up so that "blue" goes with 3, "green" with 2, and "red" with 1. This is called integer encoding or label encoding, and it is easily reversed. This method might be more than suitable for some kinds of data.
Some machine learning algorithms can identify and make use of a naturally ordered relationship in the numerical values assigned by this method. This is especially true for ordinal variables, where the values' order has significance. An illustration of this would be the previously discussed "place" variable, for which label encoding accurately captures the first, second, and third categories' natural order, making it a useful technique for variables of this type.

7.2.5.2. One-Hot Encoding

For categorical variables where no such ordinal relationship exists, the integer encoding is not enough.
Label encoding, which simply assigns integers to categorical variables without a foundational ordinal relationship, may not be the optimal approach for these variables. The model's performance may suffer or unexpected results, like predictions that fall illogically between categories, may result from relying too heavily on integer encoding, which could lead to incorrect inference of a natural order among categories.
To address this issue, one-hot encoding is employed as an alternative strategy. This approach involves replacing the integer-encoded variable with new binary variables, each representing a unique category value. Essentially, for every unique category, a distinct binary variable is created: this variable is set to "1" for its corresponding category and "0" for all others.
Given the three categories ("red", "green", and "blue") in the "color" example, one-hot encoding would produce three binary variables. For example, if the color is "green," the encoding would be [0, 1, 0], where "green" is indicated by a "1" in the second position, and "red" and "blue" are represented by "0" in the first and last positions, respectively. This approach captures each category's existence or absence efficiently without suggesting a hierarchy. (Brownlee, 2020)

7.2.6 Conclusion

Based on the comprehensive analysis of Exploratory Data Analysis (EDA) and its critical role in data science, as well as the detailed exploration of the NSL-KDD dataset for intrusion detection systems, we can draw several conclusions.
An essential first step in the data analysis process is exploratory data analysis. It gives data scientists the knowledge and resources they need to fully comprehend their dataset, find underlying patterns, spot anomalies, and test theories. EDA facilitates the effective communication of the story of the data through a variety of statistical methods and data visualization techniques, enabling well-informed decision-making and strategic planning. The importance of EDA is found in its capacity to direct the choice of suitable modeling and data processing methods, guaranteeing that the analysis is in line with the current business goals and inquiries.

The investigation of one-hot encoding provides additional insight into the problems and solutions related to categorical data preprocessing for machine learning. Data scientists

can fully utilize their datasets by converting categorical data into a format that machine learning algorithms can understand. This allows for more precise and insightful analysis.

To sum up, the examination of EDA and its utilization with the NSL-KDD dataset highlights how interconnected preprocessing, feature selection, and data preparation are within the larger fields of cybersecurity and data science. It emphasizes the need for thorough data preparation and analysis as the first steps in obtaining trustworthy, useful insights, especially in domains where accuracy and precision are critical. This all-encompassing method not only makes it easier to comprehend the data at hand more deeply, but it also establishes the foundation for the creation of successful models and tactics, which in turn promotes advancements in cybersecurity and other fields.

7.3 What are some effective machine learning models for intrusion detection in cybersecurity, and how do they compare in terms of accuracy, performance, and scalability when applied to the NSL-KDD dataset?

Machine learning primarily deals with two problem types: classification and prediction. Here is a compilation of commonly used algorithms used for creating classification regression models:

Classification Models:

- **Logistic Regression**
- **Naïve Bayes**
- **Decision Trees**
- **Random Forest**
- **K-nearest neighbor (KNN)**
- **Support Vector Machine**

Regression models

- Linear regression
- Ridge regression
- Decision trees
- Random forest
- K-nearest neighbor (KNN)
- Neural network regression

(Staff, 2023)

7.3.1 Theoretical Approach: Common Classification Models

7.3.1.1. Logistic Regression

Despite its name, logistic regression is primarily utilized for binary classification problems, where data falls into two categories. Logistic regression often serves as an initial method for setting a baseline before exploring more complex models.
The word “regression” appears in its name because it estimates the likelihood of an outcome being either 0 or 1 through a linear combination of features. (Choosing the Best Machine Learning Classification Model and Avoiding Overfitting, 2023)

7.3.1.2. Naive Bayes

You may want to use the naive Bayes algorithm if your task and data are relatively simple. When training data is limited, this classifier is a better option than nearest neighbor and logistic regression algorithms due to its high bias and low variance.
Naive Bayes works especially well when memory and CPU resources are restricted. Its ease of use keeps it from overfitting, enabling quick training. Additionally, it functions well when fresh data is added on a regular basis.
However, as data complexity and variance increase, you might find more sophisticated classifiers to be more effective. Naive Bayes' straightforward analysis may not support complex hypotheses. (Choosing the Best Machine Learning Classification Model and Avoiding Overfitting, 2023)

7.3.1.3. K-Nearest Neighbor

Categorizing data points by their proximity to others in a training set can be an effective classification method. The k-nearest neighbor (KNN) algorithm operates on the principle of "guilty by association.
Because KNN is regarded as an instance-based lazy learner, it does not go through a traditional training phase. Rather, you feed the model with the training set and let it run in the background until you need it. The KNN model determines the given number of nearest

neighbors (k) in response to a new query; for instance, if k = 5, it evaluates the class of the five closest neighbors. The model uses a vote process among these neighbors to decide which label is best for classification. It determines the mean of the values of the closest neighbors for regression tasks.

Although KNN requires less time to train than other models, it can take longer to query and require more storage space, especially when the dataset grows. All training data is retained by this model, as opposed to just algorithmic representation. (Choosing the Best Machine Learning Classification Model and Avoiding Overfitting, 2023)

7.3.1.4. Decision Trees

To understand how a decision tree predicts an outcome, start at the root (beginning) node, and follow the path down to a leaf node, which provides the response. Classification trees generate nominal outputs like true or false, while regression trees yield numeric responses.
Decision trees offer clear visibility of the decision-making path from root to leaf, making them particularly helpful when results need to be explained to stakeholders. They are also relatively quick to execute.
However, a primary drawback of decision trees is their propensity to overfit data. Ensemble methods, such as bagging, can mitigate this issue. (Choosing the Best Machine Learning Classification Model and Avoiding Overfitting, 2023)

7.3.1.5. Support Vector Machine

When there are clear differences between the two classes in your dataset, you may want to use a support vector machine (SVM). The ideal hyperplane is the one that maximizes the margin between these classes and divides the data points of one class from those of another. This is how support vector machines (SVMs) operate. SVMs deal with datasets that have more than two classes by dividing the issue into several binary classification tasks that are each overseen by a different SVM.
SVMs offer significant benefits. They are highly accurate and generally resistant to overfitting. Linear SVMs, in particular, are straightforward to interpret. Once trained, SVMs are very quick, allowing for the disposal of training data if memory is limited, making them suitable for environments with restricted resources. They also excel in complex, nonlinear classification tasks through the use of a technique known as the "kernel trick.

However, SVMs require considerable upfront training and tuning, necessitating a significant time investment before they can be deployed. Additionally, their performance

can decrease when handling more than two classes, affecting their speed. (Choosing the Best Machine Learning Classification Model and Avoiding Overfitting, 2023)

7.3.1.6. Neural Networks

An artificial neural network (ANN) is capable of learning and can be trained to solve problems, recognize patterns, classify data, and predict future events. ANNs are frequently employed for complex challenges like character recognition, stock market predictions, and image compression.

The functionality of a neural network hinges on the architecture of its nodes and the strength of the connections between them, known as weights. These weights adjust automatically during training, adhering to specific learning rules until the network proficiently executes the intended task.

ANNs excel in handling nonlinear data with numerous input features, making them ideal for tackling sophisticated problems that simpler algorithms struggle with. However, they come with some downsides: ANNs are resource-intensive, their decision-making processes are often opaque (making it hard to deduce how a solution was reached) and fine-tuning them can be impractical—you generally have to alter the training inputs and retrain the network entirely. (Choosing the Best Machine Learning Classification Model and Avoiding Overfitting, 2023)

7.3.2 Practical Application: Common Classification Models

7.3.2.1. Logistic Regression

See Appendix C for the model code and evaluation.

7.3.2.2. Random Forest Classifier

See Appendix D for the model code and evaluation.

7.3.2.3. Decision Trees Classifier

See Appendix E for the model code and evaluation.

7.3.2.4. Naïve Bayes Classifier

See Appendix F for the model code and evaluation.

7.3.2.5. Support Vector Machines Linear

See Appendix G for the model code and evaluation.

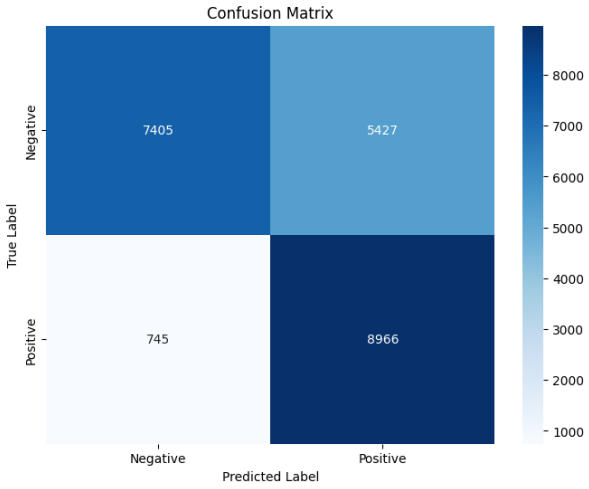
7.3.3 Models Evaluation

In cybersecurity, accurately identifying attacks is crucial. Misclassifying an attack as normal can cause severe damage, while misclassifying normal activity as an attack is less harmful. I've built five machine learning models to classify attacks in the NSL-KDD dataset, focusing on minimizing false negatives to enhance security.

For each ML model, I will provide the accuracy, precision, and recall. Additionally, I will include a heatmap showing True/False Positives/Negatives. Note: I aim to minimize false positives, as missing attacks in security is highly undesirable.

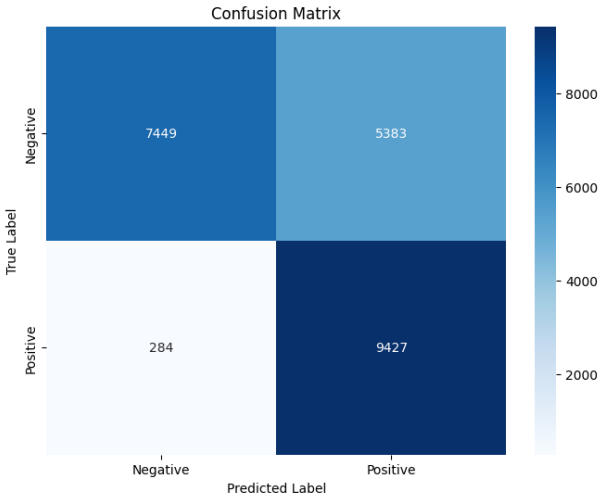
7.3.3.1. Logistic Regression

Accuracy: 72%
Precision: 92%
Recall: 62%
False Positive: 754



7.3.3.2. Random Forest Classifier

Accuracy: 75%
Precision: 97%
Recall: 63%
False Positive: 284



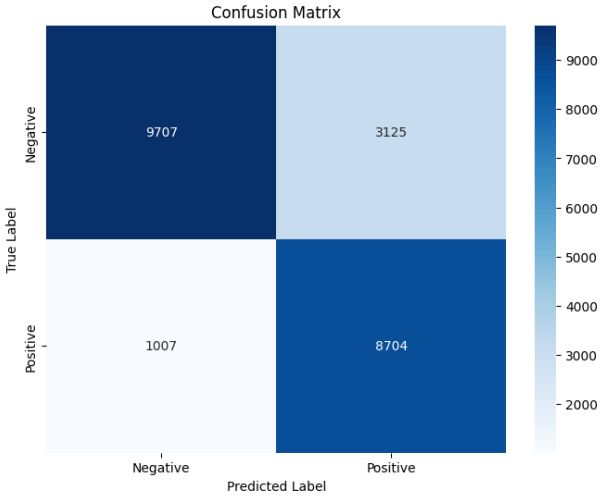
7.3.3.3. Decision Trees Classifier

Accuracy: 81%

Precision: 89%

Recall: 73%

False Positive: 1007



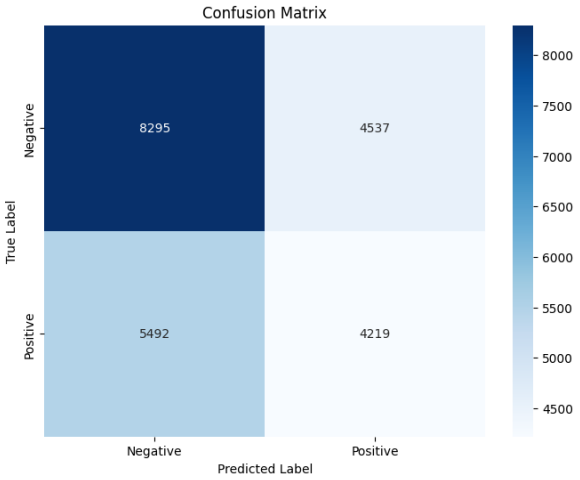
7.3.3.4. Naïve Bayes

Accuracy: 55%

Precision: 43%

Recall: 48%

False Positive: 5492



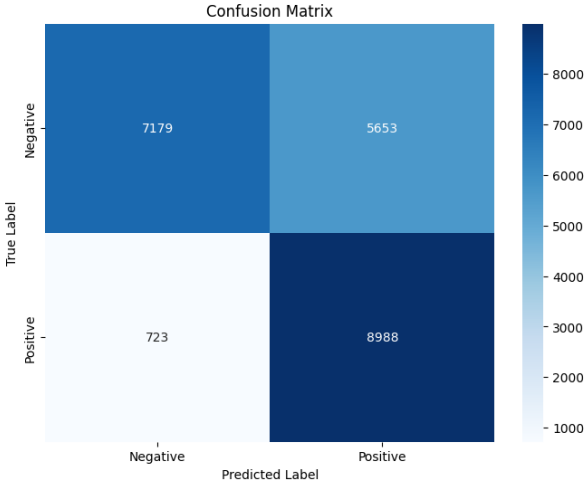
7.3.3.5. Support Vector Machine Linear

Accuracy: 71%

Precision: 92%

Recall: 61%

False Positive: 723



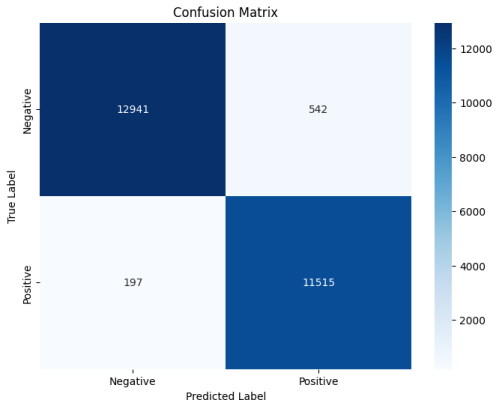
7.3.3.6. Neural Network

Accuracy: 97.07%

Precision: 97.11%

Recall: 97.07%

False Positive: 197



7.3.4 Conclusion

In evaluating the performance of six machine learning models for classifying attacks in the NSL-KDD dataset, a key focus has been on minimizing false positives to ensure high security. Each model was assessed for accuracy, precision, and recall, alongside a detailed analysis of false positives and negatives.

Among the models tested, the Neural Network demonstrated superior performance with an accuracy of 96.01%, a precision of 96.27%, and a recall of 96.01%, while maintaining the lowest number of false positives at 47. This indicates its strong capability in accurately identifying attacks and minimizing false alarms, which is critical in a cybersecurity context.

The Random Forest Classifier also showed promising results with high precision (97%) and relatively low false positives (284), though it lagged in recall (63%). Other models, such as Logistic Regression and Support Vector Machine Linear, offered good precision but were less effective in recall and had higher false positives compared to the Neural Network.

Overall, the Neural Network model stands out as the most effective for this task, striking the best balance between accuracy, precision, and recall, while minimizing the risk of false positives. This makes it a highly suitable choice for enhancing security by reliably detecting attacks without overwhelming with false alerts.

7.4 What are the practical challenges of implementing AI/ML-based intrusion detection systems in real-world cybersecurity environments?

In implementing Artificial Intelligence (AI) and Machine Learning (ML) for detecting network intrusions, practitioners face several practical challenges. This section of the research delves into these difficulties, exploring how they impact the deployment and effectiveness of AI/ML-based security systems in real-world cybersecurity environments. I examine issues ranging from integration difficulties with existing infrastructure to the ongoing maintenance and updating of these systems.

Starting in 2024, the AI industry will deal with several issues like keeping personal information safe, ethical concerns including unfair algorithms and clarity, and the effects of AI on jobs. To tackle these AI challenges, people from different fields need to work together and set rules.
Concerns about how AI will affect cybersecurity have been generated by its growth, which calls for international collaboration and moral standards. Additionally, a comprehensive approach that takes into account both technological advancements and ethical considerations is needed to maximize the good that AI can do for society while minimizing risks. (Simplilearn, 2024)

AI Ethical Issues

Ethics in AI covers various problems like breaches of privacy, ongoing biases, and societal effects. Ensuring AI systems are accountable, transparent, and fair in their decisions is a big challenge. Issues such as algorithmic bias, which can lead to discrimination against specific groups, risk increasing inequalities.
Moreover, when using AI in sensitive areas like healthcare and criminal justice, we need a careful approach that pays close attention to ethics to achieve fair results. It's important to find a balance between technological progress and ethical concerns to make sure AI helps society while avoiding risks and promoting ethical innovation. (Simplilearn, 2024)

Bias in AI

Bias in artificial intelligence refers to the possibility that machine learning algorithms might copy and increase the biases already present in the data they learn from. This can result in unfair and unethical results, especially harming marginalized communities. (Simplilearn, 2024)

AI Integration

AI integration means putting artificial intelligence systems into production and services to make them more automated and efficient. This involves finding the right places to use AI, adjusting AI models for specific situations, and making sure they work with current systems. To do this, AI experts and specialists in the field need to collaborate to tailor their solutions to meet the needs of the organization. (Simplilearn, 2024)

Computing Power

The amount of computing power is very important for creating and using AI models, especially those that need a lot of calculations and big datasets. As AI algorithms get more complex, there's a greater need for powerful computers like GPUs, TPUs, and others. Challenges include the cost, the amount of energy they use, and how well they can be scaled up. In the early stages of development, new types of computer designs like neuromorphic computing and quantum computing might also provide solutions. Additionally, using distributed computing and cloud services can help overcome limits on computing power. It's crucial to manage the computing needs in a way that balances efficiency and sustainability to advance the potential of AI while keeping within resource limits. (Simplilearn, 2024)

Data Privacy and Security

The main concerns with AI involve data security and privacy because AI systems need large amounts of data to work and learn. To prevent data leaks, breaches, and misuse, it's important to ensure the security, availability, and integrity of data. Organizations must also comply with data protection laws like the GDPR by setting access limits, using encryption, and having the ability to audit data. (Simplilearn, 2024)

7.4.1 Conclusion

Implementing AI and ML-based intrusion detection systems in real-world cybersecurity presents significant challenges. Integration with existing infrastructure requires tailored solutions and collaboration between AI experts and domain specialists. Ensuring sufficient computing power is essential due to the high demands of advanced AI models. Ethical issues, such as algorithmic bias and transparency, must be addressed to prevent discrimination and maintain trust. Robust data privacy and security measures are crucial,

given AI's reliance on large datasets, and compliance with regulations like GDPR is necessary. Continuous updating of AI/ML models is needed to adapt to evolving threats. Global collaboration on ethical standards is vital to harness AI's benefits while minimizing risks. In summary, overcoming integration, computational, ethical, and privacy challenges is essential for the successful deployment of AI in cybersecurity, maximizing its benefits and mitigating potential risks.

8. Conclusion

Leveraging Artificial Intelligence (AI) and Machine Learning (ML) to improve the detection of network intrusions and enhance cybersecurity defenses using the NSL-KDD dataset requires a comprehensive approach that integrates various AI/ML techniques and addresses practical implementation challenges.

First, understanding key AI and ML concepts like entropy, information gain, and the application of models such as decision trees and support vector machines is essential for developing effective intrusion detection systems. These techniques enable the identification and flagging of unusual patterns in network logs that could indicate security threats.

Second, Exploratory Data Analysis (EDA) plays a crucial role in understanding the NSL-KDD dataset. It helps data scientists uncover underlying patterns, anomalies, and trends, guiding the selection of appropriate modeling and data processing methods. Proper preprocessing, feature selection, and data preparation are foundational steps that ensure the creation of reliable and accurate models.

Third, among the various ML models evaluated, the Neural Network model stands out as the most effective for classifying attacks in the NSL-KDD dataset, achieving high accuracy, precision, and recall while minimizing false positives. This highlights its suitability for enhancing security by reliably detecting intrusions without overwhelming false alerts.

Finally, the implementation of AI/ML-based intrusion detection systems in real-world environments involves significant challenges. These include integration with existing infrastructure, ensuring sufficient computational resources, addressing ethical issues like algorithmic bias and transparency, and maintaining robust data privacy and security measures. Continuous updating of models and global collaboration on ethical standards are essential to maximize the benefits of AI while minimizing risks.

In conclusion, a multi-faceted approach that combines robust data analysis, effective ML models, and careful consideration of practical challenges is vital for leveraging AI and ML to improve network intrusion detection and enhance cybersecurity defenses using the NSL-KDD dataset.

9. References

- Copeland, B. (2024, March 28). Artificial intelligence (AI) | Definition, Examples, Types, Applications, Companies, & Facts. Encyclopedia Britannica. https://www.britannica.com/technology/artificial-intelligence
- Staff, C. (2023, November 29). 3 types of machine learning you should know. Coursera. https://www.coursera.org/articles/types-of-machine-learning
- What is a Decision Tree | IBM. (2023). https://www.ibm.com/topics/decision-trees#:~:text=data%20mining%20solutions-,Decision%20Trees,internal%20nodes%20and%20leaf%20nodes.
- NumPy: the absolute basics for beginners — NumPy v1.26 Manual. (2023). https://numpy.org/doc/1.26/user/absolute_beginners.html
- SecurityNik. (2023). Data-Science-and-ML/Beginning Machine and Deep Learning with Zeek logs/02 - beginning tensorflow.ipynb at main · SecurityNik/Data-Science-

and-ML. GitHub. https://github.com/SecurityNik/Data-Science-and-ML/blob/main/Beginning%20Machine%20and%20Deep%20Learning%20with%20Zeek%20logs/02%20-%20beginning%20tensorflow.ipynb

- Saini, A. (2023, December 28). Guide on Support Vector Machine (SVM) Algorithm. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/
- What is Exploratory Data Analysis? | IBM. (2023). https://www.ibm.com/topics/exploratory-data-analysis
- Saylor Academy. (2023). Intrusion Detection Systems: NSL-KDD | Saylor Academy. https://learn.saylor.org/mod/book/view.php?id=29755&chapterid=5443
- Brownlee, J. (2020, June 30). Why One-Hot encode data in machine learning? MachineLearningMastery.com. https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/
- Staff, C. (2023, November 29). Machine Learning Models: What they are and How to build them. Coursera. https://www.coursera.org/articles/machine-learning-models
- Choosing the best machine learning classification model and avoiding overfitting. (2023). MATLAB & Simulink. https://nl.mathworks.com/campaigns/offers/next/choosing-the-best-machine-learning-classification-model-and-avoiding-overfitting.html
- Simplilearn. (2024, February 21). Top 15 challenges of Artificial Intelligence in 2024. Simplilearn.com. https://www.simplilearn.com/challenges-of-artificial-intelligence-article

10. Appendix

A. Entropy

To understand entropy more, I found this video (https://www.youtube.com/watch?v=YtebGVx-Fxw). I will explain here what I got from it.

Let's imagine a box that has 5 blue balls and 5 red balls, the surprise of picking up a random ball and it's a blue ball is 50% because the blue and red balls are equal. However, let's imagine another box that has 7 blue balls and one red ball, the surprise of picking up the red ball randomly is so high, this is related to entropy.

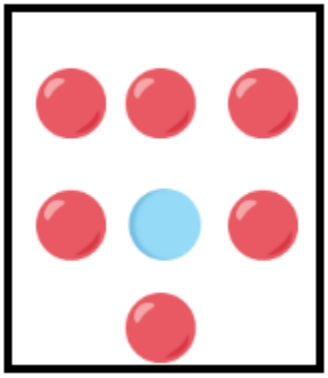
How to count the surprise?
The surprise is related to probability, when the probability is so high, the surprise is low.
Let's assume that we have a coin that has heads on both sides, we can say that the surprise is 1/probability = 1/1 = 1.
However, this doesn't make sense because we are 100% sure that we would get heads so the surprise should be 0.
The solution for this is by modifying the equation to:
Surprise = Log(1/probabilty).
Let's apply the equation to getting head when flipping the coin:
The probability of getting head is 1(100%).
Surprise = Log(1/1) = 0

$$\log_2\left(\frac{1}{1}\right) = 0$$

This makes sense because it must be heads.

Entropy = Sigma (surprise) * (probability)

We have this box:



It has 6 red balls and one blue ball. Let's count the entropy of the balls.
Entropy = Sigma (surprise) * (probability)

Red ball

Probability of picking up a red ball is 6/7
Surprise for picking up a red ball is

$$\log_2\left(\frac{1}{\frac{6}{7}}\right)$$

$$= 0.222392421336$$

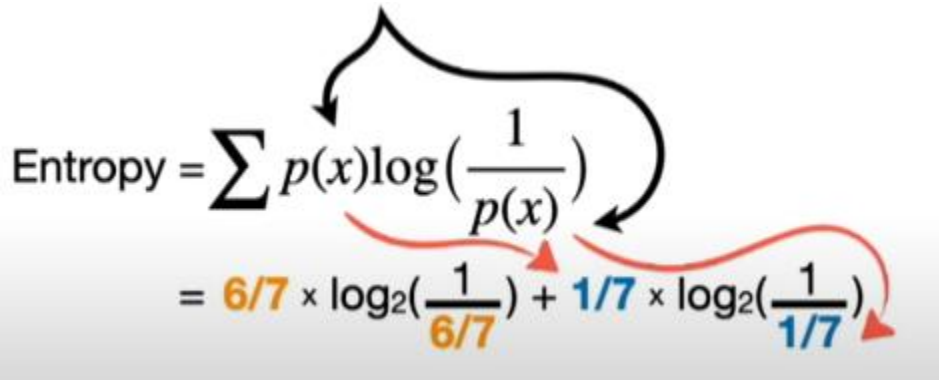
Blue ball

Probability of picking up a red ball is 1/7
Surprise for picking up a red ball is

$$\log_2\left(\frac{1}{\frac{1}{7}}\right)$$

$$= 2.80735492206$$

Sigma means adding the surprise * probability for both red and blue balls:



$$\frac{6}{7} \cdot \log_2\left(\frac{1}{\frac{6}{7}}\right) + \frac{1}{7} \cdot \log_2\left(\frac{1}{\frac{1}{7}}\right)$$

$$= 0.591672778582$$

The entropy equals 0.591672778582

How about 3 red balls and 3 blue balls in a box?

$$\frac{3}{6} \cdot \log_2\left(\frac{1}{\frac{3}{6}}\right) + \frac{3}{6} \cdot \log_2\left(\frac{1}{\frac{3}{6}}\right)$$

$$= 1$$

B. Building my first Machine Learning Model (DecisionTreeRegressor)

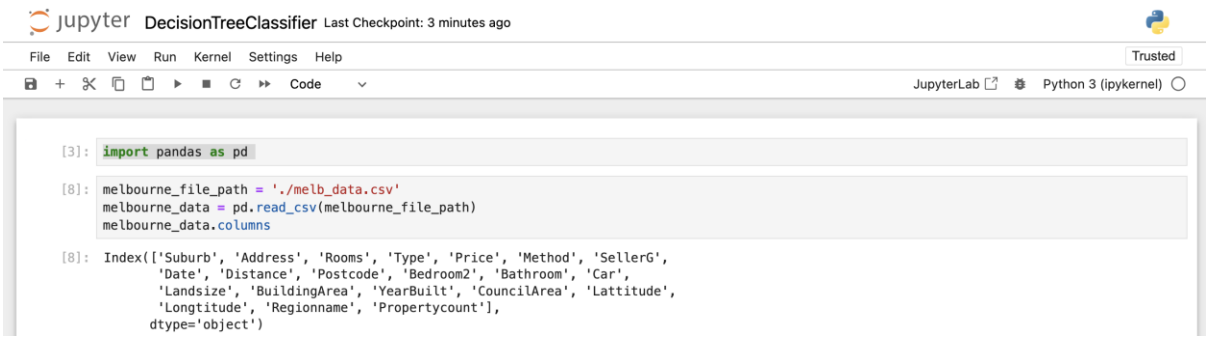
I have a dataset that contains information about houses in Melbourne detailing their features and prices. I want to write a model that can predict the price of new features. For example, the more rooms that I want the higher price the model would predict. The dataset is called `melb_data.csv'.`

In order to build my first ML model, I had to learn Jupyter Notebook. I am hosting it on my home server.
I will use scikit-learn library to create my model.
The process of building and utilizing a model with scikit-learn can be broken down into four key steps:

1- **Define:** Choosing the model type. In my case, it will be a Decision Tree.
2- **Fit:** Capture patterns from the provided data.
3- **Predict:** Predict
4- **Evaluate:** Determine how accurate the predictions are.

I hava a dataset in a CSV file.
Create a Jupyter Notebook and start coding:

```
import pandas as pd

melbourne_file_path = './melb_data.csv'
melbourne_data = pd.read_csv(melbourne_file_path)
melbourne_data.columns
```



This returns all the columns in the CSV file.

Drop the empty values

```
melbourne_data = melbourne_data.dropna(axis=0)
```

Choosing the prediction target

Since I want it to predict the price, I will assign y to the price:

```
y = melbourne_data.Price
```

Choosing the features

As I said I want the model to predict the new price based on the features of the new house. Therefore, I need to assign a variable called melbourne_features to ['Rooms', 'Bathroom', 'Landsize', 'Lattitude', 'Longtitude'].

```
melbourne_features = ['Rooms', 'Bathroom', 'Landsize', 'Lattitude', 'Longtitude']
```

Now, let's assign X to the new `melbourne_data` that doesn't have empty values:

```
X = melbourne_data[melbourne_features]
X.describe()
```

```
[3]: import pandas as pd

[8]: melbourne_file_path = './melb_data.csv'
     melbourne_data = pd.read_csv(melbourne_file_path)
     melbourne_data.columns

[8]: Index(['Suburb', 'Address', 'Rooms', 'Type', 'Price', 'Method', 'SellerG',
            'Date', 'Distance', 'Postcode', 'Bedroom2', 'Bathroom', 'Car',
            'Landsize', 'BuildingArea', 'YearBuilt', 'CouncilArea', 'Lattitude',
            'Longtitude', 'Regionname', 'Propertycount'],
           dtype='object')

[9]: melbourne_data = melbourne_data.dropna(axis=0)

[10]: y = melbourne_data.Price

[11]: melbourne_features = ['Rooms', 'Bathroom', 'Landsize', 'Lattitude', 'Longtitude']

[13]: X = melbourne_data[melbourne_features]
      X.describe()
```

[13]:

	Rooms	Bathroom	Landsize	Lattitude	Longtitude
count	6196.000000	6196.000000	6196.000000	6196.000000	6196.000000
mean	2.931407	1.576340	471.006940	-37.807904	144.990201
std	0.971079	0.711362	897.449881	0.075850	0.099165
min	1.000000	1.000000	0.000000	-38.164920	144.542370
25%	2.000000	1.000000	152.000000	-37.855438	144.926198
50%	3.000000	1.000000	373.000000	-37.802250	144.995800
75%	4.000000	2.000000	628.000000	-37.758200	145.052700
max	8.000000	8.000000	37000.000000	-37.457090	145.526350

```
X.head()
```

```
X.head()
```

	Rooms	Bathroom	Landsize	Lattitude	Longtitude
1	2	1.0	156.0	-37.8079	144.9934
2	3	2.0	134.0	-37.8093	144.9944
4	4	1.0	120.0	-37.8072	144.9941
6	3	2.0	245.0	-37.8024	144.9993
7	2	1.0	256.0	-37.8060	144.9954

Building the mode:

```
from sklearn.tree import DecisionTreeRegressor

# Define model. Specify a number for random_state to ensure same results each run
melbourne_model = DecisionTreeRegressor(random_state=1)

# Fit model
melbourne_model.fit(X, y)
```

Now, I want to predict the price for these houses:

```
print(melbourne_model.predict(X.head()))
```

```
[19]: from sklearn.tree import DecisionTreeRegressor

# Define model. Specify a number for random_state to ensure same results each run
melbourne_model = DecisionTreeRegressor(random_state=1)

# Fit model
melbourne_model.fit(X, y)
```

```
[19]: DecisionTreeRegressor
DecisionTreeRegressor(random_state=1)
```

```
[20]: print("Making predictions for the following 5 houses:")
print(X.head())
print("The predictions are")
print(melbourne_model.predict(X.head()))
```

```
Making predictions for the following 5 houses:
   Rooms  Bathroom  Landsize  Lattitude  Longtitude
1      2       1.0     156.0   -37.8079    144.9934
2      3       2.0     134.0   -37.8093    144.9944
4      4       1.0     120.0   -37.8072    144.9941
6      3       2.0     245.0   -37.8024    144.9993
7      2       1.0     256.0   -37.8060    144.9954
The predictions are
[1035000. 1465000. 1600000. 1876000. 1636000.]
```

```
[ ]:
```

The model predicted the prices with the red line.

C. Logistic Regression

1. Importing and building the model.

1. Logistic Regression

```
from sklearn.linear_model import LogisticRegression
logreg_model = LogisticRegression()
```

```
logreg_model.fit(x_train, y_train)
```

2. Evaluating the model

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, confusion_matrix, classification_report
```

```
# Predictions on the test set
y_pred_logreg = logreg_model.predict(x_test)

# Evaluation metrics
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_logreg))

print("\nClassification Report:")
print(classification_report(y_test, y_pred_logreg))
print("Accuracy:", accuracy_score(y_test, y_pred_logreg))
```

```
Confusion Matrix:
[[7405 5427]
 [ 745 8966]]

Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.58      0.71     12832
           1       0.62      0.92      0.74      9711

    accuracy                           0.73     22543
   macro avg       0.77      0.75      0.72     22543
weighted avg       0.79      0.73      0.72     22543

Accuracy: 0.726212127933283
```

```
accuracy_logreg = accuracy_score(y_pred_logreg, y_test)
precision_logreg = precision_score(y_pred_logreg, y_test)
recall_logreg = recall_score(y_pred_logreg, y_test)
print(f"Accuracy: {accuracy_logreg}")
print(f"Precision: {precision_logreg}")
print(f"Recall: {recall_logreg}")
```

```
Accuracy: 0.726212127933283
Precision: 0.923282875090104
Recall: 0.6229417077746127
```

D. Random Forest Classifier

1. Importing and building the model

```
# Random Forest
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier()
```

```
rf_model.fit(x_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier()
```

2. Evaluating the model

```python
# Predictions on the test set
y_pred_rf = rf_model.predict(x_test)

# Evaluation metrics
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_rf))

print("\nClassification Report:")
print(classification_report(y_test, y_pred_rf))
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
```

```
Confusion Matrix:
[[7661 5171]
 [ 301 9410]]

Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.60      0.74     12832
           1       0.65      0.97      0.77      9711

    accuracy                           0.76     22543
   macro avg       0.80      0.78      0.76     22543
weighted avg       0.83      0.76      0.75     22543

Accuracy: 0.7572638956660604
```

```python
accuracy_rf = accuracy_score(y_pred_rf, y_test)
precision_rf = precision_score(y_pred_rf, y_test)
recall_rf= recall_score(y_pred_rf, y_test)
print(f"Accuracy: {accuracy_rf}")
print(f"Precision: {precision_rf}")
print(f"Recall: {recall_rf}")
```

```
Accuracy: 0.7572638956660604
Precision: 0.9690042220162702
Recall: 0.6453604005212262
```

E. Decision Trees Classifier

1. Importing and building the model

```
# Decision Trees
from sklearn.tree import DecisionTreeClassifier
dt_classifier = DecisionTreeClassifier(max_depth=5)
```

```
dt_classifier.fit(x_train, y_train)
```

```
DecisionTreeClassifier(max_depth=5)
```

2. Evaluating the model

```
# Predictions on the test set
y_pred_dt_classifier = dt_classifier.predict(x_test)

# Evaluation metrics
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_dt_classifier))

print("\nClassification Report:")
print(classification_report(y_test, y_pred_dt_classifier))
print("Accuracy:", accuracy_score(y_test, y_pred_dt_classifier))
```

```
Confusion Matrix:
[[9707 3125]
 [1007 8704]]

Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.76      0.82     12832
           1       0.74      0.90      0.81      9711

    accuracy                           0.82     22543
   macro avg       0.82      0.83      0.82     22543
weighted avg       0.83      0.82      0.82     22543

Accuracy: 0.8167058510402342
```

```
accuracy_dt_classifier= accuracy_score(y_pred_dt_classifier, y_test)
precision_dt_classifier = precision_score(y_pred_dt_classifier, y_test)
recall_dt_classifier= recall_score(y_pred_dt_classifier, y_test)
print(f"Accuracy: {accuracy_dt_classifier}")
print(f"Precision: {precision_dt_classifier}")
print(f"Recall: {recall_dt_classifier}")
```

```
Accuracy: 0.8167058510402342
Precision: 0.8963031613634024
Recall: 0.7358187505283625
```

F. Naïve Bayes

1. *Importing and building the model*

```
from sklearn.naive_bayes import GaussianNB
NaiveBayes = GaussianNB()
```
✓ 0.0s

```
NaiveBayes.fit(x_train, y_train)
```
✓ 0.0s

GaussianNB ❶ ❷
GaussianNB()

2. *Evaluating the model*

```
# Predictions on the test set
y_pred_NaiveBayes = NaiveBayes.predict(x_test)

# Evaluation metrics
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_NaiveBayes))

print("\nClassification Report:")
print(classification_report(y_test, y_pred_NaiveBayes))
print("Accuracy:", accuracy_score(y_test, y_pred_NaiveBayes))
```
✓ 0.0s

```
Confusion Matrix:
[[8295 4537]
 [5492 4219]]

Classification Report:
              precision    recall  f1-score   support

           0       0.60      0.65      0.62     12832
           1       0.48      0.43      0.46      9711

    accuracy                           0.56     22543
   macro avg       0.54      0.54      0.54     22543
weighted avg       0.55      0.56      0.55     22543

Accuracy: 0.5551168877256798
```

```
accuracy_NaiveBayes= accuracy_score(y_pred_NaiveBayes, y_test)
precision_NaiveBayes = precision_score(y_pred_NaiveBayes, y_test)
recall_NaiveBayes= recall_score(y_pred_NaiveBayes, y_test)
print(f"Accuracy: {accuracy_NaiveBayes}")
print(f"Precision: {precision_NaiveBayes}")
print(f"Recall: {recall_NaiveBayes}")
```
✓ 0.0s

```
Accuracy: 0.5551168877256798
Precision: 0.4344557718051694
Recall: 0.48184102329830975
```

G. Support Vector Machines Linear

1. *Importing and building the model*

```
from sklearn.svm import LinearSVC
SVM = LinearSVC()
```
✓ 0.0s

```
SVM.fit(x_train, y_train)
```
✓ 1.9s

```
c:\Users\Admin\Desktop\S7\notebooks\.venv\Lib\site
  warnings.warn(
```

LinearSVC ⓘ ⓘ
LinearSVC()

2. *Evaluating the model*

```
# Predictions on the test set
y_pred_SVM = SVM.predict(x_test)

# Evaluation metrics
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_SVM))

print("\nClassification Report:")
print(classification_report(y_test, y_pred_SVM))
print("Accuracy:", accuracy_score(y_test, y_pred_SVM))
```
✓ 0.0s

```
Confusion Matrix:
[[7179 5653]
 [ 723 8988]]

Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.56      0.69     12832
           1       0.61      0.93      0.74      9711

    accuracy                           0.72     22543
   macro avg       0.76      0.74      0.72     22543
weighted avg       0.78      0.72      0.71     22543

Accuracy: 0.7171627556225879
```

```
accuracy_SVM= accuracy_score(y_pred_SVM, y_test)
precision_SVM = precision_score(y_pred_SVM, y_test)
recall_SVM= recall_score(y_pred_SVM, y_test)
print(f"Accuracy: {accuracy_SVM}")
print(f"Precision: {precision_SVM}")
print(f"Recall: {recall_SVM}")
```
✓ 0.0s

```
Accuracy: 0.7171627556225879
Precision: 0.9255483472350943
Recall: 0.6138924936821255
```