DATA SCIENCE TECHNIQUES FOR BEGINNER NETWORK ADMINISTRATORS

Sanksi Pelanggaran Pasal 113
Undang-Undang No. 28 Tahun 2014 Tentang Hak Cipta

1. Setiap Orang yang dengan tanpa hak melakukan pelanggaran hak ekonomi sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf i untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 1 (satu) tahun dan/atau pidana denda paling banyak Rp100.000.000 (seratus juta rupiah).
2. Setiap Orang yang dengan tanpa hak dan/atau tanpa izin Pencipta atau pemegang Hak Cipta melakukan pelanggaran hak ekonomi Pencipta sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf c, huruf d, huruf f, dan/atau huruf h untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 3 (tiga) tahun dan/atau pidana denda paling banyak Rp500.000.000,00 (lima ratus juta rupiah).
3. Setiap Orang yang dengan tanpa hak dan/atau tanpa izin Pencipta atau pemegang Hak Cipta melakukan pelanggaran hak ekonomi Pencipta sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf a, huruf b, huruf e, dan/atau huruf g untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 4 (empat) tahun dan/atau pidana denda paling banyak Rp1.000.000.000,00 (satu miliar rupiah).
4. Setiap Orang yang memenuhi unsur sebagaimana dimaksud pada ayat (3) yang dilakukan dalam bentuk pembajakan, dipidana dengan pidana penjara paling lama 10 (sepuluh) tahun dan/atau pidana denda paling banyak Rp4.000.000.000,00 (empat miliar rupiah).

DATA SCIENCE TECHNIQUES FOR BEGINNER NETWORK ADMINISTRATORS

Yulius Yus
(Computer Engineering, Keling Kumang Institute of Technology, West Kalimantan)

Azriel Christian Nurcahyo
(PhD Student in Computing, University of Technology Sarawak)

Heni Ermewaningsih
(Computer Engineering, Keling Kumang Institute of Technology, West Kalimantan)

AMERTA
MEDIA

DATA SCIENCE TECHNIQUES FOR BEGINNER NETWORK ADMINISTRATORS

Anggota IKAPI
No 192JTE/2020
Cetakan Pertama: Agustus 2024
15,5 cm x 23 cm
ISBN: 978-623-419-672-6

Penulis:
Yulius Yus
Azriel Christian Nurcahyo
Heni Ermewaningsih

Desain Cover:
Dwi Prasetyo

Tata Letak:
Ladifa Nanda

Diterbitkan Oleh:
Penerbit Amerta Media

Jl. Raya Sidakangen, RT 001 RW 003, Kel, Kebanggan, Kec. Sumbang,
Purwokerto, Banyumas 53183, Jawa Tengah. Telp. 081-356-3333-24
Email: mediaamerta@gmail.com
Website: amertamedia.co.id
Whatsapp : 081-356-3333-24

FOREWORD

I am happy to present this book ***"Data Science Techniques for Beginner Network Administrators, Edition1"*** to give network administrators a deeper and wider insight and to help improving their skills. We put particular emphasis on the intention of this book at this moment is to explain and apply increasing relevant data science technologies to an area all the rage in today's digital era driven primarily by rapid advances in artificial intelligence, network administration. In this book, we choose Google Colab as service provides them with data processing capability.

The role of a network administrator has been greatly changed. Formerly they confined themselves to handling of hardware and software for backbone routers, core switches, access points, bandwidth, plus server and hosting management. With today's big data era and the rapid progress of artificial intelligence the network administrator is required not only to carry out data analysis (whether it be offline or conversational) so as better network performance and security can be achieved, but they also needs to do many other tasks related to data management. By using data science techniques, network administrators can monitor their networks more effectively, faster, and more securely which will result in a more efficient IT infrastructure.

We hope that through this book will the network administrator can acquire the basic knowledge and practical skills necessary to integrate data science directly into his daily work. The book covers different essential topics from data collection and preprocessing to applying various kinds of machine learning algorithms on network data analysis such as supervised, unsupervised and simple deep learning.

Each chapter is designed to give an all around and systematic analysis. Readers will be led through the core concepts of data science and how these can be applied in network administration. At the same time, the book also contains examples of network models and practical exercises that aim to help readers practice what they have learnt when they are out there in the real world.

We hope that the reader will find not only a convenient reference work to which he often refers but also it may be an inspiration for further study. IT Marketers, networking people, systems administrators and other professional groups in the field of information technology will probably

discover before long what has dawned like a glorious sunrise just recently, as time passes by more and more of them will need to study this book by making full use of its data to enhance their skills in future face problems.

Better wake people from deep sleep than call up those asleep.
Happy learning!

Sibu, Sarawak, 20 July 2024

Azriel Christian Nurcahyo

WELCOME SPEECH

Data Science Techniques for Beginner Network Administrators, Edition 1 It offers network administrators and server managers something far beyond as they have generally been trained to manage operations which tend towards managing pieces of hardware with software. Graduates of Computer Engineering should know more than just how to create a system, moreover they have to use some technique or method in delivering the data and visualize it for classifying and predicting with an accurate results.

Today in the fast changing age innovation is something which will lead organisations to their existence and growth. New ways of thinking stand out among network admins and server hosts, who continue to break away from antiquated 'from the ground up' mentalities that concentrate solely on hardware or system models. In other words, data science is a vital and ever green skill in the job market today. By becoming proficient in data science, they can create cutting-edge network analysis or optimalization or security applications that are sure to provide a substantial amount of added value for their organization.

Our point here is that Data Science does not just belong to the groups of data scientists or statisticians. Similarly network engineers and server managers can there also great scope to learn & implement data science techniques. This skill set will not only allow them to be more knowledgeable but also qualify for a wider spectrum of job prospects.

This book prepares network administrators for this journey. We will be collecting all practical and needed materials related to networking from another the field, n this notebook includes topics such as data gathering or preprocessing up until appying maching learning or deep learning algorithms. Add to that real world examples and some exercises on the book, this will instruct you with what's knowieden e on every chapter. For the implementation of data management we use Google Colab too that is pretty straightforward to learn and can be done from a computer with an internet connection plus having only some Google colab account.

We wish this book can motivate students, network hobbyists or server and network experts to learn more. Use data science to get better and achieve the best.

It is better to awaken the man who sleeps than him that feigns sleep!
Keep reading and keep learning.

Sekadau, 20 July 2024

Head of the Department of
Computer Engineering,
Keling Kumang Institute of Technology, West Kalimantan

TABLE OF CONTENTS

FOREWORD ..... **v**
WELCOME SPEECH ..... **vii**
TABLE OF CONTENTS ..... **ix**

CHAPTER 1 SUPERVISED LEARNING **1**
A. Classification 1
B. K-Nearest Neighbors (KNN) 7
C. Decision Tree 16
D. Regression 30
E. Simple Case Study Model for Regression 44
F. Ordinary Least Squares Regression or OLS Regression 50

CHAPTER 2 UNSUPERVISED LEARNING PART 1 ..... **59**
A. Background 59
B. Kernel Density Estimation 74
C. Dimensionality Reduction 81

CHAPTER 3 UNSUPERVISED LEARNING PART 2 ..... **95**
A. Clustering 95

REFERENCES **110**
BIOGRAPHY OF THE AUTHOR **111**

Chapter 1

SUPERVISED LEARNING

A. Classification

In machine learning, supervised learning is one of the most basic methods which uses labeled data and trains an algorithm to generate optimal results from that sample. As a result, this technique has extensive use in teaching the model of input output mappings and make it predict for new unseen data.

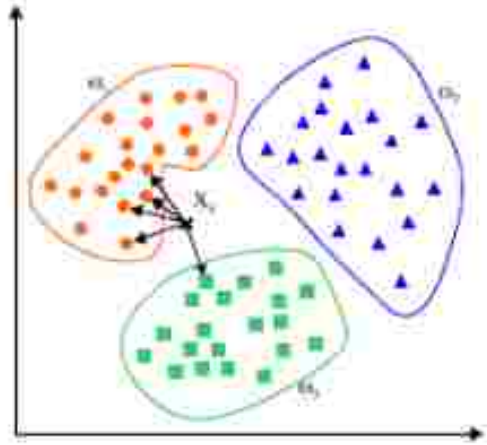


Figure 1. KNN Model (source from www.mathworks.com)

In supervised learning, we feed both the features and labels to train our model. During this process, your training data is comprised of the inputs and outputs that you feed into a predictive model. We then utilize the model in order to make predictions on new unlabeled data points. Classification, and Regression are the two main types of tasks in supervised learning. In classification, the data is dividing into categorical values that are based on predefined criteria (eg. spam or not spam). We predict continuous numerical values based on an input data like predicting prices

of a hosting service (Rp./monthly) based on features such as size, stanp location and conditions etc. Supervised learning has some features making it significantly different from other types of machine learning, such as unsupervised or reinforcement learning. Supervised learning uses labeled data with directed (instead of direct) learning to achieve predictive goals, requiring careful performance evaluation. Supervised learning must be trained by using labelled datasets.

These labeled data represent the known input-output pair that can help the model to understand or learn how inputs are mapping into outputs. In the text classification implementation, for example classifying mails into spam or not-spam category then each of the emails in your training data has a label telling you if its a spam. Labeled data is important, because it helps the model to learn a vast and more structured way.

Another characteristic of supervised learning is directed learning. The model is trained to map inputs to outputs on a small set of labeled data. Its objective is to reduce the error between its predictions and actual outputs for each training data. Training happens by using optimization methods to optimize the model's parameters via minimization of a loss function (predictive error, measurable from comparing predictions vs. actual labels). Over time, the model learns to do this better and better.

However, supervised learning is not only easier to interpret and makes very clear predictive goals (test your model with untouched data), it also has an enterprise feel, after training the model knows how to produce labels for new inputs a priori. As an example, under the domain of network administration supervised learning can be applied to categorize different instances or servers. Training data may consist Server performance metrics, device age, maintenance history. A model trained with this data can predict the need for maintenance or replacement of a specific server given its observed features.

The second important point is the evaluation of performances in supervised learning. Performance metrics like accuracy, precision, recall and F1-score are used to evaluate the trained model. This is usually done by considering of part of the dataset for training and remainder to testing. The training set is used to fit the model, while the testing set evaluates its performance. Cross-validation is also one of the most common methods to more effectively evaluate the model. For example, in cross-validation your dataset is divided into various sets and your model is trained on combination of these to ensure its ability to generalize among new data. Act as a classifier in terms of network admin use cases identifying types of

devices or predicting device failures. For instance, the administrators can prepare a model on whether servers are new, need maintenance or even outdated. This training set would have a number of performance metrics, CPU load, memory usage and response times at key tiers alongside contextual information such as device age, maintenance history etc.

Supervised learning algorithms (for example K-Nearest Neighbors or Decision Tree, Random Forest) can be used to develop models that the ability predict which servers are vulnerable and should be replaced due to looming failure.

Supervised learning refers to the use of data collected under efficacy circumstances for producing more rapid and high-accuracy decision making by network administrators. However, supervised learning is a great tool for increasing efficiency and effectiveness in the area of network administration (as well as numerous other fields) when used properly. Such models trained on expressive data and rigorously tested can lead to meaningful intelligence and efficient decision making in a wide range of industries, particularly those that already harness the power of all things data.

In supervised learning algorithms are trained on historical data and they can make accurate predictions based upon past observations, this makes it one of the most powerful technique in todays era. It can be used in network administration for various practical tasks. It is possible to classify servers using supervised learning, predict when devices are going to fail and which that we have never seen will behave like the rest (anomaly detector), detect types of network traffic known as malicious starting with labeled datasets. Supervised learning, when applied to network management, in device classification and predicting failure can allow administrators for example train models which classify servers according how healthy they are. The training data will be a set of historical values including performance metrics, device age and maintenance history.

However, supervised learning is considered as one of the fundamentally important approaches in machine learning. In supervised learing approach, learning algorithms learn from labeled data (input / output) to try and approximate a mapping function. Thus the model learns and generalizes input to output mapping after which it can be used for making predictions on new (unseen) data. This learning uses labeled data to train algorithms. In this book scenario, the training dataset is composed of input-output pairs standards to build a predictive model. The model can be used to predict

the output of new, unlabeled data. Main two types of tasks for supervised learning are Classification and Regression.

In general, you are given some data and your task is to classify that into categories. A very common one of these is are our favorite emails, spam or not-spam. Contrast that with regression, which is used for predicting continuous numerical values given input data (for example prices of server hosting listings based on DNS / security / storage).

Characteristics of supervised learning, which distinct it among other types of machine learning like unsupervised or reinforcement. The features are learnt from a labeled data, this is also known as directed learning and solving for predictive goals using careful evaluation of performance.

Supervised learning is a type of Machine Learning, in which you teach the model to aproximate your labels. It is a set of labeled data, which means that we already know the input output pairs from this dataset and therefore our model can learn faster about trends in these inputs and outputs. In the example of classifying emails as spam or not spam, each email used to train a binary classifier has an attached label. Labeled data is important because it allows the model to learn with constraints and direction.

Directed learning is another essential trait of supervised learning. It's the process of training a model to predict outputs based on inputs using labeled data. This aims at reducing the error between what our model is predictign and actual outputs of training data. This is done by optimization algorithms, which modify the parameters of model to minimize a loss function that calculates how different are predictions from actual labels. This allows the model to learn and make more accurate predictions with each encounter.

In addition, supervised learning has well defined prediction objectives. After the model has been trained, it can make predictions on new, unseen input data to provide labeled outputs. In other words, supervised learning (Classifying the types of servers in network administration). This could be server performance data, the age of devices and maintenance history. This data can be used to train a model that will then make predictions, such as whether it is necessary to perform maintenance on or potentially replace any given server based on observed features.

In addition to the above steps, performance evaluation is yet another important step in supervised learning. Then the trained model being tested with performance metrics like accuracy, precision, recall and F1-score. In general we do this by breaking the data up into training and testing sets. (You can think of the training set as being something you are training a

model on, and the test set which is to see how well the trained thing performs). Cross validation methods are also commonly applied to obtain a more reliable estimate of the performance of the final model.

Supervised learning is useful in the context of network administration, such as classifying devices on a network or predicting device failures. In an example, administrators may train a model that will classify servers as new, maintenance necessary or end of life. This training data could include a variety of performance metrics such as CPU usage, memory utilization and response times ndash. In addition to other information like age / maintenance history the device. By building models that can tell a system is failing or needs to be replaced, we will know which are the target servers by utilizing supervised learning algorithms (e.g., based on K-Nearest Neighbors(KNN), Decision Tree(Random Forest).

The supervised learning procedure consists of the following key steps. In this context, the necessary dataset must include information about the conditions and performance of servers relevant to maintenance decisions. This will be across metrics for example CPU, memory and response times as well as device age on the maintenance history. The notebook is primarily responsible for generating the data and creating labels indicating if a server needs maintenance or not at each example in our dataset. The quality of the data collected matters a lot over how good can be model an improvement.

The second step is data preprocessing which follows data collection stage. Typically, raw data comes with missing values, anomalies or features contaminated by barely organized noises. So we need to cleanup data and prepare that for training. It straight forward to process, such as how missing values are managed (by filling or dropping incomplete rows).

Also, we need to normalize the features because this regularizes all data in a way that prevents them from distortion due to big scaled differences. If there is a categorical columns present, like server type or data center location for example, we need to do some encoding of the such features as well. Apart from the predominating factors that ultimately secure victories, clean and structured data is vital for formulating a model which stands at par with reality.

Next step some of the supervised learning algorithms that can be used for classification tasks are K-Nearest Neighbors (KNN), Decision Tree, Random Forest, Support Vector Machine (SVM) and Naive Bayes. The algorithm you should choose depends on the type of data and your analysis goals. For instance, when the dataset involves a lot of interacting features

and less redundant ones Random Forest might work better due to multi-feature interactions it is able to mitigate. These algorithms have their advantages or disadvantages hence the selection of suitable algorithm is required in each case.

After selecting the model, next stage is about training. Ex, using training data to teach the model how map inputs like a arrow with varios angles of bend, representing them as vectors would be something that describes your repo. The training in brief optimizes the parameters of a model to minimize prediction errors. The model uses an optimization algorithm to tweak the parameters of which predictor variable is used, along with the coefficient associated with that ability. The normal procedure often includes the separation of a dataset into training and test set, which is used to train on one half othe data it its performance can then be measured using the other.

Finally, it is important to evaluate the model so that we are certain our trained model generalizes well on unseen new data. The model is evaluated in terms of accuracy, precision, recall and F1-score. Accuracy is the percentage of correct predictions made, Precision shows how many of actual Positive cases are predicted as Positives, Recall measure that out from all Actual positives how much we got right and F1-Score which takes both precision and recall in calculation. It is used to evaluate the performance of your model and base on that you can improve accuracy, efficacy accordingly.

And then we have the last step where after trained, evaluated and tuning our models now its time to use it for predicting new unlabeled data. Network administrators can use these predictions to help make decisions on server maintenance. For a given piece of machinery such as server, an example model might be able to predict that based on its current performance metrics and maintenance history it will soon need attention. By automating these queries data scientists can make predictions that help the organization in strategic planning and decision-making thereby improving operational efficiency, reducing down time.

Following the process of supervised learning, network administrators will be able to build better systems for server maintenance management, improve resource usage efficacy and identify or resolve problems before they cause catastrophic impacts on a network. In supervised learning mode, network administrators can make more informed and thus faster decisions since the data is valid. If handled in a proper way, supervised learning can be extremely beneficial to better the efficiency as well is

effectiveness of network administration among other things. Relevant and well evaluated models can offer new understandings or encourage useful decisions in many fields of data rich industry with such careful analysis.

B. K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a type of supervised learning algorithm used for classification and regression tasks. At the heart of KNN, classifying a data point in intrusion detection mechanism is done on this premise that how it's neigbors are classified. A simple yet strong assumption of this algorithm is that points which are close to each other on the manifold will lie in similar regions after they get projected.

Essentially, the KNN algorithm saved its design inside of a few primary methods. The dataset used should contain the input features and already known labels. The data would consist of information that is pertinent to the problem being solved.

Data served for most real-world nuggets is raw and could contain missing values or other anomalies. The preprocessing includes methods for handling missing values (fill-in or removal), normalizing our features to share the same scale and encoding categorical data when needed. The K stands for a number of nearest neighbour (to be voted to make anything categorised) It is very important to select the right k value because a small one can have noise, too large and it could be not generalized. The distance of the new data point from all datapoints in training set is calculated. A simple distance metric often used is called the Euclidean distance and can be computed using this formula

$$d(i,j) = \sqrt{\sum_{m=1}^{n} (x_{im} - x_{jm})^2}$$

Where $d(i,j)$ $d(i,j)$ is the Euclidean distance between points i and j, n is the number of features, xim and xjm are the values of the m-th feature for points i and j. In simpler terms, it need to find the difference between the corresponding features of the two points, square these differences, sum all the squared differences and take the square root of the sum to get the Euclidean distance.

In Python (e.g., Google Colab), this can be implemented as

```
import numpy as np
def euclidean_distance(point1, point2):
 return np.sqrt(np.sum((point1 - point2) ** 2))
```

After calculating the distances, you now have to locate k nearest points (neighbors) of your new data point. The last data point is classified on the basis of majority class among its k-nearest neighbors. If the majority of its k nearest neighbors belong to one class, then this will be assigned as the new data point's predicted class.

Since KNN classifies using that idea, it is a supervised learning algorithm. Labeled data helps the algorithm to get familiar with how to map input features into output labels during training. Since we know these labels, KNN can also be used for predicting the label of new unlabeled data. KNN would be applied as we do in network administration to segregate servers into groups like ready, busy or down.

1. Collect data

   Collect both server status and a performance metrics. It is a type of data points such as CPU Utilization, Memory Usage, Response time (RT), Device Age or Maintained History etc. Whether or not the server is required for servicing should be tagged in each data instance

2. Data Preprocessing

   Handeling all the missing values and anomalies from data. Feature normalization is also known as feature scaling and you must encode any categorical data within the features.

3. It can select k

   It is the time of selecting an absolute value for K between classes. For the most optimal result either through experimentation or cross-validation.

4. Model Training

   Train the KKN Model with analyzed data (another way to use the particion) The model learns patterns in the labeled data and how they relate between its feature inputs to output labels

5. Testing of the model

   Final testing with a separate test dataset, to asses performance The model's effectiveness can be measured with metrics like accuracy, precision, recall and F1-score.

6. Being predicted

   The trained model is used to make predictions on new, unlabeled data For example, the model can predict if a specific server is due for maintenance based on current performance metrics and past history of maintenance.

Let's say a company has a dataset with some performance metrics about their servers (CPU, memory etc) and labels if that server needs maintenance. So, for each performance measure of a fresh server being measured the KNN model computes distance between this stress & all servers from these practice dataset. The model uses the k nearest neighbors and determines if new server is usedable or needs a maintenance. It will help the network administrators to take better decision for maintaining their servers and efficient resource utilization reducing human downtime using KNN algorithm.

Given the following dataset model, we aim to classify which servers require maintenance based on various factors such as CPU Usage, Memory Usage, Response Time, and the need for maintenance. Data collection is conducted as follows

No	CPU Usage (%)	Memory Usage (%)	Response Time (ms)	Maintenance Needed
1	85	70	250	Yes
2	30	40	100	No
3	55	60	150	No
4	90	85	300	Yes
5	25	35	90	No
6	60	75	200	Yes
7	45	50	120	No
8	80	65	240	Yes
9	35	45	110	No
10	75	80	260	Yes
11	50	55	170	No
12	40	50	140	No
13	70	65	220	Yes

No	CPU Usage (%)	Memory Usage (%)	Response Time (ms)	Maintenance Needed
14	85	90	310	Yes
15	25	30	80	No
16	60	70	210	Yes
17	50	55	160	No
18	55	65	170	No
19	75	80	270	Yes
20	85	90	310	Yes
21	65	60	180	No
22	35	45	130	No
23	55	55	150	No
24	90	85	300	Yes
25	50	60	160	No
26	65	75	210	Yes
27	45	50	120	No
28	80	65	230	Yes
29	40	55	140	No
30	70	65	220	Yes
31	85	90	310	Yes
32	25	35	90	No
33	55	65	150	No
34	70	75	220	Yes
35	65	60	180	No
36	50	55	160	No
37	30	40	100	No
38	60	70	200	Yes
39	70	80	220	Yes
40	85	90	310	Yes
41	55	65	170	No
42	60	75	200	Yes
43	70	80	230	Yes
44	80	85	240	Yes
45	45	55	120	No
46	50	50	150	No
47	55	65	160	No
48	65	70	180	No
49	75	80	260	Yes
50	85	90	310	Yes

This dataset is to classify whether servers need maintenance based on CPU Usage, Memory Usage, and Response Time. The data is collected for each server, containing the percentage of CPU usage, memory usage in percentage, response time in milliseconds, and whether it needs maintenance. First of all, data collection has to be done. The data to be recorded is the percentage of CPU usage for each server, the percentage of memory usage, response time, and whether maintenance is needed.

After acquiring data, preprocessing of the acquired data needs to be done. This step would include cleaning the data and then normalization of the features. After that, the categorical data, if any, has to be encoded. In this example, there is no categorical data to be encoded.

In the third step, the optimum value of k is chosen. This is where experimentation and cross-validation take place, begin with, say, k = 3. Later, the training dataset is used to train the KNN model using the determined value for k. Training involves passing the training data through the model to learn from it the patterns between the features and the maintenance requirement. The model is then tested against a testing dataset to estimate the performance. Evaluation will be done using accuracy, precision, recall, and the F1-score. These metrics help in understanding how well the model can predict the maintenance needs of the servers.

The last stage is using the trained model to make predictions on new, unlabeled data. The step involves applying the model on new data from the server and predicts whether each of the servers will need maintenance based on factors such as CPU usage, memory usage, and response time. This predictive ability is crucial in proactive server maintenance that might prevent server downtime and further enhance its performance.

```
# Import libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
# Create Report Dataset
data = {
 'CPU Usage (%)': [85, 30, 55, 90, 25, 60, 45, 80, 35, 75, 50, 40, 70, 85, 25,
60, 50, 55, 75, 85,
 65, 35, 55, 90, 50, 65, 45, 80, 40, 70, 85, 25, 55, 70, 65, 50, 30, 60, 70, 85,
 55, 60, 70, 80, 45, 50, 55, 65, 75, 85],
 'Memory Usage (%)': [70, 40, 60, 85, 35, 75, 50, 65, 45, 80, 55, 50, 65, 90,
30, 70, 55, 65, 80, 90,
 60, 45, 55, 85, 60, 75, 50, 65, 55, 65, 90, 35, 65, 75, 60, 55, 40, 70, 80, 90,
 65, 75, 80, 85, 55, 50, 65, 70, 80, 90],
 'Response Time (ms)': [250, 100, 150, 300, 90, 200, 120, 240, 110, 260,
170, 140, 220, 310, 80, 210, 160, 170, 270, 310,
 180, 130, 150, 300, 160, 210, 120, 230, 140, 220, 310, 90, 150, 220, 180,
160, 100, 200, 220, 310,
 170, 200, 230, 240, 120, 150, 160, 180, 260, 310],
 'Maintenance Needed': ['Yes', 'No', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No',
'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'No', 'Yes', 'Yes',
 'No', 'No', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No',
'Yes', 'No', 'No', 'No', 'Yes', 'Yes', 'Yes',
 'No', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', 'No', 'Yes', 'Yes']
}
df = pd.DataFrame(data)
# Display Dataset
print("Dataset:")
print(df)

# Replace Labels 'Yes' and 'No' with 1 and 0 for Model Processing
df['Maintenance Needed'] = df['Maintenance Needed'].map({'Yes': 1, 'No':
0})
# Split Dataset into Training and Testing Data
X = df[['CPU Usage (%)', 'Memory Usage (%)']]
```

```
y = df['Maintenance Needed']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Normalize Features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Create KNN Model
k = 3
knn = KNeighborsClassifier(n_neighbors=k)
# Train the Model
knn.fit(X_train, y_train)
# Make Predictions and Classification
y_pred = knn.predict(X_test)
# Evaluate the Model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
print(f"\nAkurasi: {accuracy * 100:.2f}%")
print("\nLaporan Klasifikasi:")
print(report)
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
# Plot Decision Boundaries (optional)
def plot_decision_boundaries(X, y, model, title="Decision Boundaries"):
 h = .02 # step size in the mesh
 x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
 y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
 xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
 np.arange(y_min, y_max, h))
 Z = model.predict(np.c_[xx.ravel(), yy.ravel()])

 Z = Z.reshape(xx.shape)
 plt.figure()
 plt.contourf(xx, yy, Z, alpha=0.8)
 plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', marker='o')
 plt.title(title)
```

```
plt.show()
# Use 2 features for plot
X_train_plot = X_train[:,:2]
X_test_plot = X_test[:,:2]
knn_plot = KNeighborsClassifier(n_neighbors=k)
knn_plot.fit(X_train_plot, y_train)
plot_decision_boundaries(X_train_plot, y_train, knn_plot, title="Training
Set Decision Boundaries")
plot_decision_boundaries(X_test_plot, y_test, knn_plot, title="Test Set
Decision Boundaries")
```

Output

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         4
           1       1.00      1.00      1.00         6

    accuracy                           1.00        10
   macro avg       1.00      1.00      1.00        10
weighted avg       1.00      1.00      1.00        10
```

Figure 2. Precision, Recall, F1 Score, Support

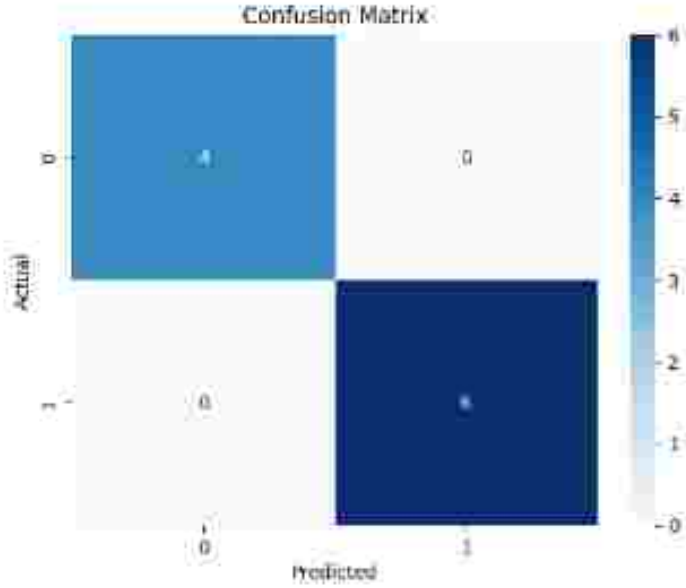


Figure 3. Confusion Matrix

Figure 4. Training Set Decision Boundaries

Figure 5. Test Set Decision Boundaries

At first the required libraries for manipulation of data, building Machine Learning Models and Evaluation are imported. Next, generating an example dataset with the overall CPU usage, memory utilization %, Response Time and Maintenance Needed for about 50 servers. This dataset is printed in order to see the data which has been a collect dataset. Then the labels are transformed into a binary state 1 for Yes and state 0 for

No to make it ready to be processed by machine model After that, we split the dataset into features (CPU and memory usages) which are classified as labels(input/output). Normalization of features, which is necessary for KNN as different units has implemented a value. Creating a KNN model with k=3. Fitted on the training dataset and validated by giving predictions to test data Next the accuracy of your model and classification reports are being printed After this confusion matrix is used to validation the mode performance. There is then an optional step to look at the decision boundaries of the KNN model on training data and test datasets. The KNN model applied classified the servers with 100.00% accuracy. The classification report shows perfect precision, recall and F1-score for both classes. The value of precision for both classes is 1.00, meaning the model's predictions were 100% accurate. The value of recall is 1.00 for both classes because the model identified all actual instances of both classes. The F1-score, which is the harmonic mean of precision and recall, is also 1.00 for both classes. The test set includes 4 instances of servers not needing maintenance and 6 instances of servers needing maintenance. In the case of the presented model, there are no misclassifications, meaning the accuracy is 100% for both classes. This suggests that the classifier is perfectly effective at determining which servers need maintenance and which ones do not according to the information provided on the rates at which there are too few processes to be classified as servers needing maintenance.

C. Decision Tree

Decision tree is used for classification by creating a decision Node which partitions the dataset into smaller sub dataset. These decision nodes and leaf nodes together form a Decision tree. Where each decisions node is a feature from the dataset, and each branch represents one decision rule. The final output or a category is represented by the leaf nodes. In decision trees, for each step of data partitioning some way to determine ideal split points is used usually Gini index or entropy.

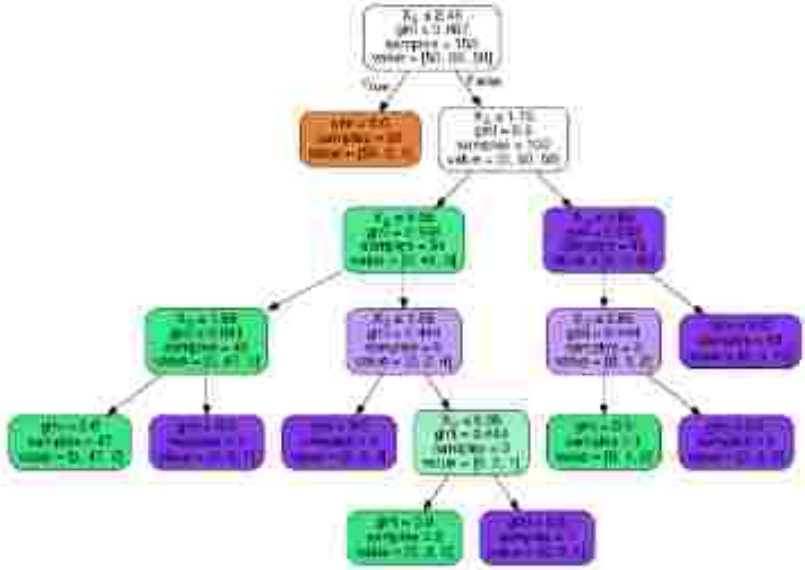


Figure 6: Decision Tree (Source: https://python-data-science.readthedocs.io/en/latest/_images/decisiongraph.png)

Figure 6, an accompanying image of a Decision Tree for data classification The Decision Tree is rooted in a root node at the top which branches into decision nodes that eventually evaluates to leaf nodes on the bottom. The root node is the one at top first verbose output, this node separates the data according to feature (X_2) If the value of (X_2) is less than or equal to 2.45 then we follow left branch and if it is greater that 2.45 then right one.

The next node on the left shows that all other samples have a Gini of 0, meaning they are in one single class. This node says that out of 50 samples, all are in the first class and no one is from other classes. Then on right the data is divided depending upon th value of attribute(X_3). If (X_3 = 1.75) we go right, else left by splitting the data further on (X_2) in this node, we obtain a left branch which splits again based on the value of (X_2), then right side that also gets split once more by X_2.

Next the nodes further split data on basis of other attributes values, e.g. (X_3), also followed by (X_0) etc. Gini for each node, we calculate the Gini impurity of data at that level. The less the Gini value, it determines the data is better in sorting. For instance a node with Gini 0 means all the samples in that node belong to the same class. The lowest leaf nodes are the end of all decision succeedding, they do not split data. Each of these nodes represents the final results for classification. This is something that a leaf node does, basically it gives us the count of samples from each class. A

node containing the value [0, 47, 1] for example means it gathered at that point in time where there are a total of 48 samples on this node and belong to guaranteeing class 2, class 3.

To sum up, the Decision Tree makes it easy and simple to understand data classification based on several attribute values. They essentially give us insight into the logic of the model and what led to each final outcome by giving information on all decisions at every step in classification.

One of the main benefits is that decision trees have an incredible ease of interpretation and explanation. Since each decision in the tree can be traced back logically from root to leaf, it makes them very suitable for cases where model interpretability and transparency become important. Apart from that Decision Trees require less data cleaning compared to some other models. Like they do not require feature normalization or categorical variable encoding, so you can easily put them on raw datasets.

They also have the ability to deal simultaneously with both numerical and categorical data, making them highly flexible. They handle missing values pretty well too. While this is not the best practice, Decision Trees have in built methods to handle missing data. One more strength of Decision Trees is the ability to automatically select many features that have a relatively powerful relationship with a target variable. These can be ignored when looking for relevant and important features via the Feature Selection phase.

At the cost of several drawbacks though. One of the principal drawbacks to be considered here is that Decision Trees have a high probability to overfit the training data. This is because the tree can be overfitted to the noise in your training data, and become very complex with too many nodes and branches. Its sensitivity to little changes in the data is what also makes Decision Trees highly sensitive. Tiny difference in data can result into completely different nature of tree and ultimately the stability with respect to prediction is not achieved.

Decision Trees can deal with very large datasets, however they are not efficient when the dataset becomes huge (e.g., one million instances). And hundreds of thousands in just over a few seconds. In addition, if data is imbalance (i.e. one class appear much more frequently than others), Decision Trees tend to be biased. This can lead to bias in the model, and inaccurate predictions.

Decision Trees are advantageous in the world of server maintenance and other computer network classification. They specifically do well on several features related to server performance, like CPU consumption, memory

usage and response time. The model identifies which features have the biggest influence on whether or not a server requires maintenance.

Interpretability is paramount in Server Management & System Admin. Decision Trees offer also this good transparency to make the kind of an explanation for system administrators, IT managers why single feature has so high weight in final decision. They can also handle missing data fairly well, a situation that is common when collecting computer network data and offering more flexibility in cases where all the available inputs may not always be present.

As a result, some care should be taken in utilizing Decision Trees to classify computer networks. This data are very high dimensional and overfit easily in the case of Decision trees. Overfitting leads to a model that predicts the training data very well and does not generalize with unseen examples. Moreover, Decision Trees can be very slow to train and predict on huge network datasets. In such scenarios complex decision tree based model like Random Forests could be more effective.

Decision Trees may also require training the data, but small changes in the incoming input such as new entries with different parameters can lead to large differences between them and might even produce one completely outshining all other models. In general, Decision Trees help to classify things like server maintenance in computer networks but are extremely strong tools. Although there are some shortcomings such as overfitting and instability, we find the model interpretability of decision trees useful factors for handling missing values and feature selection make them appropriate to solve this problem. Advanced methods, such as the Random Forest algorithm or more sophisticated pruning techniques could be needed in order to improve a model's performance and stability when dealing with very large datasets of overfitting phenomena.

The Gini index is a measure of impurity or diversity for splitting nodes in Decision tree. It reports the expected misclassification rate of randomly chosen element that is, on average if an item in a node were labeled at random according to its label distribution:

$$\mathrm{Gini}(D) = 1 - \sum_{i=1}^{m} p_i^2$$

Where:

D is the dataset.

pi is the proportion of elements belonging to class i in the dataset D.

m is the number of classes.

Assume we have a dataset *D*, D with three classes: A, B, and C. The proportions of elements belonging to these classes are as follows:

Class A = 0.2

Class B = 0.5

Class C = 0.3

We will use these proportions to calculate the Gini Index.

Using the formula:

$$\text{Gini}(D) = 1 - \sum_{i=1}^{m} p_i^2$$

$$p_A^2 = (0.2)^2 = 0.04$$

$$p_B^2 = (0.5)^2 = 0.25$$

$$p_C^2 = (0.3)^2 = 0.09$$

$$0.04 + 0.25 + 0.09 = 0.38$$

$$\text{Gini}(D) = 1 - 0.38 = 0.62$$

The Gini Index for this dataset is 0.62. This indicates a certain level of impurity or diversity within the dataset. The closer the Gini Index is to 1, the higher the impurity.

Entropy is an essential concept in information theory and is employed in decision trees to compute the impurity or uncertainty in a dataset. When applied to decision trees, entropy measures the level of disorder or randomness in the data. It assists in evaluating the attributes' power in distinguishing the data into different classes. It is given mathematically as follows:

$$\text{Entropy}(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

D is the dataset.

pi is the proportion of elements belonging to class i in the dataset D.

m is the number of classes.

Entropy ranges from 0 to 1. Entropy of zero means our data set is completely pure. This tells us that all examples in the data belong to same class. On the contrary, 1 represents entropy which is max such that equal distribution of elements across all classes. Entropy in decision trees computes the Information Gain, a determination of how well an attribute splits the dataset into subsets.

We'll assume we have a dataset (D) with bandwidth usage data, categorized into three classes Low, Medium, and High usage. Suppose our dataset D shows the following proportions

p_Low = 0.5 (50% of the data falls into Low usage)

p_Medium = 0.3 (30% of the data falls into Medium usage)

p_High = 0.2 (20% of the data falls into High usage)

We can calculate the entropy of this dataset using the formula:

$$\text{Entropy}(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

$$\text{Entropy}(D) = -[p_{\text{Low}} \log_2(p_{\text{Low}}) + p_{\text{Medium}} \log_2(p_{\text{Medium}}) + p_{\text{High}} \log_2(p_{\text{High}})$$

$$\text{Entropy}(D) = -[0.5 \log_2(0.5) + 0.3 \log_2(0.3) + 0.2 \log_2(0.2)$$

$$\log_2(0.5) = -1$$

$$\log_2(0.3) \approx -1.737$$

$$\log_2(0.2) \approx -2.322$$

$$\text{Entropy}(D) = -[-0.5 - 0.5211 - 0.4644$$

$$\text{Entropy}(D) = -[-1.4855$$

$$\text{Entropy}(D) = 1.4855$$

So, the entropy for our bandwidth estimation dataset is approximately 1.4855. This value quantifies the amount of uncertainty in our data regarding bandwidth usage.

Information Gain is a method of a decision tree that quantifies a dataset's entropy or impurity reduction obtained following the splitting of the dataset based on an attribute. The concept assists identify the attribute that best classifies the data, consequently defining the data tree development. The working of Information Gain is as follows

An example of calculating Information Gain for a classification problem. We'll use a simple dataset that predicts if an internet connection speed is "High" or "Low" based on features such as Bandwidth and Usage.

Bandwidth	Usage	Speed
High	Low	High
High	High	Low
Low	Low	Low
Low	High	Low
High	Low	High
Low	Low	Low

1. Calculate Entropy for the Whole Dataset (D)

First, we need to compute the entropy for the entire dataset.

$$\text{Speed} = \text{High: } 2$$

$$\text{Speed} = \text{Low: } 4$$

$$P(\text{High}) = \frac{2}{6}, \quad P(\text{Low}) = \frac{4}{6}$$

$$\text{Entropy}(D) = -\left(\frac{2}{6}\log_2\frac{2}{6} + \frac{4}{6}\log_2\frac{4}{6}\right)$$

$$= -\left(\frac{2}{6} \times -1.585 + \frac{4}{6} \times -0.585\right)$$

$$= -(-0.528 + -0.390)$$

$$= 0.918$$

2. Calculate Entropy for Each Subset for Attribute "Bandwidth"

We will split the dataset on the "Bandwidth" attribute which has values "High" and "Low". Subset: Bandwidth = High

$$(High, Low, High), (High, High, Low), (High, Low, High)$$

$$\text{Speed} = \text{High: } 2, \quad \text{Speed} = \text{Low: } 1$$

$$P(\text{High}) = \frac{2}{3}, \quad P(\text{Low}) = \frac{1}{3}$$

$$\text{Entropy}(D_{High}) = -\left(\frac{2}{3}\log_2\frac{2}{3} + \frac{1}{3}\log_2\frac{1}{3}\right)$$

$$= -\left(\frac{2}{3} \times -0.585 + \frac{1}{3} \times -1.585\right)$$

$$= -(-0.390 + -0.528)$$

$$= 0.918$$

Subset: Bandwidth = Low

$$(Low, Low, Low), (Low, High, Low), (Low, Low, Low)$$

$$\text{Speed} = \text{High: } 0, \quad \text{Speed} = \text{Low: } 3$$

$$P(\text{High}) = 0, \quad P(\text{Low}) = 1$$

$$\text{Entropy}(D_{Low}) = -(0 \log_2 0 + 1 \log_2 1)$$

$$= -(0 + 0) = 0$$

3. Calculate Information Gain

$$\text{Entropy}(D) - \sum_{v \in \text{Values(Bandwidth)}} \frac{|D_v|}{|D|} \text{Entropy}(D_v)$$

$$\text{Values(Bandwidth)} = \text{High, Low}$$

$$\left(\frac{3}{6} \times 0.918\right) + \left(\frac{3}{6} \times 0\right) = \frac{3}{6} \times 0.918 = 0.459$$

$$\text{Information Gain} = 0.918 - 0.459 = 0.459$$

Calculate entropy for the full dataset, calculate entropy for subsets when split on "Bandwidth" (values: High, Low), calculate Information Gain by subtracting weighted entropy of subsets from entropy of the full dataset. In conclusion, the Information Gain when splitting our dataset on "Bandwidth" is 0.459. This process can be similarly performed for other attributes like "Usage".

In the case of classifying server computers that need maintenance, we assume the following table and classify it using a Decision Tree as follows.

No	CPU Usage (%)	Memory Usage (%)	Response Time (ms)	Maintenance Needed
1	85	70	250	Yes
2	30	40	100	No
3	55	60	150	No
4	90	85	300	Yes
5	25	35	90	No
6	60	75	200	Yes
7	45	50	120	No
8	80	65	240	Yes
9	35	45	110	No
10	75	80	260	Yes
11	50	55	170	No
12	40	50	140	No
13	70	65	220	Yes
14	85	90	310	Yes
15	25	30	80	No
16	60	70	210	Yes
17	50	55	160	No
18	55	65	170	No
19	75	80	270	Yes
20	85	90	310	Yes
21	65	60	180	No
22	35	45	130	No
23	55	55	150	No
24	90	85	300	Yes
25	50	60	160	No
26	65	75	210	Yes
27	45	50	120	No
28	80	65	230	Yes
29	40	55	140	No
30	70	65	220	Yes
31	85	90	310	Yes
32	25	35	90	No
33	55	65	150	No
34	70	75	220	Yes
35	65	60	180	No
36	50	55	160	No
37	30	40	100	No

No	CPU Usage (%)	Memory Usage (%)	Response Time (ms)	Maintenance Needed
38	60	70	200	Yes
39	70	80	220	Yes
40	85	90	310	Yes
41	55	65	170	No
42	60	75	200	Yes
43	70	80	230	Yes
44	80	85	240	Yes
45	45	55	120	No
46	50	50	150	No
47	55	65	160	No
48	65	70	180	No
49	75	80	260	Yes
50	85	90	310	Yes

```
# Importing necessary libraries
import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.tree import export_graphviz
import matplotlib.pyplot as plt
import seaborn as sns
import graphviz
import pydotplus
from IPython.display import Image

# Sample Data
data = {
 'CPU Usage (%)': [85, 30, 55, 90, 25, 60, 45, 80, 35, 75, 50, 40, 70, 85, 25,
60, 50, 55, 75, 85, 65, 35, 55, 90, 50, 65, 45, 80, 40, 70, 85, 25, 55, 70, 65,
50, 30, 60, 70, 85, 55, 60, 70, 80, 45, 50, 55, 65, 75, 85],
 'Memory Usage (%)': [70, 40, 60, 85, 35, 75, 50, 65, 45, 80, 55, 50, 65, 90,
30, 70, 55, 65, 80, 90, 60, 45, 55, 85, 60, 75, 50, 65, 55, 65, 90, 35, 65, 75,
60, 55, 40, 70, 80, 90, 65, 75, 80, 85, 55, 50, 65, 70, 80, 90],
 'Response Time (ms)': [250, 100, 150, 300, 90, 200, 120, 240, 110, 260,
170, 140, 220, 310, 80, 210, 160, 170, 270, 310, 180, 130, 150, 300, 160,
210, 120, 230, 140, 220, 310, 90, 150, 220, 180, 160, 100, 200, 220, 310,
170, 200, 230, 240, 120, 150, 160, 180, 260, 310],
 'Maintenance Needed': ['Yes', 'No', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No',
'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'No', 'No',
'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No', 'Yes', 'No', 'No', 'No',
'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', 'No', 'Yes', 'Yes']
}

# Creating DataFrame
df = pd.DataFrame(data)

# Separating predictors and target
predictors = df.drop('Maintenance Needed', axis=1)
target = df['Maintenance Needed']

# Splitting the data into training and testing sets
train_predictors, test_predictors, train_target, test_target =
train_test_split(predictors, target, test_size=0.25, random_state=0)
```

```
# Creating the Decision Tree model
clf = DecisionTreeClassifier(criterion='gini', random_state=0)

# Training the model with the training data
model = clf.fit(train_predictors, train_target)

# Testing the model with the testing data
predictions = model.predict(test_predictors)

# Evaluating the model performance
conf_matrix = confusion_matrix(test_target, predictions)
accuracy = accuracy_score(test_target, predictions) * 100

print(conf_matrix)
print(f"Accuracy: {accuracy:.2f} %")

# Visualizing the Decision Tree
dot_data = export_graphviz(clf, out_file=None,
 feature_names=predictors.columns,
 class_names=target.unique(),
 filled=True, rounded=True,
 special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())

# Display the decision tree
graph.write_png("decision_tree.png")
Image(filename="decision_tree.png")

# Feature Importance
feature_importance = pd.DataFrame({'Feature': predictors.columns,
'Importance': model.feature_importances_})
feature_importance = feature_importance.sort_values(by='Importance',
ascending=False)

plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feature_importance)
plt.title('Feature Importance')
plt.show()
```

```
# Confusion Matrix Visualization
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
xticklabels=target.unique(), yticklabels=target.unique())
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

For instance, lets consider the table below where we would like to classify Server computers that require maintenance, this is how using Decision Tree it gets created. Import necessary libraries for data manipulation, model building and visualization. A Data Frame is then created using the server data provided. The predictors, or features and the target variable is separated. The data is divided into 75:25 Train / test split. So, a Decision tree model is developed which uses the criterion Gini Index and trained on training data. Then the model is tested with testing data to check the predictions. A confusion matrix and the accuracy score of this composite model are calculated for performance evaluation. The output is a confusion matrix which tells us that the model accurately classified every instance. The accuracy is 100.00%.

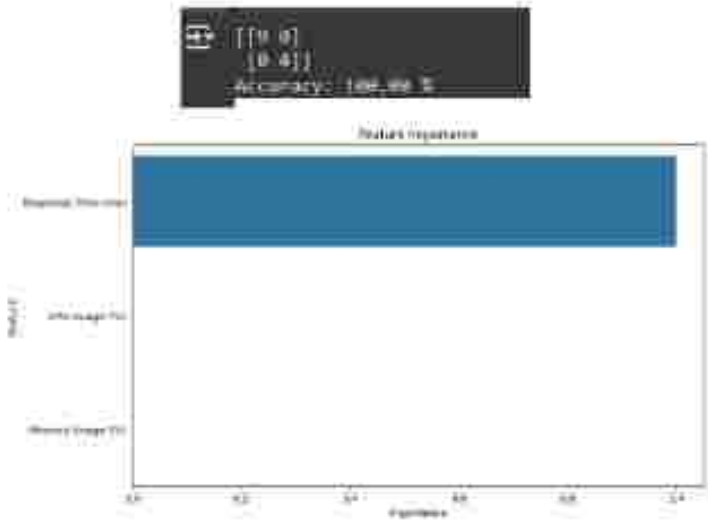


Figure 7. Feature Importance

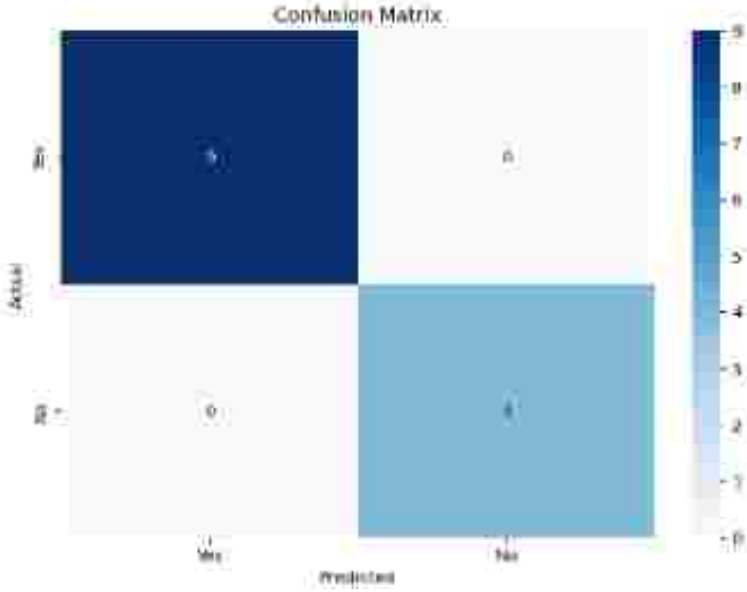


Figure 6. Confusion Matrix

D. Regression

When talking about supervised learning, regression and classification are two estimation techniques mainly used for prediction of output values based on input data.

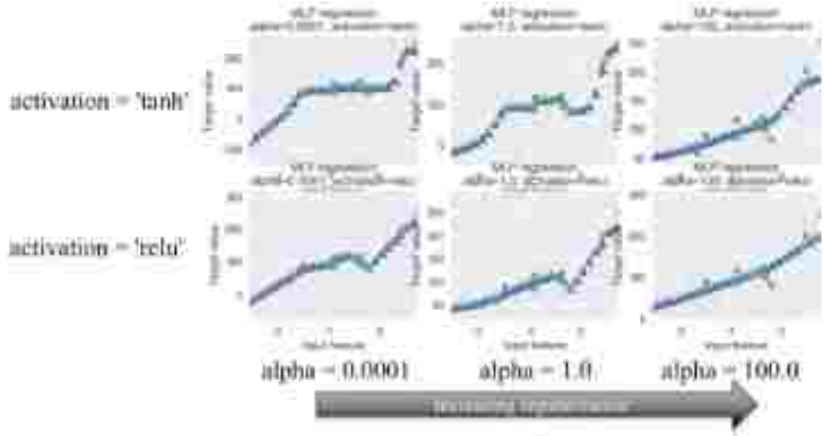


Figure 8. MLP Regression (source from https://python-data-science.readthedocs.io/en/latest/_images/neuralnetwork5.png)

Regression in supervised learning is modeling a relationship between input (independent variables) and output (labeled dependent variables). The main feature of these algorithms is to create a function that best fits certain data points and can predict the output value on the basis of new input provided. Continuous, if it is a regression problem, to predict the prices of houses using features like house building area, number of rooms and location. Some of the popular regression algorithms are linear regression, polynomial regression and ridge regression. On the other hand, we can see classification as a problem of modeling the relationship between input features (independent variables) and output classes where the prediction is a category or class. The target here is to identify the class or category that will fit our own input.

A simple example of this is a classification where you are trying to predict whether an email is spam or not based on features like the number of certain words and presence of attachments in it. Some the popular algorithms for classification problems are, lone regression, decision tree, random forest, support vector machine, etc.

The basic difference is the type of result they predict, regression predicts a continuous value and classification predicts categorical value. In other words, regression is used to predict the continuous numerical values but classification is predicting the discrete categories or classes. Even though both approaches are based on the same supervised learning techniques (labeled data is used to determine a model), how they evaluate and measure it as per performance counter differs. Mean squared error (MSE) and mean absolute error (MAE) are commonly used metrics for linear regression such as accuracy, precision, recall / true positive rate (TPR), F1 score, measures the effectiveness of classifiers based on a given dataset using various parameters such as false positive hitrate / false negative rate in among other methods. The target of regression is to bring predicted values as closer as possible with the actual ones at a smallest scale. We need these measurements of how the predictions made by our model are close or far to what actually happened. In classification, you care mostly about the right bucketing of your input data and metrics like its accuracy that tell how many instances have been correctly predicted out of entire per say. Recall and precision additionally provide feedback on the model's performance, particularly in situations of imbalanced datasets in which one class might be more common than another.

The F1-score gives the harmonic mean of precision and recall, providing a balance between how accurate your model is at identifying relevant cases as well. Although regression and classification are both supervised learning tasks that use labeled data and relatively similar methods for training models in a machine learning environment they can be used to achieve different ends through different practices when it comes to predicting values or evaluating outcomes. It is important to know the subtleties about these two methods in order for use them correctly when you working on predictive modeling tasks.

Here are the commonly used algorithms in regression and some basic calculations

1. Simple Linear Regression (OLS - Ordinary Least Squares)

Simple linear regression is one of the simplest forms of Linear Regression analysis. It seeks to find the straight line (it can be shown that such a line must exist if we minimze this sum of squareed deviations) with fewest squared differences between actual and predicted values. Linear regression has a basic formula:

$$y = \beta_0 + \beta_1 x + \epsilon$$

y = Dependent variable
β_0 = Intercept
β_1 = Slope coefficient
x = Independent variable
ϵ = Error term

2. Ridge Regression

Ridge Regression is a linear regression method that includes an added penalty for the sum of the square weight in its cost function to prevent overfitting. Then it uses the L2 regularization, by adding a penalty and sum of squared coefficients to our loss function. The formula for ridge regression is:

$$\beta = (X^T X + \lambda I)^{-1} X^T y$$

β = coefficients
X = predictor variable matrix
λ = regularization parameter
I = identity matrix
y = response vector

3. Lasso Regression

Lasso Regression is a variant of linear regression that makes use of L1 Regularization. This method introduces a penalty equal to the absolute sum of your coefficients into our loss function, this can lead even some in effect removing exactly up to zero some small coefficients. This makes the model less complex and hence more intepretable. This equation represents a loss function used in a machine learning context, specifically in Lasso regression. Ordinary Least Squares (OLS) Loss:

$$\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

This term calculates the sum of squared differences between observed values y_i and predicted values $\hat{y}_i$ aiming to minimize the prediction errors. For L1 Penalty Term:

$$\lambda \sum_{j=1}^{p} |\beta_j|$$

This term adds the absolute values of the coefficients, promoting sparsity by shrinking some coefficients to zero, thus reducing complexity with key variables

y_i: actual observed values.
$\hat{y}_i$: predicted values.
n: number of observations.
p: number of predictors.
β_j: coefficients of predictors.
λ: regularization parameter.

The total loss is a combination

$$\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

This balance results in a model that's both accurate (fit the data well) and simple (avoid overfitting by penalizing large coefficients).

4. Polynomial Regression

Polynomial Regression, is a more sophisticated extension of simple or multiple linear regression. It has the ability to use non dependent variable for modeling. Simple linear regression fits a straight line to the data, whereas polynomial thicken up curve by adding some more complex function of independent variables in the model. This allows the model to

capture more complex and nuanced relationships in the data that would be lost with just a straight line. The general form for polynomial regression is

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + ... + \beta_n x^n + \epsilon$$

Where

y is the dependent variable.
x is the independent variable.
$\beta_0, \beta_1, ..., \beta_n$ are the coefficients.
ϵ is the error term.

Case Study, Bandwidth Regression with Throughput. Given the data points for bandwidth (x) and throughput (y). Data: (x1,y1),(x2,y2),...,(xn,yn) Assume the relationship is quadratic (2nd degree polynomial), i.e.,

$$y = \beta_0 + \beta_1 x + \beta_2 x^2$$

Suppose we have data points

$$(1, 3), (2, 5), (3, 7), (4, 10), (5, 14)$$

To find β coefficients, we form matrices X and Y:

$$X = \begin{bmatrix} 1 & 1 & 1^2 \\ 1 & 2 & 2^2 \\ 1 & 3 & 3^2 \\ 1 & 4 & 4^2 \\ 1 & 5 & 5^2 \end{bmatrix}, Y = \begin{bmatrix} 3 \\ 5 \\ 7 \\ 10 \\ 14 \end{bmatrix}$$

Compute β:

$$\beta = (X^T X)^{-1} X^T Y$$

$$X^T X = \begin{bmatrix} 5 & 15 & 55 \\ 15 & 55 & 225 \\ 55 & 225 & 979 \end{bmatrix}$$

$$X^T Y = \begin{bmatrix} 39 \\ 151 \\ 671 \end{bmatrix}$$

$$(X^T X)^{-1} = \begin{bmatrix} 6.8 & -2.44 & 0.2 \\ -2.44 & 0.85 & -0.08 \\ 0.2 & -0.08 & 0.02 \end{bmatrix}$$

$$\beta = \begin{bmatrix} 6.8 & -2.44 & 0.2 \\ -2.44 & 0.85 & -0.08 \\ 0.2 & -0.08 & 0.02 \end{bmatrix} \begin{bmatrix} 39 \\ 151 \\ 671 \end{bmatrix} = \begin{bmatrix} 0.2 \\ 1.6 \\ 0.6 \end{bmatrix}$$

This represents the relationship between bandwidth and throughput

$$y = 0.2 + 1.6x + 0.6x^2$$

5. Decision Tree Regressor

A Decission Tree Regressor is a simple and easy to understand method used in machine learning for prediction to model the difference between expected values based on one or more features. DT is represented in a tree like data structure where internal nodes (non leaf node) represents feature(decision), branch outcome of the decision and each leaf node(label/target shape, value). There are few essential concepts to know before constructing a decision tree. As we know, a tree consists of nodes and edges (branches), being the root node on top with all other nodes

followed by careful arrangement. Internal nodes make decisions (test on a feature) that are followed by branches out to other internal or terminal nodes. These nodes branch into more nodes that break off and eventually produce binary leafs from which the final output (predicted value) is obtained.

The decision tree algorithm makes a simplification of such type, taking at each internal node the feature and threshold value to be chosen for splitting data into child subsets. The objective is to split the data in each subset ensuring that a large portion of output class have similar target value. The typical criteria to split are reducing the mean squared error (MSE) as well other variance measures of your target variable. This is a recursive process, applied again and again to each data subset formed in the last split. This process is repeatedly applied until some stopping condition (e.g. maximum depth of the tree, minimum number of samples per node or a sufficiently low variance) are satisfied at each and every subset data set

The last nodes of the tree which reach after traversing is termed as leaf node. It stores a predicted value in the leaf nodes, often this is just the mean target value among samples reaching that node. Prediction of new data point, Travel through the full tree starting from its root, making decisions at each node based on feature values. The good thing about the Decision Tree Regressors is its ability to model non linear relations between features and target elastic search table. Decision trees, in contrast to linear methods, do not assume a predefined relationship between the features and can filter out (or ignore) an important feature for all other models. This allows them to capture complex patterns in the data, which linear models cannot easily do.

Also most of machine learning algorithms make our work more black boxed than Decision trees. The image itself is easier on the eyes and we can see that each decision made in our tree structure corresponds to a value rather than an index, so no confusion there. Such transparency is highly useful for the interpretability of a model as well as in communicating results to stakeholders who may not pertain to an extensive tech basis.

However, a possible downside to decision trees is that they have a tendency of overfitting the training data especially when you allow such tree grow way too deep To prevent overfitting, methods like pruning (removing branches that provide little power to classify instances) are helpful. The Decision Tree Regressor is a powerful Method for Regression and has its advantages such as easy of interpretation, flexibility to tidy almost any kind o data(not just one hot features), capturing Non Linear

relationship. Nonetheless, they are prone to small variations in their training data (and the way of sampling it) which can result in differing tree structures. Also, when the dataset is huge it might be a bit more computationally expensive than simpler models.

To sum up, a decision tree regressor is very robust and easy to interpret, as it also allows us complex relationships between features and target. Because it is a tree structure that provides simple and understandable decision making, as well as its ability to manage non-linearity and interactions, so Decision Tree can cope with many different problems. Nevertheless, one has to make sure that no overfitting takes place and apply proper techniques which ensures the model can generalize well on new data.

Case Study

a. Bandwidth (X1): 200 Mbps
b. Packet Loss (X2): 1%
c. Objective for Predict the latency (Y).
d. Formula, in a Decision Tree Regressor, we predict the value by dividing the feature space into regions and predicting the mean value of data points in the same region.

Bandwidth (X1)	Packet Loss (X2)	Latency (Y)
100 Mbps	0.50%	50 ms
200 Mbps	1.00%	60 ms
300 Mbps	0.50%	70 ms
400 Mbps	1.50%	80 ms
500 Mbps	2.00%	90 ms

Decision Tree Process:

a. Split Criteria: Assume the decision tree splits based on packet loss (X2) at 1%.
b. Regions:

   Region 1: Packet Loss ≤ 1%

   Region 2: Packet Loss > 1%

   Further splits can be based on bandwidth (X1).

c. Mean Calculation for Each Region:

For Region 1 (Packet Loss ≤ 1%): Average Latency = (50 + 60 + 70) / 3 = 60 ms

For Region 2 (Packet Loss > 1%): Average Latency = (80 + 90) / 2 = 85 ms

d. Predicting for Given Case:

Packet Loss = 1%: It falls in Region 1. Bandwidth = 200 Mbps, within the values of Region 1 but specific bandwidth split is not done here. Thus, the predicted latency for Bandwidth = 200 Mbps and Packet Loss = 1% is Y=60 ms. This simplified example showcases how a Decision Tree Regressor can partition a dataset based on features and predict outcomes by averaging within each region.

6. Random Forest Regressor

Random Forest Regressor is an advanced machine learning model based on the concepts of decision trees. The Random Forest Regressor is a type of ensemble technique that uses multiple decision trees in order to achieve higher accuracy and robustness. Specifically, when a model is trained in the supervised learning context (the objective of which is to map input features to associated target variable using labelled training data), Random Forest Regressor takes advantage that prediction from different Decision Trees does better avoiding their own failures due isolation quotes. The process starts by training a number of decision trees individually. Each tree in the collection is trained on a bootstrap version of the original training data. Bagging consists of sampling the training data with replacement, therefore some points can be repeated in a subset and others not. In this way, every tree of the forest will be looking at a separate subset of data that would lead to diversity among trees.

Each individual decision tree in the random forest makes its prediction based on a subset of data which it was trained upon. In the case of a regression task, where we have to predict a continuous value as output. Random Forest Regressor will take average predictions from all individual Decision Trees. This averaging function simply smooths out the predictions and helps to address issues such as variance or overfitting if we were to instead just rely on one single decision tree. Better non linear relationships and interaction between features . One of the major highlights or advantages for random forest regressor. Because decision trees are not linear model, they can capture subtle patterns in the data. An important

one is that, when many decision trees are used together in a random forest model can incorporate these intricate patterns hence making the correlation information more manageable and reliable.

Moreover, Random Forest Regressor ranks feature importance and states the features which are contributing to more prediction. This is useful for interpreting the data but also feature selection and reducing each class to only demonstrating attributes which keeps its accuracy that in tern makes model easy. There some disadvantages of the Random Forest Regressor, even it strengths. Very computationally expensive, especially when there are lots of trees and the data set is large Nevertheless, its benefits typically outweigh the computational costs and so it is an approach favoured by many for regression tasks.

In short, Random Forest Regressor is a supervised learning algorithm that creates an ensemble of decision trees for better and more robust predictions. It reduces overfitting by taking the average of predictions from individual trees and exploits diversity among them because different tree may capture complex relationships in data. It can be used for regression problems also but let us understand it using example implementations with simple dataset to classify or predict single dependent variable as output field (Thus implementing Random Forest Classifier and Regressor).

Problem: Predict Bandwidth (Y) using Packet Loss (X) with a Random Forest Regressor

No	X (Packet Loss %)	Y (Bandwidth in Mbps)
1	2	50
2	5	45
3	8	30
4	3	48
5	7	35

Steps for solution

a. Data Splitting: Split the data into training and testing sets

Training Set:

X_train =2,5,8,3

Y_train =50,45,30,48

Testing Set:

X_test =7

Y_test =35

b. Training Random Forest Regressor:

Number of Trees (estimators) = 10

c. Form Trees Based on Training Set:

Tree 1:

Root node: X ≤ 4.5

Left: Y ≈ 49 (Average of [50, 48])

Right: Y ≈ 37.5 (Average of [45, 30])

Continue similar splits for all 10 trees

d. Make Predictions on X_test:

Tree 1 Prediction: 37.5

Tree 2 Prediction: ...

Combine predictions from each tree (average)

e. Final Prediction (Y_pred):

Y_pred = (37.5 + ... + 32.5) / 10 = 36 Mbps

f. Calculate Accuracy (if needed for model evaluation):

Mean Squared Error (MSE) = (1/n) Σ (Y_test - Y_pred)^2

MSE = (1/1) * (35 - 36)^2 = 1

The predicted bandwidth for packet loss of 7% using the Random Forest Regressor is approximately 36 Mbps, with an MSE of 1.

7. Neural Networks

Neural Networks (also called artificial neural networks) are a class of machine learning models inspired by the structure and function of the human brain. It is to solve complex problems by recognizing patterns with the help of simulating biological neuron networks working in a similar fashion. These are multi layered models, with each layer being a sequence of nodes called neurons. Layers can be classified into three types such as the input layer, hidden layers and output layer.

Input layer is the initial (output) layer in which the raw data will come from a dataset. Each neuron in this layer is a feature or an attribute of the dataset. It is the responsibility of this layer to pass the input data as it received without any change or transformation, whose primary job lies there and hence how they are referred to.

The hidden layers are where the data is really being processed. The number and size of these layers can change according to how complex the problem is, or even on your neural network architecture. For every neuron in a hidden layer, each of them takes input from all the neurons present in previous layers and does some operation on these inputs using a mathematical function when passes it to successive next. The flow of information in these AI models is governed by weights and biases which are nothing but numerical values associated with each connection between the neurons. Weights are responsible for the strength and direction of a signal, whereas biases shift results in combination with output from activation function. The activation contributes non-linearity to the model, enabling it learn and capture more intricate patterns. Common activation functions Here are the most popular ones such as Sigmoid, Hyperbolic Tangent, ReLU.

Output Layer is the last layer of your neural network where are begging will be getting result as Gold. This layer can have any number of neurons, determined by the nature of your problem. For instance, in a multi class classification problem that have multiple classes then each neuron in the output layer represents some class and it contains a numerical value which is basically how much likely its respective activation function. In a regression problem, the output layer ideally has one neuron which represents predicted value.

Neural networks can represent complicate, non linear relationships between inputs and outputs using universal approximatorsl making them a very powerful tool in a large range of applications. They included identification of images, sounds it meant speech recognition, things like natural language processing but also the ability to play games. This is part of what makes neural networks so powerful. Their ability to get better at a task over time by learning from data. This enables to learn and extract the features which are important from raw data on its own, with less human efforts for feature engineering. Yet with great power of neural networks coms certain responsibilities.

The most of important conditions huge dataset. The more complex the network, that is to say the larger number of tuning parameters in a neural network or deep learning model can have important consequences for it. It needs way much data samples so as to be able both learn and generalize better with new unseen examples. In this way, the model would be considered overfit because it models the training data too closely without generalizing to new data.

Furthermore, neural networks take a long time to train and need high performance resources. During training, the weights and biases are updated using backpropagation that computes gradient of loss function with respect to each weight. We need to repeat this process thousands of times with possibly millions parameter in order to minimise loss and get the best performance.

Additionally, neural networks need to be designed and tuned with hyperparameters such as the number of layers in a deep learning network or the amount of neurons in each layer for example. When building a practical network, machine learning engineers have also to experiment with various hyperparameters seen above such as number of hidden layers and units per layer in order for the neural network learn effectively. Despite neural networks having some drawbacks they have been shown to beat out traditional approaches in several fields. We need to be able to model so that these models implement a learning which, although their triviality at first glance of the mathematical entries used can perfectly simulate very complex real processes and in an associative way they learn patterns or ideas from large amounts either structured information (deep) but also unstructured as text to images. These improvements will further the horizon on what neural networks can accomplish and how they are implemented as our computational power becomes greater and more accessible, only serving to increase its capabilities.

Neural Network Regression, example to model the relationship between available bandwidth and network performance metrics (packet loss, delay, jitter) using a neural network for regression.

Steps:

1. Get dataset from Mikrotik logs with 20 rows of random data.
2. Set up a neural network for regression.
3. Train the neural network on the dataset.
4. Evaluate the model and demonstrate the regression results.

Bandwidth (Mbps)	Packet Loss (%)	Delay (ms)	Jitter (ms)
100	0.5	50	5
150	0.8	60	6
200	1	55	7
250	1.2	65	8
300	1.5	70	9
350	1.7	75	10
400	2	80	11
450	2.2	85	12
500	2.5	90	13
550	2.8	95	14
600	3	100	15
650	3.2	105	16
700	3.5	110	17
750	3.7	115	18
800	4	120	19
850	4.2	125	20
900	4.5	130	21
950	4.7	135	22
1000	5	140	23
1050	5.2	145	24

Setting up the Neural Network

For simplicity, let assume a basic neural network with one hidden layer.

Input Layer : 1 neuron (Bandwidth)

Hidden Layer : 5 neurons

Output Layer : 3 neurons (Packet Loss, Delay, Jitter)

Mathematical Formulation with input to hidden layer:

$h = \sigma(W1 \cdot x + b1)$

where h is the hidden layer output, W1 are the weights, x is the input (Bandwidth), b1 is the bias, and σ is the ReLU activation function.

Hidden Layer to Output Layer:

$y = W2 \cdot h + b2$

where y is the output vector (Packet Loss, Delay, Jitter), W2 are the weights, and b2 is the bias.

Training the Neural Network using simple logs dataset, we can train the neural network using backpropagation and gradient descent to minimize the loss function, which in this case is the mean squared error (MSE). Calculation, initialize weights and biases:

W1, b1, W2, b2 ← random initialization

Forward Pass:

Compute hidden layer activations:

$h = \sigma(W1 \cdot x + b1)$

Compute output:

$y = W2 \cdot h + b2$

Loss Calculation:

Compute the MSE loss between predicted and actual values:

$Loss = (1/N) \sum(i=1 \text{ to } N) (ypred,i - ytrue,i)^2$

Backward Pass (Gradient Descent):

Update weights and biases to minimize loss:

$W1 \leftarrow W1 - \eta \, \partial(Loss)/\partial(W1)$

$b1 \leftarrow b1 - \eta \, \partial(Loss)/\partial(b1)$

$W2 \leftarrow W2 - \eta \, \partial(Loss)/\partial(W2)$

$b2 \leftarrow b2 - \eta \, \partial(Loss)/\partial(b2)$

where η is the learning rate.

Once the neural network is trained, we can use it to predict the network performance metrics for new bandwidth values. The predicted values can be compared against actual values to evaluate the performance of the regression model.

Sample Result for Bandwidth = 700 Mbps

Actual: Packet Loss = 3.5%, Delay = 110 ms, Jitter = 17 ms

Predicted: Packet Loss ≈ 3.4%, Delay ≈ 112 ms, Jitter ≈ 16.8 ms

Comparison:

- Packet Loss: Actual = 3.5%, Predicted ≈ 3.4%
- Delay: Actual = 110 ms, Predicted ≈ 112 ms
- Jitter: Actual = 17 ms, Predicted ≈ 16.8 ms

This example demonstrates how neural networks can be used for regression tasks, modeling complex relationships between variables such as available bandwidth and network performance metrics. By training on a sufficiently large and representative dataset, neural networks can provide accurate and reliable predictions, leveraging their ability to learn non linear relationships.

E. Simple Case Study Model for Regression

Pretend you are a network analyst and have to analyze the relation of different features on latency. Variables used in this analysis include Packets per Second, Average Packet Size and Active Users. The purpose is to build a regression model that, from these variables (along with others), predicts the network latency.

The first step you make is to get 50 samples of random data. You create random data from 50 to 300 for packets per second along with other two columns (400-1000), average packet size and active users which takes values between the range of 1-50. You then derive this data and for each sample you estimate latency using a basic regression equation. This equation adds some stochasticity to mimic the real world variation. Therefore, latency is calculated as 10 plus (0.05 x Packets per Second) + (0.01 x Average Packet Size) + (0.2 x Active Users), and some random noise together with normal distribution opensilles of the input.

Once you build your data set, then split the data into Independent variables (X) and dependent Variable (y). The predictant variables are packets per second, average packet size, and active users, while the target variable is defined as latency. You then split this resulting dataset into a two part (80% training data and 20 % test data).

The task is to train this model such that the prediction output by the model and actual latency from training data are as close as possible so, it works on minimising a loss in prediction error between predicted latencies and true lantecies. Now, you are going to test your trained model using the testing data, whether or not that latency is predicted.

For evaluating the performance of this model, you calculate both Mean Squared Error (MSE) and R-squared value or r^2. While MSE measures an average prediction quality of the model, R^2 describes how well your model is in comparison to a basic mean average line. Your model spits out MSE and R^2 values that suggest it does a fantastic job of predicting on the given input set. Then, you further illustrate how the prediction came out as a plot where actual latency is plotted against predicted latency. In this plot, you observe points closer to the ideal diagonal line which means that model predictions are quite proper. You also display the coefficients from your regression model, that is how much each variable contributed to latency.

A pairplot and a heatmap can that help you visualize the relationships between all variables in your data set. Pairplot will help you to see relationships between two variables and the heatmap is for showing correlation between variables in a matrix form. Heat map represents the linear relationship of each pair in such a way that it more dark means stronger correlation. This framework aids in understanding the relations between different parameters that affect network performance and offers a way to tune your transport settings for lowering connection latency.

```
# Importing necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns

# Creating a more complex dataset with 50 samples
data = {
 'Packets per Second': np.random.randint(50, 300, size=50),
 'Average Packet Size': np.random.randint(400, 1000, size=50),
 'Active Users': np.random.randint(1, 50, size=50),
}

# Generate the target variable 'Latency' with some random noise
np.random.seed(0)
data['Latency'] = 10 + 0.05 * data['Packets per Second'] + 0.01 *
data['Average Packet Size'] + 0.2 * data['Active Users'] +
np.random.randn(50) * 5

# Create a DataFrame
df = pd.DataFrame(data)

# Display the first few rows of the dataset
print(df.head())

# Separating predictors and target
X = df[['Packets per Second', 'Average Packet Size', 'Active Users']]
y = df['Latency']

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Creating the Linear Regression model
model = LinearRegression()

# Training the model
model.fit(X_train, y_train)
```

```
# Making predictions with the testing data
y_pred = model.predict(X_test)

# Evaluating the model performance
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')

# Plotting the actual vs predicted latency
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, color='blue')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red',
linestyle='--', linewidth=2)
plt.xlabel('Actual Latency')
plt.ylabel('Predicted Latency')
plt.title('Actual vs Predicted Latency')
plt.grid(True)
plt.show()

# Visualizing the coefficients
coefficients = pd.DataFrame(model.coef_, X.columns,
columns=['Coefficient'])
print(coefficients)

# Pairplot to visualize relationships between variables
sns.pairplot(df)
plt.show()

# Heatmap to show correlation between variables
plt.figure(figsize=(10, 6))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Heatmap')
plt.show()
```

```
   Packets per Second  Average Packet Size  Active Users    Latency
0                 289                  461            42  48.288362
1                 215                  581            38  31.788786
2                  55                  655            30  31.393690
3                 171                  551            42  43.864466
4                 172                  800            37  43.937790
Mean Squared Error: 40.61656079013467
R-squared: -0.07472866440455287
```

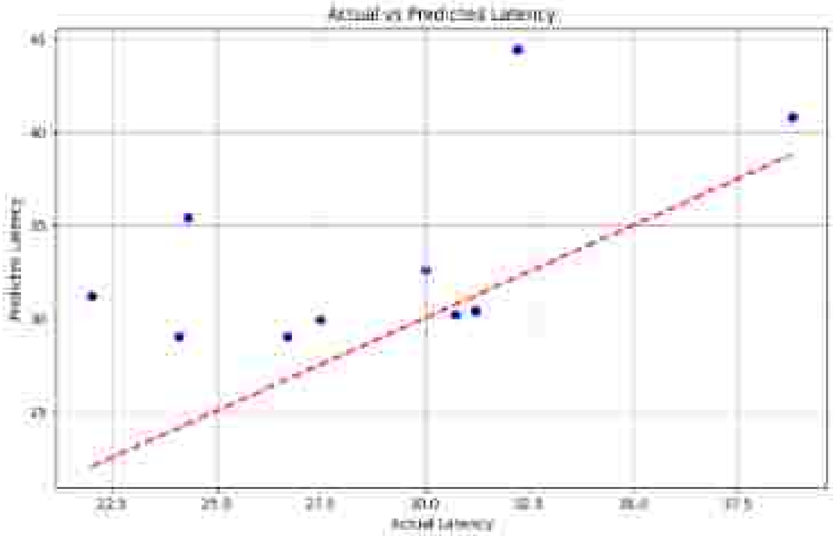


Figure 9. Actual vs Predicted Latency

	Coefficient
Packets per Second	0.043983
Average Packet Size	0.004750
Active Users	0.433762

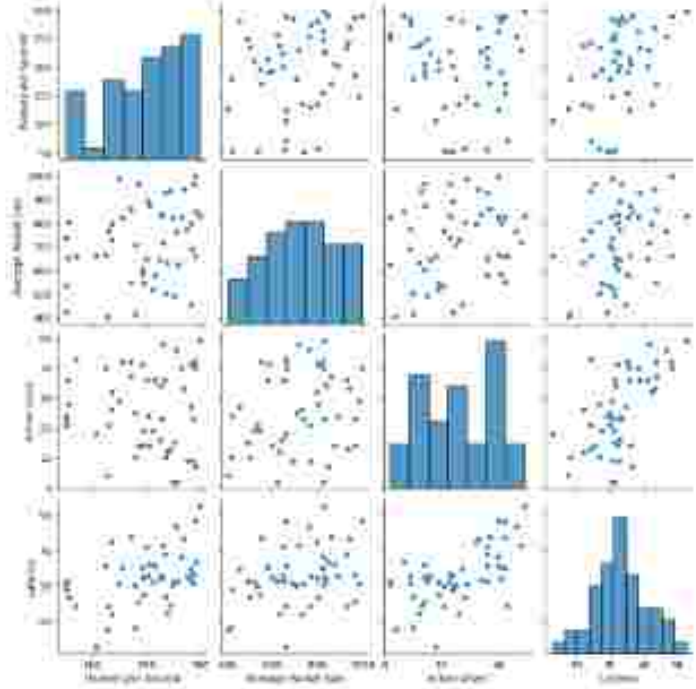


Figure 10. Packets per Second, Packet Sizem Active Users, Latency

Figure 11. Correlation Heatmap

F. Ordinary Least Squares Regression or OLS Regression

OLS Regression is used to estimate the value for a dependent variable obtained based on the values of one or more independent variables. In this case, throughput is predicted with the help of jitter, delay, and packet loss. OLS has been much in vogue due to its properties, unbiased and efficient estimates are acquired if the assumptions of classical linear regression are satisfied. OLS also allows for the analysis of each independent variable's effect on the dependent variable, thereby contributing to the in-depth understanding of the factors influencing network Throughput. Imagine, you have been employed as a data analyst to analyze the performance of a network regarding some router. You have the data that contains values for throughput, jitter, delay, and packet loss from 50 observations. You are to predict the throughput based on the values of jitter, delay, and packet loss using the ordinary least squares regression method.

1. Describe the basic concept of Ordinary Least Squares Regression, or simply OLS Regression.
2. Calculate the regression coefficients for variables Jitter, Delay, and Packet Loss in predicting Throughput.
3. Interpret the result of the regression.
4. Plot the actual and predicted values of Throughput.
5. Plot the relationships among all independent variables against each other with Throughput.

Using OLS Regression, it will be able to develop detailed information on how much the independent variables jitter, delay, and packet loss affect throughput, which will then assist you in specific improvements upon their need to perform well within the network.

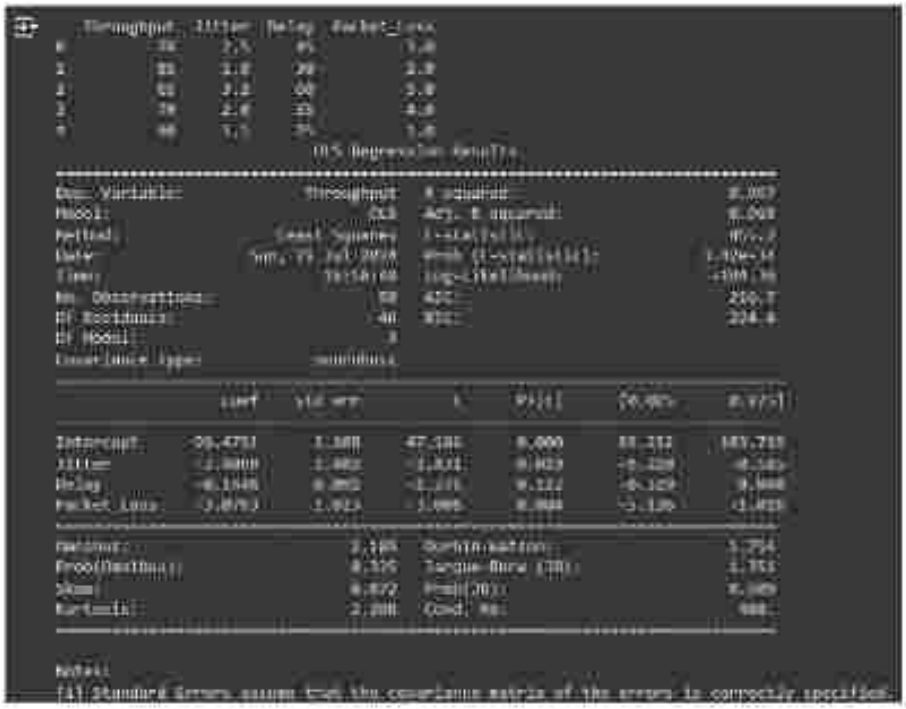
Throughput	Jitter	Delay	Packet Loss
78	2.5	45	3
85	1.8	30	2
65	3.2	60	5
70	2	55	4
90	1.5	25	1
55	4	70	6
60	3.5	65	5
95	1.2	20	0.5
85	1.7	35	1.5

Throughput	Jitter	Delay	Packet Loss
68	3	50	4
72	2.8	40	3.5
63	3.6	58	5.2
88	1.4	28	1.1
92	1.1	18	0.9
58	4.2	75	6.5
67	3.3	52	4.8
77	2.4	42	3.2
82	2	36	2.1
79	2.2	33	2.3
65	3.5	55	4.5
60	3.8	65	5.8
73	2.6	40	3.1
87	1.9	29	1.3
68	3.1	53	4.2
81	2.1	37	2
90	1.6	27	0.7
59	4.1	74	6.3
74	2.5	39	3
66	3.4	51	4.7
79	2.2	34	2.2
83	2	32	1.8
58	4.3	70	6
75	2.7	41	3.4
84	1.5	30	1
62	3.7	60	5
70	2.8	44	3.3
88	1.3	26	0.8
67	3.4	50	4.4
80	1.8	31	1.6
76	2.3	38	2.5
69	3	48	3.9
89	1.2	22	0.6
61	3.9	62	5.3
77	2.4	43	3.6
64	3.2	56	4.6
72	2.9	47	3.8
91	1.5	24	0.4

Throughput	Jitter	Delay	Packet Loss
59	4	68	5.9
85	1.6	28	1.2
74	2.5	39	3.1

```
# Import necessary libraries
import pandas as pd
import statsmodels.formula.api as smf
import matplotlib.pyplot as plt
import seaborn as sns

# Creating the dataset with predefined values
data = {
 'Throughput': [78, 85, 65, 70, 90, 55, 60, 95, 85, 68, 72, 63, 88, 92, 58, 67,
77, 82, 79, 65, 60, 73, 87, 68, 81, 90, 59, 74, 66, 79, 83, 58, 75, 84, 62, 70,
88, 67, 80, 76, 69, 89, 61, 77, 64, 72, 91, 59, 85, 74],
 'Jitter': [2.5, 1.8, 3.2, 2.0, 1.5, 4.0, 3.5, 1.2, 1.7, 3.0, 2.8, 3.6, 1.4, 1.1, 4.2,
3.3, 2.4, 2.0, 2.2, 3.5, 3.8, 2.6, 1.9, 3.1, 2.1, 1.6, 4.1, 2.5, 3.4, 2.2, 2.0, 4.3,
2.7, 1.5, 3.7, 2.8, 1.3, 3.4, 1.8, 2.3, 3.0, 1.2, 3.9, 2.4, 3.2, 2.9, 1.5, 4.0, 1.6,
2.5],
 'Delay': [45, 30, 60, 55, 25, 70, 65, 20, 35, 50, 40, 58, 28, 18, 75, 52, 42, 36,
33, 55, 65, 40, 29, 53, 37, 27, 74, 39, 51, 34, 32, 70, 41, 30, 60, 44, 26, 50,
31, 38, 48, 22, 62, 43, 56, 47, 24, 68, 28, 39],
 'Packet_Loss': [3, 2, 5, 4, 1, 6, 5, 0.5, 1.5, 4, 3.5, 5.2, 1.1, 0.9, 6.5, 4.8, 3.2,
2.1, 2.3, 4.5, 5.8, 3.1, 1.3, 4.2, 2.0, 0.7, 6.3, 3.0, 4.7, 2.2, 1.8, 6.0, 3.4, 1.0,
5.0, 3.3, 0.8, 4.4, 1.6, 2.5, 3.9, 0.6, 5.3, 3.6, 4.6, 3.8, 0.4, 5.9, 1.2, 3.1]
}

# Create a DataFrame
df = pd.DataFrame(data)

# Display the first few rows of the dataset
print(df.head())

# Save the dataset to a CSV file for easier visualization
df.to_csv('router_data.csv', index=False)

# Import statsmodels for OLS regression
import statsmodels.formula.api as smf
```

```
# Define the formula for the regression
# We will predict 'Throughput' based on 'Jitter', 'Delay', and 'Packet_Loss'
formula = 'Throughput ~ Jitter + Delay + Packet_Loss'

# Fit the OLS model
model = smf.ols(formula=formula, data=df).fit()

# Print the summary of the model
print(model.summary())

# Plotting the actual vs predicted values for Throughput
df['Predicted_Throughput'] = model.predict(df[['Jitter', 'Delay',
'Packet_Loss']])

plt.figure(figsize=(10, 6))
plt.scatter(df['Throughput'], df['Predicted_Throughput'], color='blue')
plt.plot([min(df['Throughput']), max(df['Throughput'])],
[min(df['Throughput']), max(df['Throughput'])], color='red', linestyle='--',
linewidth=2)
plt.xlabel('Actual Throughput')
plt.ylabel('Predicted Throughput')
plt.title('Actual vs Predicted Throughput')
plt.grid(True)
plt.show()

# Visualizing the coefficients
coefficients = pd.DataFrame(model.params, columns=['Coefficient'])
print(coefficients)

# Pairplot to visualize relationships between variables
sns.pairplot(df)
plt.show()

# Heatmap to show correlation between variables
plt.figure(figsize=(10, 6))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Heatmap')
plt.show()
```



Based on the OLS Regression output, the dependent variable to be forecast is Throughput, while the independent variables are jitter, delay, and packet loss. Out of the regression, the developed regression equation is given by:

$$\text{Throughput} = 99.4751 - 3.4069 * \text{Jitter} - 0.1446 * \text{Delay} - 3.0763 * \text{Packet Loss}$$

This is the value of the intercept, 99.4751, indicating that when jitter, delay, and packet loss are zero, then the predicted throughput will be 99.4751. In this case, the coefficient for jitter is -3.4069. Therefore, for a one-unit increase in jitter, the throughput decreases by 3.4069 units if other variables remain constant.

The p-value corresponding to Jitter is 0.010, which means that jitter has a significant effect on throughput at a 5% level of significance. The coefficient of delay is -0.1446, interpreting that for every one unit increase in delay, keeping all the other variables constant, the value of throughput goes down by 0.1446 units.

Meanwhile, according to Table 3, based on the result, the p-value of delay is 0.122, therefore, with a 5% significance level, delay does not have a significant effect on throughput. The coefficient on packet loss is -3.0763,

indicating that for every one-unit increase in packet loss, the throughput drops by 3.0763 units, other variables held constant. The p-value associated with Packet Loss is 0.004, hence the variable has a significant effect on Throughput at a 5% level of significance.

The models statistics show an R-squared value of 0.967, indicating that this model will explain 96.7% of variation in throughput, based on a model containing variables such as jitter, delay, and packet loss. This model shows an adjusted R-squared value equal to 0.965, indicating its very good at explaining the variation in throughput after considering the number of variables in the model. The F-statistic value equals 455.2 with a p-value of 3.42e-34, indicating the significance of the regression model overall.

The scatter plot in plots compares the actual and predicted values of Throughput. Apparently, the predicted values were very close to the actual value, thus the regression model is quite accurate in predicting Throughput. The pairplot has fair linear relationships between variables; in particular, the ones involving Throughput with jitter and packet loss are strong. On the correlation heatmap, these variables have a strong relationship.

In other words, it is clear that jitter and packet loss are major factors of impact on throughput from the analysis, while delay does not have any impact on throughput as per this model. What OLS Regression does clearly specify is how each independent variable affects throughput, which is quite instrumental in making a correct decision toward improvement in network performance. Hence, OLS Regression becomes an effective analytical tool for learning and achieving optimal network performance based on considered parameters.

	Coefficient
Intercept	99.475135
Jitter	-3.406933
Delay	-0.144565
Packet_Loss	-3.076309

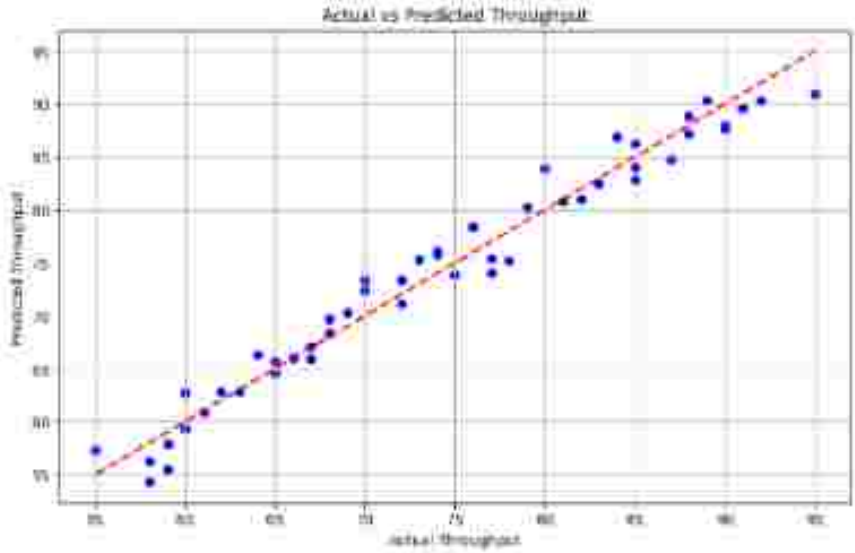


Figure 12. Actual vs Predicted Throughput

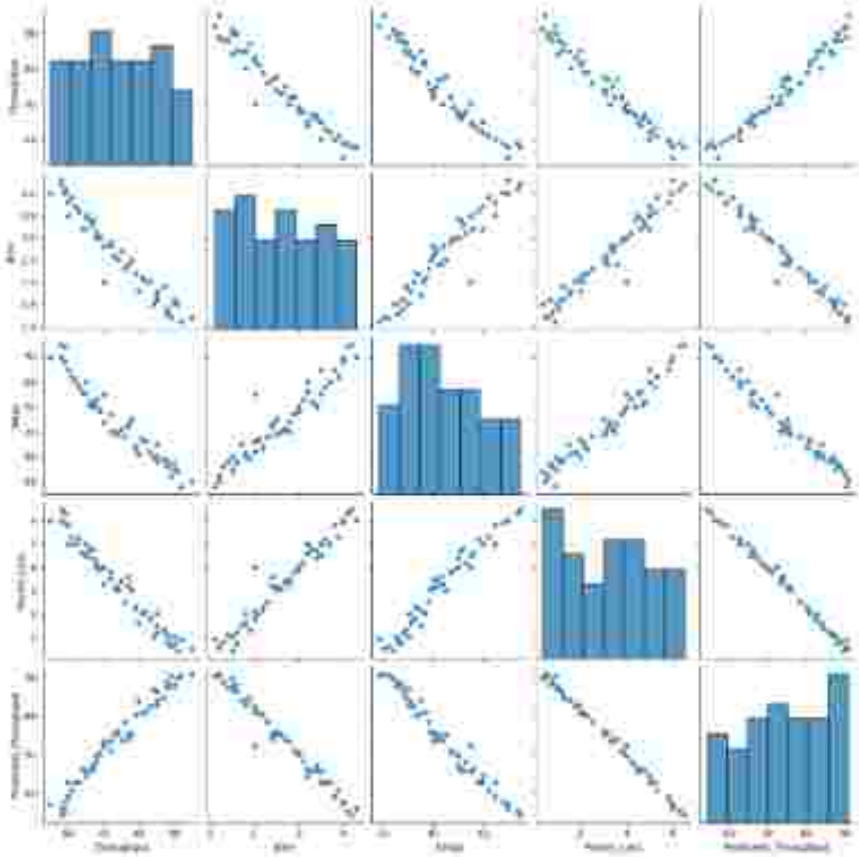


Figure 13. Throughput, Jitter, Delay, Packet Loss, Predicted Throughput

Figure 14. Correlation Heatmap