CS & AI

B722100

COC257

INVESTIGATING MACHINE LEARNING METHODS FOR HUMAN BEHAVIOUR ACTIVITY WITH DEPLOYMENT

by

Khrishawn Randal Samuel Powell

Supervisor: Dr SS Fatima

Department of Computer Science
Loughborough University

Abstract

The increase in the number of elderly peoples living longer lives means that many are going to be unable to look after themselves in their old age. With the current elderly population, many live alone if they are still able to look after themselves, or family members decide to put them in old people homes. This is because family members are beginning to live very busy lives in the pursuit of wealth and have other commitments. However instead of the elderly struggling when living alone or family members sending them to old people homes. Is there a way that we can enable them to live at home, but make their lives easier with the use of technology? The rise of Artificial intelligence has seen automation being used in many areas such as the use of robotics to reduce manual labour. However, is there a possibility to automate the homes of the elderly using AI by the use of a technique called Machine Learning? This project will explore how this can be done using machine learning to predicting human behaviour activity and based on the human behaviour activity that is predicted send messages via an API to smart appliances resulting in the automation of their homes.

This project will be done using a dataset from Kaggle titled “Human Activity Recognition with Smartphones”, this dataset contains a training and testing set. The training set is used to carry out training of the models that will be discussed in this project and the test set will be used to evaluate the effectiveness of these models on unseen data.

This project will look at machine learning methods such as K Nearest Neighbour, Decision Trees, Random Forests, Neural Networks, Logistic Regression and Support Vector Machine. This project will also look to evaluate different parameters that impact the accuracy of the different models. In addition to this will also feature the deployment of the best model using an API created with FastAPI. This allows the model to be deployed in the future where further exploration on how this API can be used with smart appliances will be done. This project is carried out with the help of Python and many different libraries which is discussed during the implementation of this project.

Acknowledgements

I would like to thank my close family, my Father, Leslie Powell who taught me to make sure I am ambitious in life, his teaching enabled me to strive to where I am and causes me to strive further.

I would like to thank my Mother, Lena Williams that worked hard throughout my childhood to put food on the table and taught me to never give up no matter the circumstances and always loves me.

I would like to thank my Brother, Gareth Powell for at his young age continually inspiring me to not care about being like everyone in the World, and with his hard work and drive gets it popping.

I would like to also thank my close childhood friends, Tariq, Tyreke, Tobi, Nino, Ore, Eddy and Maxwell for supporting me in times of need.

I would like to especially thank Tariq a close friend who proofread this document and taught me to chase excellence academically since I was approximately 13, provided me with the blueprint and allowed me to replicate it and continues to teach me to chase excellence till this date.

I would like to thank the friends I met during University.
A1, Joel, Mayowa, Fadhil Jude and Ira.
Most importantly I'd like to thank Joel for sharing his pain with me whilst at Loughborough but more importantly supporting me when I was in the gutter and allowed me to share my pain too.

I would like to thank my Church Family for their prayers and support.
David J, Marlene J, Jacky M, Keith, Jennifer, Kadila, Kirshan, Bruno, Hannah, Claudette C, Claudette M, Sharon G, Pastor Herby, David M, Lorna, Lena, Juliett, Gareth.
Most importantly I'd like to thank Pastor Curdell who taught me I was destined for greatness.

I would like to thank my childhood teachers especially Denise Hilliman and Mr Osebor who despite my unruly nature believed in my ability and pushed me further.

I would like to thank the people I met in my year out of Loughborough who inspire me and push me to get it going. Victor, Denzil, and everyone else.

I'd like to thank my mentor Carl Konadu who pushed me to get it out the mud when I was going through my final year and provides me with opportunities to be great and inspires me to one day be someone that can give hope to many in my area.

I would like to thank those in my Network, that give me opportunities to shine and also chase greatness.

Most importantly Demetrius Bethea who supports me in my endeavours and sees potential in me to be great.
I'd also like to thank Siobhan Baker who taught me on my year out of Loughborough how to program in Java and think like a developer.

I would like to thank all Loughborough members of staff who supported me in some way throughout my 3 years. Most importantly my Final Year supervisor that allowed me to contact her throughout the journey of this project.
I'd like to thank Bishop Jan and Loughborough Student Services that encouraged me to not give up on life when it became extremely hard.

Last and not least I would like to thank the man above for helping me through the painful years at Loughborough University and helping me to get to the end of my studies and blessed me in ways I cannot describe.

Pain brings purpose, trust the journey!

Contents

1 – Introduction

According to Arthur Samuel, an American Pioneer in Computer Science *"Machine learning is the field of study that gives computers the ability to learn without being explicitly programmed"* (Sciencedirect.com, 2017). Whilst this is a general definition, American Computer Scientist Tom Mitchell provides a more engineering focussed definition: "*A computer program is said to learn from experience (E) concerning some task (T) and some performance measure (P), if its performance on T, as measured by P, improves with experience E"* (chinmay da s, 2017). Overall machine learning can be defined as the process at which statistics and computer science is brought together to enable machines to think like humans (OxfordSparks, 2017). It does this by training algorithms to find patterns and features within large amounts of data.

According to IBM, Machine learning is used everywhere. These are some ways machine learning is used:

- Digital Assistants - Apple Siri, Amazon Alexa and many other digital assistants all use machine learning. These assistants are powered by natural language processing (NLP). As a result, they can process human languages and understand what we say to them, like humans would (IBM Cloud Education, 2020).
- Recommendations – Machine learning models are used during customer experiences to recommend things to customers they might like (IBM Cloud Education, 2020). For example, Spotify recommends users music based on what they commonly listen to using ML.
- Online advertising- Machine learning can evaluate the contents of a web page that a visitor visits and provide them with online advertisements based on the webpage they previously viewed (IBM Cloud Education, 2020).
- Chatbots – Chatbots use machine learning in the sense of NLP and deep neural networks to read the text and give a response to users.
- Fraud detection – Banks use machine learning to build up knowledge on how customers spend their money. This enables banks to detect unusual transactions made on bank cards and verify the customer's identity. (IBM Cloud Education, 2020).
- Cybersecurity- ML has also been used to detect possible threats or anomalies in networks and advise those in charge of the security of networks what is going on to enable them to make a response (IBM Cloud Education, 2020).
- Medical field - AI is here to process the image data on patients. Systems can now look at the image of a human's organ and diagnose them.
- Self-driving cars – Cars like the famous Tesla use machine learning to continuously be aware of objects and the environment around them. As a result, these cars can autonomously drive (IBM Cloud Education, 2020).

Overall, machine learning is being used to gain insights into data and solve complex problems for us that we may previously have never been able to solve.

Whilst it can be said that ML can be found everywhere there is not enough use of ML in the home as only 20% of the public use smart home devices that use an element of artificial intelligence (by RJ Reinhart, 2018). The lack of homes with smart home devices shows that there is greater room for growth in the number of people that use AI in their homes.

In addition to this, the advancement of medicine and healthcare is causing the elderly to live longer. The Office of National Statistics states 18% of UK residents are aged 65 and older (Ons.gov.uk, 2018). But with this carry's negatives, such as the possibility of elderly people growing too old to the point they cannot look after themselves and needing assistance in their homes. To solve this, Ambient Assisted Living can be used. Ambient Assisted Living can be defined as "the use of information and communication technologies (ICT) in a person's daily living and working environment to enable them to stay active longer, remain socially connected and live independently into old age" (AAL Programme, 2021).
Therefore, I have decided to carry out my project on machine learning on human behaviour. This is because I believe it is possible to automate a human home based on data collected on their actions. This can, in turn, lead to old people being able to look after themselves better when living on their own, and this technology could be tailored to the wider public to help those that want to live more comfortable lives. We will be looking to explore this by comparing the different machine learning techniques otherwise known as models on a dataset containing human behaviour activity and deploy the best technique so it can be used in real life.

1.1 - Aims and Objectives

The project will focus on the evaluation of different machine learning models based on the change of model parameters on human behaviour activity, but will also feature the deployment of the best model that has been evaluated. To help work prioritisation MoSCoW Prioritisation will be used.

MoSCoW is a prioritisation technique that is used to understand the major priorities of a project (Agilebusiness.org, 2021).

The letters stand for:

- **M**ust have
- **S**hould have
- **C**ould have
- **W**on't have this time

Therefore, according to MoSCoW the prioritisation for this project will be:

- **M**- Must carry out hyperparameter tuning on all models I use and evaluate their effectiveness on different models.
- **S**- Should ensure that I deploy the model as an API.
- **C** – Could build a UI and deploy the best model as a web application.

- **W** – Will not develop a system that automates your home based on human behaviour activity.

This project will begin with the creation of the ML pipeline by watching tutorials and reading information that give the fundamentals on how ML is carried out. Next information that shows us how to carry out hyperparameter tuning and then deploy the model as an API will be explored. After this, the project will be implemented.

The language that will be used is Python. This is because, having carried out some prior research, it can be said that Python is the best language to use when carrying out machine learning, due to the libraries it offers. These are some of the libraries:

- Scikit Learn
- Numpy
- Pandas

Overall, the main objective of this project is to investigate different machine learning methods on a human behaviour activity dataset which will be located on Kaggle.

There is an aim to reach a threshold of 90%+ accuracy on the test data before the decision to take any models further and deploy them. This is because it is important to make sure the model is reliable for the real world.

2- Literature review

2.1 Gap Analysis

In this chapter, the different academic papers which are relevant to this project will be evaluated for what knowledge can be gained from them but also where they have failed or been unable to cover specific areas. The areas they fail to cover will be added to this project.

Machine Learning Methods for Classifying Human Physical Activity from On- Body Accelerometers
((5) (PDF) Machine Learning Methods for Classifying Human Physical Activity from On-Body Accelerometers, 2021a)

Research carried out by Andrea Mannini and Angelo Maria Sabatini highlighted the use of human behaviour activity in the real world. It discusses how information about physical activity can be used to help the elderly to walk via the use of rehabilitation engineering. It then goes on to discuss the process at which a generic classification system can be carried out for supervised learning. It then discusses how wearable sensors must be lightweight and small when carrying out analysis on human behaviour activity as anything bulky is invasive to humans. It discusses that since there are many features to take into consideration when determining the class of an activity PCA can be used to reduce the dimensionality. With this

literature review, we can gain a greater understanding of what the current uses of Human Behaviour Recognition are, and the knowledge of what methodology should be used.

Applications of Machine Learning Techniques in Human Activity Recognition
(Rana, Jha and Shetty, n.d.)

This research that was carried out by Jitenkumar B Rana, Tanya Jha and Rashmi Shetty. It highlights that out of the 4 algorithms they used on their Human Behaviour Activity dataset. The best algorithm was Random Forest with 99.8% accuracy with 10 trees this shows that a more careful look at Random Forest should be carried out in our evaluation. However, they did not carry out thorough Hyperparameter tuning and discuss what happens when different parameters are changed and as a result, we will investigate this and ensure that it is part of our research.

Machine Learning Algorithms for Human Activity Recognition
(Machine Learning Algorithms for Human Activity Recognition, 2019)

From reading this piece of research it has become apparent that when using Human Behaviour activity to assist living conditions no single type of living system sensors is used. This is because when carrying out ambient assisted living multiple different sensors must be used to capture different information. For example, temperature sensors for temperature, pressure sensors for pressure and door sensors for door movement. This, therefore, makes us aware that once completing this study, to take findings further in the Ambient Assisted Living space the use of the different type of sensors must be considered.

This research paper then goes on to discuss that in human activity recognition a vital part of it is the sensors and evaluation of information through a recognition engine. It then goes and discusses how these sensors capture data on the X-axis, Y-axis, and Z-axis. The X-axis in sensors is used to indicate sideways or horizontal movement. The Y-axis is used to indicate upwards or downwards movement. The Z-axis is used to indicate forward or backward movement. This therefore makes us aware of the kind of data sensors collect and what they mean. It then goes on to discuss that when carrying out ambient assisted living the Hidden Markov Model is best used to carry out speech recognition. Dictionary learning is best used when analysing someone sitting down or standing up from chairs whilst Neural Networks are best used for arm posture recognition. Overall knowledge in learning about why different kinds of models should be used to monitor different sensors and why different sensors must be used in the carrying out of ambient assisted living was gained. In this research paper they did not carry out an effective evaluation of different machine learning algorithms and deployment, therefore this adds confidence to our aims and objectives of the project.

Recognition of Daily Human Activity Using an Artificial Neural Network and Smartwatch
(Kwon and Choi, 2018)

In an experiment carried out by Kwon and Choi. The models they used were Decision Tree, SVM, Random Forest and Artificial Neural Network. The best performing algorithm was Random Forest, followed by the Artificial Neural Network. Decision tree and SVM achieved an accuracy of less than 75%, meaning there is no way they could be used to classify any activities. In addition to this, their research used Accuracy, Precision, Recall and F1 score to evaluate each model which the previous research discussed did not. Overall, this paper shows what was the best models were based on their dataset but did not go into deploying the model. In addition to this, the use of evaluating the models using F1, Precision and accuracy score is interesting and we'll look to incorporate this as a possible metric for evaluating results.

Overall, these research papers have shown the methodologies these researchers used. This has therefore shown the methodology we should use and because of this we can have confidence in carrying out this project. They have also shown the benefit of human behaviour activity and how it has been used, this, therefore, has increased the scope of vision of what is possible post-University. In the research paper carried out by Kwon and Choi and by Jitenkumar B Rana and others it shows that Random Forest is the best. Due to this we'll be exploring if this is true for this project also. An understanding of why we should reduce the dimensionality of our Human Body Activity Dataset was given by Andrea Mannini and Angelo Maria Sabatini. We also gained an understanding of what the XYZ -axis values represent and the usefulness of tracking F1 Score, Precision, Accuracy and Recall. With all these papers one common problem is that we are unable to find an extensive report on the effects that different hyperparameters have on the accuracy of some models and did not find any researchers that deployed their findings. Therefore, these will be included in this project.

3 -Plan

3.1 Original Work Plan

Week	Project Initiation	Technical Research	Literature Review	Programming	Deployment	Testing	Evaluate findings	Project Diary
w/5.10								
w/12/10								
w/19.10								
w/26.10								
w/2.11								
w/9.11								
w/16.11								
w/23.11								
w/30.11								
w/7.12								
w/14.12								
w/21.12								
w/28.12								
w/4.1								
w/11.1								
w/18.1								
w/25.1								
w/1.2								
w/8.2								
w/15.2								
w/22.2								
w/1.3								
w/8.3								
w/15.3								
w/22.3								
w/29.3								
w/5.4								
w/12.4								
w/19.4								
w/26.4								
w/3.5 - Deadline Week 5.05								
w/10.5								

3.2 Revised Work Plan

4 – Design

4.1 – Python Research

In Chapter 1.1 it was discussed that the language that will be used is Python, due to the libraries that are available to enable effective application of machine learning.

Before the creation of this project, we were never taught how to program in Python throughout our 3 years at Loughborough University. However, we were taught similar object-oriented programming languages like Java in the first year with the module **Introduction to Programming**. We then had to build upon this knowledge with the use of Java in **Mobile Application** development and **AI methods**.

4.1.1 – Pluralsight

Due to our lack of knowledge in Python, we will be learning the fundamentals of Python using Pluralsight, an online educational platform for those in the field of technology and IT.

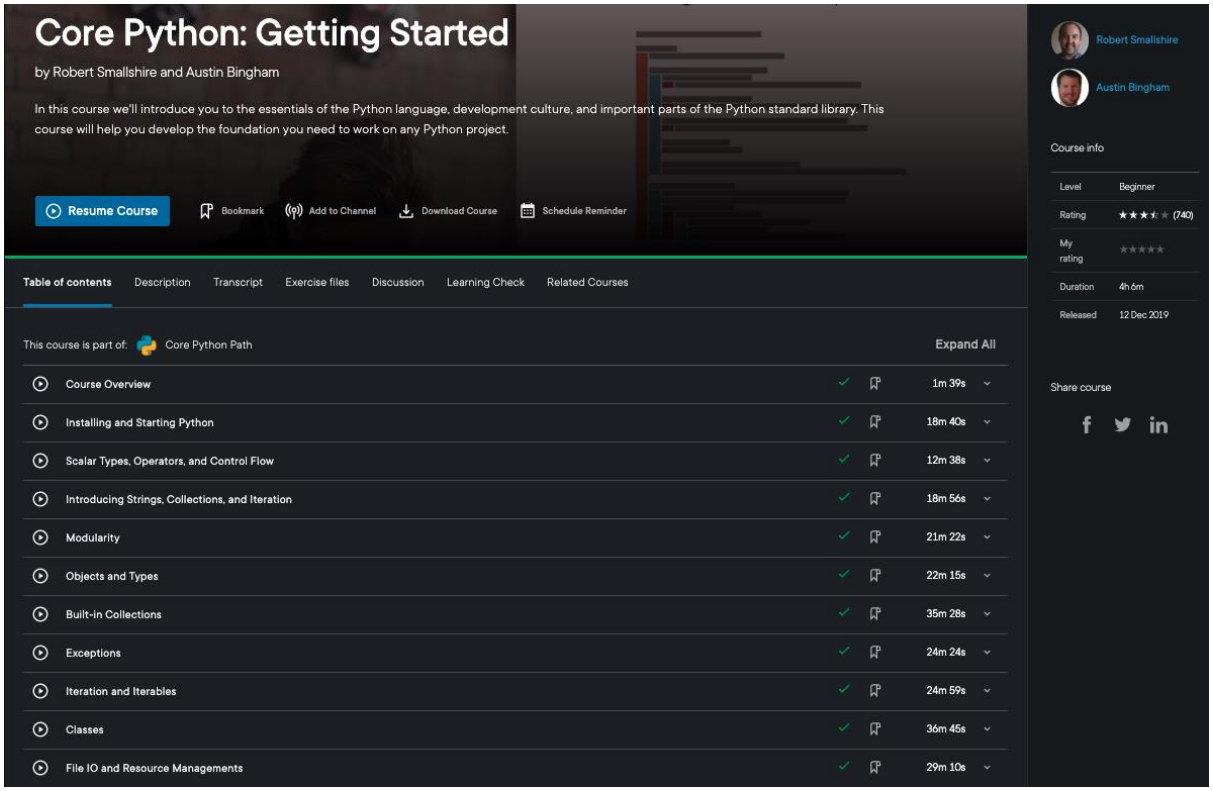


Figure 1 Pluralsight Python Course Content

After learning Python, it is evident that Python is very similar to Java due to its object-oriented design. However, it has its differences such as the syntax of Python being simpler and more concise to interpret. Because of this you can write fewer lines of code with Python to get the same result as Java. In addition to this, Python has many libraries that support different tasks and objectives where Java does not.

4.2 – Machine Learning Research

Whilst during my 3 years at Loughborough University I was not taught how to carry out machine learning however was taught about the use of different AI methods by Dr Syeda Fatima. It can be said the module was extremely enjoyable as a vast amount of knowledge was gained which was never known beforehand, resulting in my ability to grasp new concepts to increase. As a result, this module gave me the confidence to go out and learn how to carry out machine learning.

4.2.1 – Pluralsight

To get knowledge on how to carry out ML the course 'Building Your First Machine Learning Solution' was used on Pluralsight.

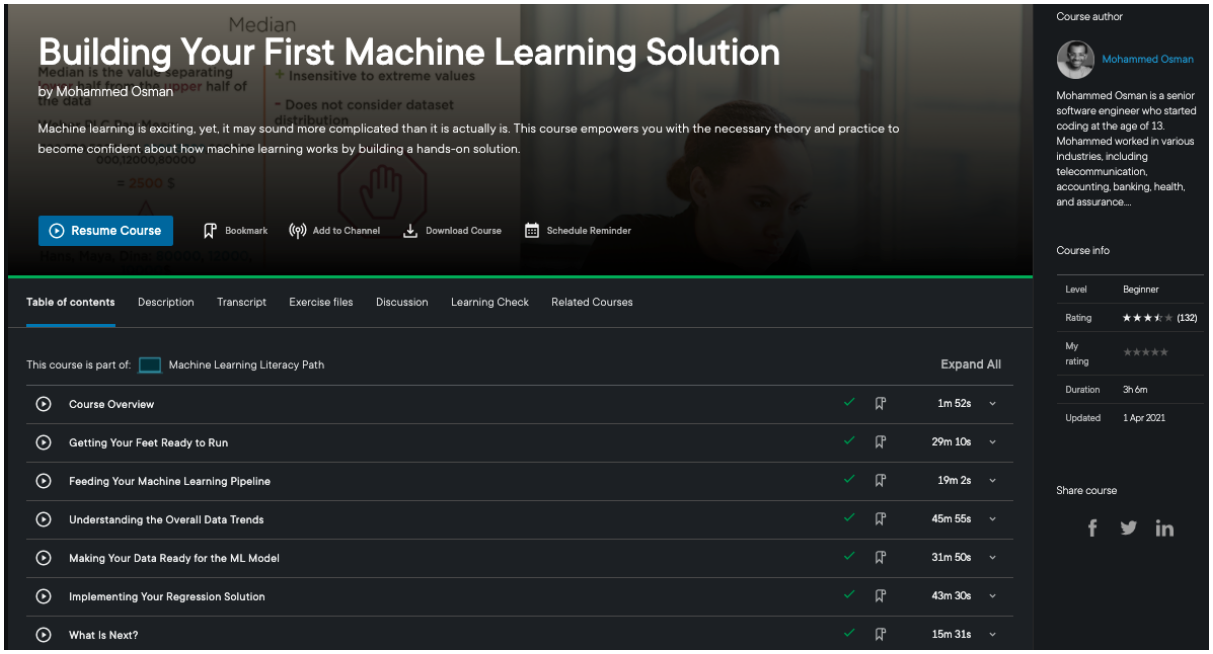


Figure 2 Pluralsight ML course

Using this course to learn how to carry out ML, highlights how Python is used to work on ML. Whilst it gives us a clear understanding on how ML is carried out it is still not totally clear as a result of this other sources will be used.

4.2.2 – Code Academy

(The Machine Learning Process | Codecademy, 2021)

As discussed above in the previous section the knowledge gained from Pluralsight was not sufficient. Because of this a greater clarification was needed and the following knowledge below was gained from Code academy.

There are several processes involved in the machine learning process:

1) We should decide what problem we are going to solve
2) Collect our data and understand what it contains
3) Clean the data and engineer its features
4) Choosing a Model
5) Tuning and evaluating the model
6) Deploying the model or presenting its results.

Deciding what problem, we are solving.

During machine learning it is important to know what problem you are trying to solve. This, therefore, enables you to decide the best predictor that allows you to solve your problem. For example, if you are trying to reduce the amount of bank fraud claims, you could decide that you want to predict what customers could make fraudulent claims. This could be done by getting data on customers as they register for the bank such as occupation, address, demographic and much more and using this information alongside previous fraudulent

claims to predict if certain customers are a risk to the bank. From this, you can keep a closer eye on customers or just refuse them to join the bank

It is also important to understand what kind of problem you will be solving using ML.
For example, is the problem an unsupervised, supervised r,einforcement learning, classification, or a regression task.

Knowing this and understanding our problem can therefore allow us to correctly choose our model.

From our research discussed before it is aware, we will be using supervised classification algorithms. This is because prior to starting this project a brief look at the dataset located on Kaggle was carried out. It is evident that this project will be classifying what category different instances of data come from as the dataset contains labelled data.

Collect our data and understand what it contains

The process of collecting data can be done in many ways. First, we can decide to use data that has been externally collected. This can be found online on websites such as Kaggle, or open-source datasets like the Popular Iris Dataset or MNIST. Alternatively, we can decide to collect data via the use of sensors, surveys, visual analysis, research and much more. We must then understand what our data contains by reading information on it or its features.

Clean the data and engineer its features

The process of cleaning and engineering features is done by removing noisy, incomplete, or inconsistent data from the collection. Missing data can be dealt with by ignoring the tuple, filling the missing value manually by finding the missing data or using your general knowledge to enter the data or using the central tendency or median to ensure that the data is consistent. To remove noise, you can carry out binning. Binning methods are carried out by sorting the data into bins or buckets after this smoothing by mean, median and bin boundaries are then carried out where each bin is replaced by the mean, median or bin boundaries (Softwaretestinghelp.com, 2021). We may also decide to normalise or standardise the data or even adding new columns.

Choosing a model

When choosing a model, we must now take into consideration our problem and what type of model is required. For example, are we predicting a number such as how long someone would take to complete a given task, or how much someone makes at a specific job?
For this, we use regression models only.

However, if we are trying to predict what shoes someone likes to wear based on their interest, we can do these using classification models as what is being predicted is discrete.

Regression and Classification problems will be discussed further in the next section.

We must then tune and evaluate our model.

We can tune and evaluate our model based on our metrics of success which is usually the accuracy of any given model. Each model has certain parameters which, if changed, can cause the accuracy of the model to increase or decrease. As a result, enables us to be able to tune the algorithm for the greatest level of accuracy. When tuning our model and evaluating it, it is important that we plot graphs to visualise our results. This enables us carefully to understand and visualise what is working better and make conclusions as to why.

Deploying the model and presenting its results.

Once we have carried out the above processes and achieved our desired level of accuracy on our training set, we must then test it on our test set to see how it behaves on unseen data. If the accuracy given from unseen data is equally high, we can decide to then present the results and deploy the model. Models can be deployed using API's, websites, mobile apps, and hardware.

4.3 - Types of ML problems

In this chapter, we will be discussing the different types of ML problems that we can face and how to decide what type of ML problem they are. This is because the type of ML problem causes us to use different algorithms when building a model.

4.3.1 Regression problems

A regression problem can be described as a problem where the output variable is continuous and not discrete. Values such as salary and height are regression problems. Some examples of regression models are Linear regression, Polynomial Regression, Support Vector Regression, Decision Tree Regression and Random Forest Regression (GeeksforGeeks, 2017b).

3.3.2 Classification problems

Classification problems can be described as a problem where the output is a specific category and is therefore discrete and not continuous. For example, "Running" and "Walking". A classification model works by trying to make conclusions based on the values present. For example, when determining if a human being is running a model will look at all the other features which are the sensor values and will decide based on the data available what category does the data entry belong to. Some examples of classification models are Logistic Regression, Decision Tree, Naive Bayes, Random Forest and Neural Networks (GeeksforGeeks, 2017b).

The implications of this section for this project shows that since the data being used will consist of discrete data, models that help carry out regression problems will not be used but instead I will focus on using algorithms that carry out classification problems.

4.4 - Types of ML

In this chapter the different types of Machine Learning will be discussed. This is because the type of values within our dataset dictates the type of learning method that will be required to carry out, this will be discussed below.

4.4.1 - Supervised Learning

Supervised learning is learning which is carried out by manually training the machine learning model via the use of a labelled dataset so that it can easily predict results from a dataset.
When carrying out supervised learning you must ensure you create a model using labelled data that contains input data and output data. You must then run a supervised learning algorithm on the data set and train the model. At this point, you must then test the model ,after this point a percentage of accuracy will be received and based on the accuracy you can change the hyperparameter to produce the correct result (IBM Developer, 2017). At this point, you can input new observations into the model and get a prediction or classification as shown in **Figure 3**. Examples of supervised learning algorithms include K-Nearest Neighbours, Linear Regression, Logistic Regression Support, Support Vector Machines, Decision Trees, Random Forests and Neural networks (Aurélien Géron, 2017).

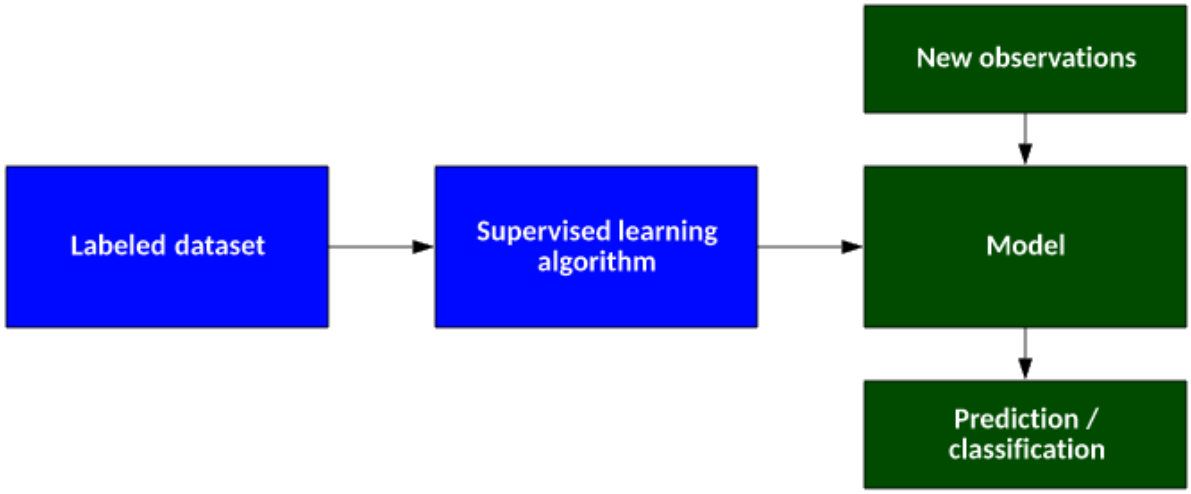


Figure 3 Supervised learning algorithm ***(IBM Developer, 2017)***

4.4.2 - Unsupervised Learning

Unsupervised learning is learning which is carried out automatically without human input when teaching the system. As a result of this, the data is unlabelled. Unsupervised learning algorithms group unlabelled data based on the hidden features present within the dataset. Since the data is unlabelled there is no way to evaluate the result of the models and as a result, the data must be

grouped when hidden features are discovered (IBM Developer, 2017b). This enables the model to learn something about the data it did not know before as shown in **Figure 4**.

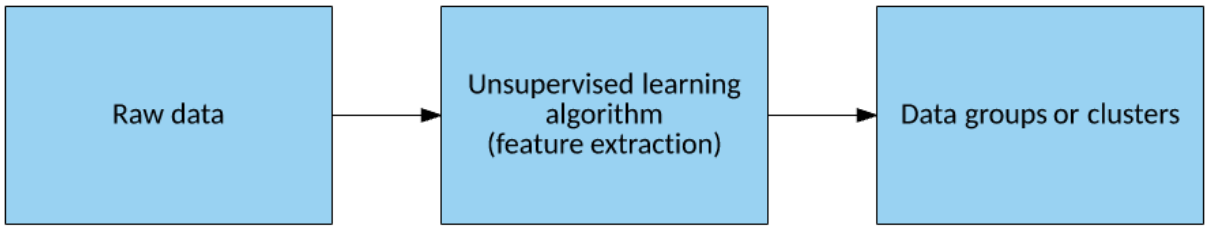


Figure 4 The process of Unsupervised learning (IBM Developer, 2017b)

An example where unsupervised learning would be useful is where you have data on people that purchase a specific product from you. But you do not tell the algorithm the age groups or classifications of each customer; as a result the system is forced to deduce what groups these customers belong to. Examples of Unsupervised learning consist of K-Means Clustering, Kernel PCA and much more (Aurélien Géron, 2017).

To conclude, this chapter as the dataset that will be used consists of labelled data, I will have to carry out supervised learning on this dataset. This means we will be manually training the machine learning model and therefore will explore supervised learning methods. In the next chapter, the different commonly used supervised algorithms for classification problems will be discussed.

4.5 - Comparison of different ML Models

In this chapter, I will be discussing the different supervised learning algorithms that support classification problems. What the model is; how it works, what the advantages and disadvantages are, what its hyperparameters are and what is its loss function.

A Hyperparameter is a value that is used to control the learning process of the model and as a result can impact its predicted output (Wikipedia Contributors, 2020).

A loss function is a method to evaluate how well a specific algorithm models data. If the loss function is too large this means the prediction is too far from the actual results (Ravindra Parmar, 2018).

4.5.1 - K Nearest Neighbour

K-Nearest Neighbours is a supervised machine learning algorithm that seeks to find patterns within data. It is not parametric and therefore it does not make any assumptions about the distribution of data. KNN is commonly used in classification problems, (AnalyticsSteps, 2020) KNN works by thinking every data point which is close to each other is therefore within the same class or category.
This algorithm works by selecting a number k which is a close neighbour to that data entry. For example, if K equals 8 the algorithm will seek to find 8 nearest data neighbours to that data point. See **Figure 5** for visual representation. As a result, the greater the number of K, the broader the data search and the smaller the value of K the narrower the data search. In

addition to this smaller, as the value of K increases the lower the amount of confidence in the prediction as the prediction becomes broader.

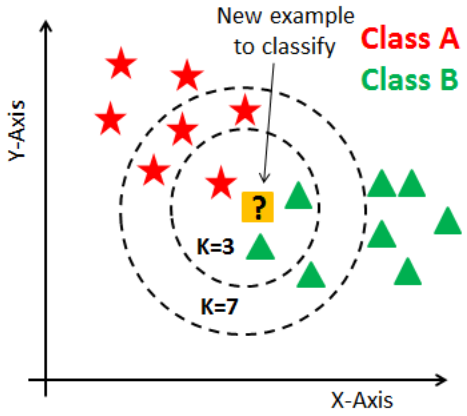


Figure 5 K-Nearest Neighbour (Avinash Navlani, 2018)

Loss Function

No training is required in KNN. This is since KNN instead partakes in classification or regression (Varghese, 2018a).

Advantages

- Easy and simple to implement (Varghese, 2018a).
- Low amount of hyperparameters to tune (Varghese, 2018a).

Disadvantages

- If the right value of K is not selected this can cause problems, therefore special care must be taken when selecting K (Varghese, 2018a).
- If the sample size is large, it will require a lot of computational power and time to carry out the algorithm (Varghese, 2018a).
- The data must be scaled well to ensure features are treated fairly (Varghese, 2018a).

Hyperparameters

The hyperparameters for KNN consist of two values: The K-value and distance function.

- **K value** – The amount of neighbour data points that should be included in the algorithm should be changed till the optimal K value is reached based on the magnitudes of error (Varghese, 2018a).

- **Distance Function** – This is to tell the patterns within the data. The commonly used distance function is the Euclidean distance but there are alternatives like the Manhattan distance and hamming distance (Varghese, 2018a).

Comparisons

KNN VS Linear Regression

- When the signal to noise ratio is high KNN is better than Linear Regression.

KNN vs SVM

- KNN does not handle outlier's well compared to SVM.
- If the training data is more than the number of features (m >>n). KNN operates at an advantage over SVM. SVM does better than KNN when the number of features is large and not more than the training data (Varghese, 2018a).

KNN vs Neural Networks

- Neural Networks require large amounts of training data whereas KNN does not require so much data to achieve a certain amount of accuracy.
- KNN does not require much hyperparameter training compared to NN.

4.5.2- Logistic Regression

Logistic regression is an algorithm used to calculate the odds of a particular event occurring. It is a supervised machine learning algorithm that is commonly used to solve classification problems. Classification problems are essentially when you solve data into categories. Logistic Regression operates based on the sigmoid function to find the relationship between data it takes it inputs any real number and outputs any number between 0 and 1 which can be seen from **Figure 6** below (Saishruthi Swaminathan, 2018) .

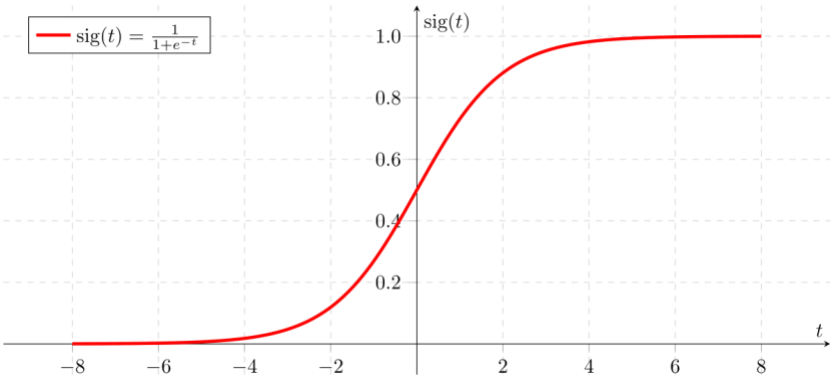


Figure 6 Sigmoid Function (Saishruthi Swaminathan, 2018)

Logistic regression algorithm can be used in the detection of coronaviruses by collecting symptoms and classifying a patient if their symptoms indicate they do where 1 is if they have it or 0 being not.

The output of logistic regression will be a probability ranging between 0 and 1 and can be used to predict “t” a binary value where 0 represents false and 1 represents true. If the probability outputted is less than 0.5 the output is 0 (false) and if the probability outputted is anything else the output is 1.

Loss Function

Due to the fact logistic regression uses a non-linear support function at the end the mean squared error cannot be used as a loss function.

In addition to this, the mean squared error will introduce local minimums and as a result, will lead, to an impact on the gradient descend algorithm.

Instead, cross entropy is used where two equations should be used where y = 1 and Y =0. Cross entropy is a measurement of the difference between two probability distributions. when the prediction is extremely wrong and causing errors, the cost will be -log (0) which is infinity (https://www.facebook.com/MachineLearningMastery, 2019).

Advantages

- It is easy fast and safe (Varghese, 2018a).
- The theta parameters explain the direction and intensity of significance in term of the prediction that is made (Varghese, 2018a).
- Can be used for multiclass classification which means it can be used to cover multiple types of data (Varghese, 2018a).

Disadvantages

- Cannot be applied to non-linear classification problems and as a result cannot be used on all datasets (Varghese, 2018a).
- A good signal to noise ratio is expected and as a result makes coming to a solution much more complex (Varghese, 2018a).

Hyperparameters

The Regularization parameter which is lambda is used to control overfitting in the data. The higher the value of lambda the more biased your solution will be. Whereas the lower your lambda value the more variance it will have.

Theta is used to calculate the learning rate. It estimates how much the theta values should be changed while the gradient descent algorithm is being applied (Varghese, 2018a).

Logistic regression vs SVM

- SVM can handle nonlinear solutions whilst logistic regression can only handle linear solutions (Varghese, 2018a).

- Linear regression and SVM handle outliers due to the fact is derives the maximum margin solution (Varghese, 2018a).

Logistic regression vs Decision Tree

- Collinearity is better dealt with decision trees compared to linear regression
- The significance of features can be dealt with easily with linear regression however not with decision trees (Varghese, 2018a).
- Categorical entries are best dealt with using decision trees compared to that of Linear Regression (Varghese, 2018a).

Logistic regression vs Neural Networks

- Linear Regression cannot handle nonlinear solutions whereas neural networks can.
- Neural Networks may hang on the local minima whereas Linear Regression does not hang due to the convex loss function.
- Linear regression does better than Neural Networks when the dataset contains fewer data entries and features are large, compared to Neural Networks which needs large training data.

Logistic regression vs KNN

- LR and LR do not use the same type of parametric model, LR is parametric whilst KN is non-parametric.
- When comparing KNN to LR is apparent that KNN is slower than LR.
- LR can only be used for linear solutions whilst KNN can only be used for non-linear solutions.

4.5.3 - Support Vector Machine

Just like Logistic Regression, SVM (Support Vector Machine) is used to solve classification problems however whilst it does this it also can solve regression problems however is commonly used during classification problems. The algorithm creates a line that is used for these lines are also otherwise known as hyperplane decision making and as a result separates data into classes to easily classify the data. It uses the kernel trick. The reason why it uses the kernel trick is that there can be loads of ways to separate the data and create hyperplanes as shown in **Figure 7** below (Bassey, 2019).

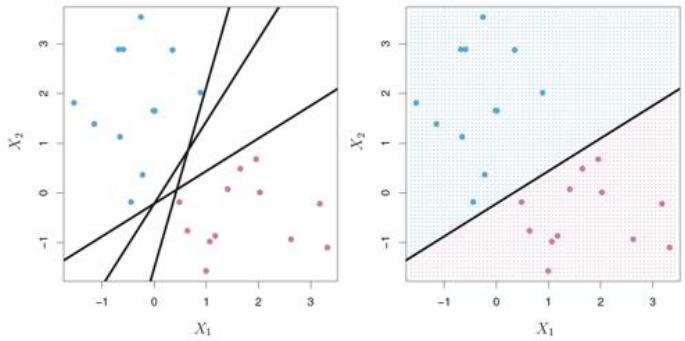


Figure 7: Several hyperplanes (Medium, 2019)

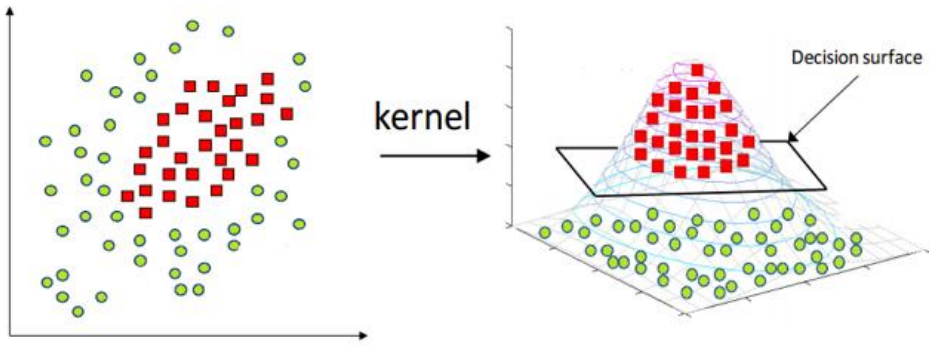


Figure 8 Data from 2D to 3D space

As a result, the kernel trick maps data from a 2d space to a 3d space allowing us to find a surface area that distinctly separates the data points. However, when there are more dimensions, computations become more costly as there is more surface are the computer must calculate this is shown in **Figure 8** (Medium, 2019).

For the hyperplane to be derived a maximum margin must be calculated. The hyperplane is then the midpoint.

The margin(m) is inversely proportional to ||w|| w represents the group of weight matrices. To ensure the margin is maximized ||w|| must minimize the optimization.

$$\text{Minimize} \quad \frac{||\vec{w}||^2}{2} \quad \text{where} \quad y_i\,(\vec{w}.\vec{x}+b) \geq 1 \text{ for any } i = 1,\ldots,n$$

Figure 9 Optimization problem (Varghese, 2018c)

However, when dealing with outliers we must use the following equation:

$$\text{Minimize } \frac{1}{2}\|w\|^2 + C\sum_i \max(0, 1 - y_i(w^\intercal x_i + b))$$

C represents the regularization parameter that balances the error penalty and margin width (Varghese, 2018c).

Loss Function

$$1)\ \frac{1}{2}\|w\|^2 \qquad 2)\ C\sum_i \max(0, 1 - y_i(w^\intercal x_i + b))$$

The equation used can be split into two.

The first term enables the algorithm to minimize the parameters of W and achieve a greater margin. The second term has to do with hinge loss. This is used to calculate the slack variable for every dataset (Varghese, 2018c).

If any dataset falls within the margin or not on the correct side of it a penalty is given by the hinge loss. W decreases and the margin widens due to the minimization of the first term (Varghese, 2018c).

To reduce hinge loss the margin must be shortened, this is obtained by minimizing the second term. To deal with non-linear kernels we can use Gaussian kernel, polynomial kernel and many more (Varghese, 2018c).

Advantages

- To solve complex problems SVM uses the kernel trick (Varghese, 2018c).
- SVM uses a convex optimization function, because of this global minimum can always be obtained (Varghese, 2018c).
- Higher accuracy is achieved by using hinge loss (Varghese, 2018c).
- By using soft margin constant C outliers can be easily handled (Varghese, 2018c).

Disadvantages

- Sparsity is caused by hinge loss (Varghese, 2018c).
- Hyperparameters and kernels must be carefully tuned to ensure sufficient accuracy (Varghese, 2018c).
- Large datasets require training time.

Hyperparameters

- Set margin constant (c): The penalty over the outliers is caused by the C constant. It is the inverse to the regularization parameter. When the value of C is big. Outliers are given a great penalty and because of this, the hard margin is discovered (Varghese, 2018c).
  When the value of C is small outliers are ignored and the margin begins to become bigger in width (Varghese, 2018c).
- Degree of the polynomial (d): When d = 1 this is the same as a linear kernel when d is a larger value the kernel can tell complex patterns by extending them to a new hyperplane (Varghese, 2018c).
- The width parameter in the Gaussian Kernel(Y): gamma determines the width of the gaussian curve. When gamma increases width also increases (Varghese, 2018c).

SVM vs Random forests

- Random forests can carry out multi-class classification however SVM requires different models to do the same (Varghese, 2018c).
- Random forests can give a probability over the prediction, whereas SVM cannot do this (Varghese, 2018c).
- Random forests handle categorical data better than SVM (Varghese, 2018c).

SVM vs NN

- SVM has a convex optimization but NN may hang in local minima.
- SVM performs better than NN where there is limited training data, and lots of features. NN also requires large training data for good accuracy.
- Multi-class classification needs many models for SVM but NN can do it with a single model.

SVM vs NN

- SVM has a convex optimization but NN may hang in local minima.
- SVM performs better than NN where there is limited training data, and lots of features. NN also requires large training data for good accuracy.
- Multi-class classification needs many models for SVM but NN can do it with a single model.

4.5.4 - Decision Trees

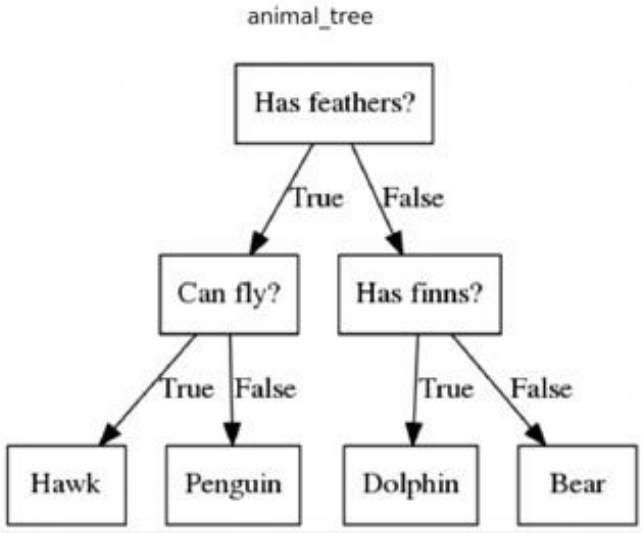


Figure 10 Decision Trees (Towards AI Team, 2020)

Decision Trees is a machine learning model that is used to cover classification and regression problems. A Decision tree can be used to visually represent the task of making decisions during the decision-making process. As shown by **Figure 10,** Decision trees comprise nodes and edges where each node represents a condition used to decide and each edge represents an answer. For example from what can be seen on the right-hand side, this decision tree is trying to decide about an animal, therefore, the root node begins by asking the most definite question which is " Has feathers? " and later goes into asking more questions until a decision on what animal it is, is made. There are two types of decision trees, classification trees that are used to determine a discrete value such as an animal, person, or thing. You next have regression trees that are used to predict continuous values (Prashant Gupta, 2017).

Loss function

The Gini index is used as a metric for classification. It is used to calculate how well mixed the data points are. When this calculation is being done the attribute with the maximum Gini index is used as the condition to evaluate all other data points upon creation of the decision tree. If the Gini score is maximum that the data points are unequally mixed.

ID3 (Iterative Dichotomiser 3 algorithm) is also used to generate a decision tree from a dataset. In ID3, entropy is used to gain information to select the next attribute. Referring to the equation below H(s) represents entropy and IG(s) represents Information gain. Information gain is used to calculate the entropy difference between the child and parent node (Wikipedia Contributors, 2020).

$$H(s) = -\sum P_c \cdot log(P_c)$$

$$IG(s) = H(s) - \sum_t P_t \cdot H(t)$$

Figure 11 Information gain and Entropy Equation (Varghese, 2018)

Hyperparameter

(Varghese, 2018) Decision trees contain many specific values that change the machine learning process.

These are:

- Criterion: The criterion is used to calculate the cost function when deciding what the next node in the tree should be. The criterion which is commonly used is the Gini and entropy values.
- Max Depth: This shows how deep the decision tree should be for example if it should be two or three layers deep.
- Minimum Samples Split: This shows the lowest number of nodes that must be used to split an internal node.
- Minimum samples leaf: At each leaf node samples are needed to be there this value tells us the minimum samples that are required.

Comparison to other models

Decision tree vs Random Forest

- Random Forests are essentially clusters of decision trees and the average vote of a forest is used as the predicted output (Varghese, 2018).

- Decision trees are more likely to cause overfitting compared to the Random Forest model which is less likely to cause overfitting.
- Decision trees are less accurate than Random Forests

Decision Trees vs KNN

- Both are non-parametric which means both models do not assume and rely on any distributions (Varghese, 2018).
- Automatic feature interaction is supported during decision trees where KNN does not support Decision tree (Varghese, 2018).
- A decision tree is faster since KNN has a very expensive time to execute (Varghese, 2018).

Decision Trees vs NN

- It can also be said that both do not find linear solutions, and as a result have interactions between independent variables (Varghese, 2018).
- When large categorical values are apparent in the dataset decision trees perform better than neural networks (Varghese, 2018).
- Due to the logical process's decision trees take when coming to a decision it is easier to use them when you know that you need to come up with an explanation for your decisions compared to that of NN (Varghese, 2018).

Decision Trees vs SVM

- Collinearity is better dealt with using SVM compared to Decision trees (Varghese, 2018).

4.5.5 - Random Forests

Random forests are multiple decisions trees that are used to obtain a better predictive performance, as a result, Random Forests uses ensemble learning to carry out results when making predictions. Random Forests work by each tree predicting a specific class; the class with the most votes that is calculated then becomes the prediction for the Random Forest Model as shown below in **Figure 12**. The way this would work in the real world is to imagine if someone is searching for car insurance on an insurance comparison website. Multiple questions are asked because of the answer being inputted certain insurance providers provide specific quotes these insurance providers represent the various trees.

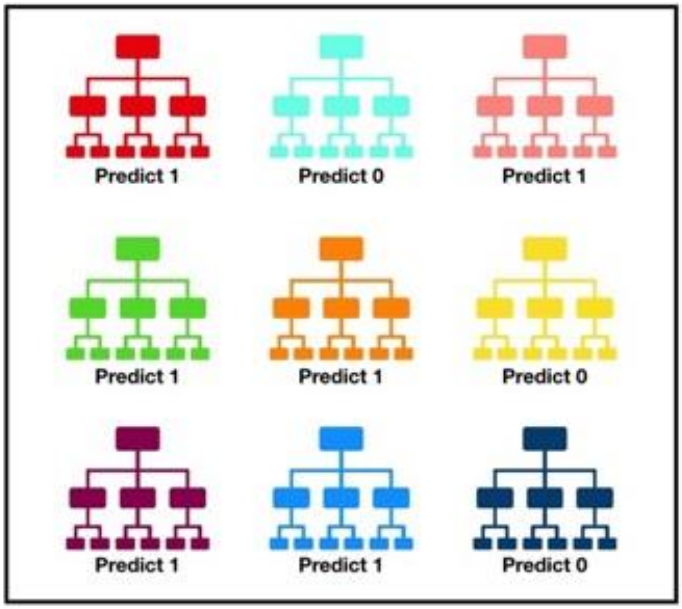


Figure 12 Random Forest Model making a prediction (Yiu, 2019)

Loss function

For the loss function we can use entropy and the Gini score this is described above in my description of the loss function for Decision trees.

Hyperparameters

- N_estimators: These are described as the number of trees within the forest the higher the number of trees the more accurate the models this is because the model has more trees to evaluate and can therefore be more accurate.
- Maximum features: This is the number of features within an individual tree.
- Minimum sample leaf: This is the minimum number of samples required to split an internal node.

Advantages

- Is very accurate and powerful when used correctly (Varghese, 2018a).
- Deals with overfitting well (Varghese, 2018a).
- Can handle implicit feature selection and can work out feature importance well (Varghese, 2018a).

Disadvantages

- Computationally complex and as a result is slower when the trees become large (Varghese, 2018a).
- Does not describe the model well when arriving at a prediction (Varghese, 2018a).

Random Forest vs NN

- Both are very powerful and because of this gives high degrees of accuracy (Varghese, 2018a).
- Both have internal feature interactions and are less explainable (Varghese, 2018a).

- NN needs to be feature scaled whilst Random Forests does not need to be(Varghese, 2018a).

4.5.6 - Neural Networks

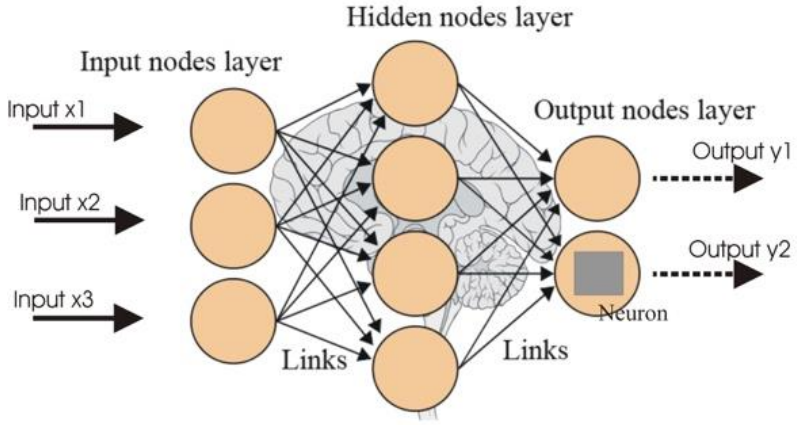


Figure 13 Neural network showing different layers

Neural Networks is a machine learning algorithm that is inspired by the architecture of neurons in the brain. Brain neurons operate by a brain neuron receiving input and firing off an output based on what was inputted. Neural Networks are created from three layers of neurons, the input layer, hidden layer, and output layer (LearnCode.academy, 2018).

Input Layer- This is the layer that is used to bring initial data within the neural network to allow it to be processed by the neural network.

Hidden Layer – This layer lies between the Input and output layer. In this layer, weights are put on the inputs and pass them through an activation function to get the output.

Output Layer – This layer is responsible for producing accurate results from the processes that have been done before. As a result, it provides the ultimate outputs for the algorithm.

Data is inputted within the network and input dimensions are moved into the network. Neural Networks are done by feeding the model thousands of pieces of data in my case I would feed the Neural Network data on what sensor values correspond to certain actions such as walking, running, and sitting down. Processing nodes that are connected are used to process the data. There are several types of Neural networks one of which is a **Feed-Forward** Neural Network. This type of Neural Network does not form a cycle and as a result only goes forward. Each node assigns a weight to its connections. When the Neural Network is activated, each node receives a different data item that carries a different value and multiplies it by its assigned weight. This result is then used to compare to the threshold value if the number that is produced because of the weight and its data value is below the threshold, the data is not passed to the next layer and as a result, can ensure data that does

not meet the benchmark is not passed as a result. During the training process weights and thresholds are continually adjusted to ensure massive bias in results.

Loss function

- **Mean Squared Error –** Neural Networks can use mean squared error. This is calculated using the mean of the squared differences between the target and the predicted values (Shiva Verma, 2019).
- **Binary Cross entropy** – BCE is used for binary classification. BCE requires you to only use one node to classify the data to different classes the output should then be passed through the sigmoid function where it is given an output between 0 and 1 (Shiva Verma, 2019).
- **Categorical Cross entropy** – This is used when you are dealing with a multi-classification task. To use this function there must be a number of output nodes as the classes must be the same. The final layer must then be put through a SoftMax activation resulting in every node to give output between 0 and 1 (Shiva Verma, 2019).

Hyperparameters:

- **The number of hidden layers –** More the hidden the layers the better accuracy up to a certain point. Therefore, it is not worth increasing the number of layers too high as after a certain point the difference will not be significant and will make the algorithm more expensive to run (Alto, 2019).
- **Learning Rate-** If the learning rate is too small compared to the optimum values it will take a much longer time to reach the ideal state. However, if it is way too larger than the optimum value it will overestimate the ideal state (Leonel, 2019).
- **Hidden layer size-** This represents the number of neurons in a Neural Network. (Scikit-learn.org, 2015).

Advantages

- They can learn and model non – linear and complex relationships almost to that of real-life (Jahnavi Mahanta, 2017).
- They can generalise and predict unseen data well (Jahnavi Mahanta, 2017).
- They do not impose restrictions on output variables, as a result, can model data with high volatility and non-constant variance (Jahnavi Mahanta, 2017).

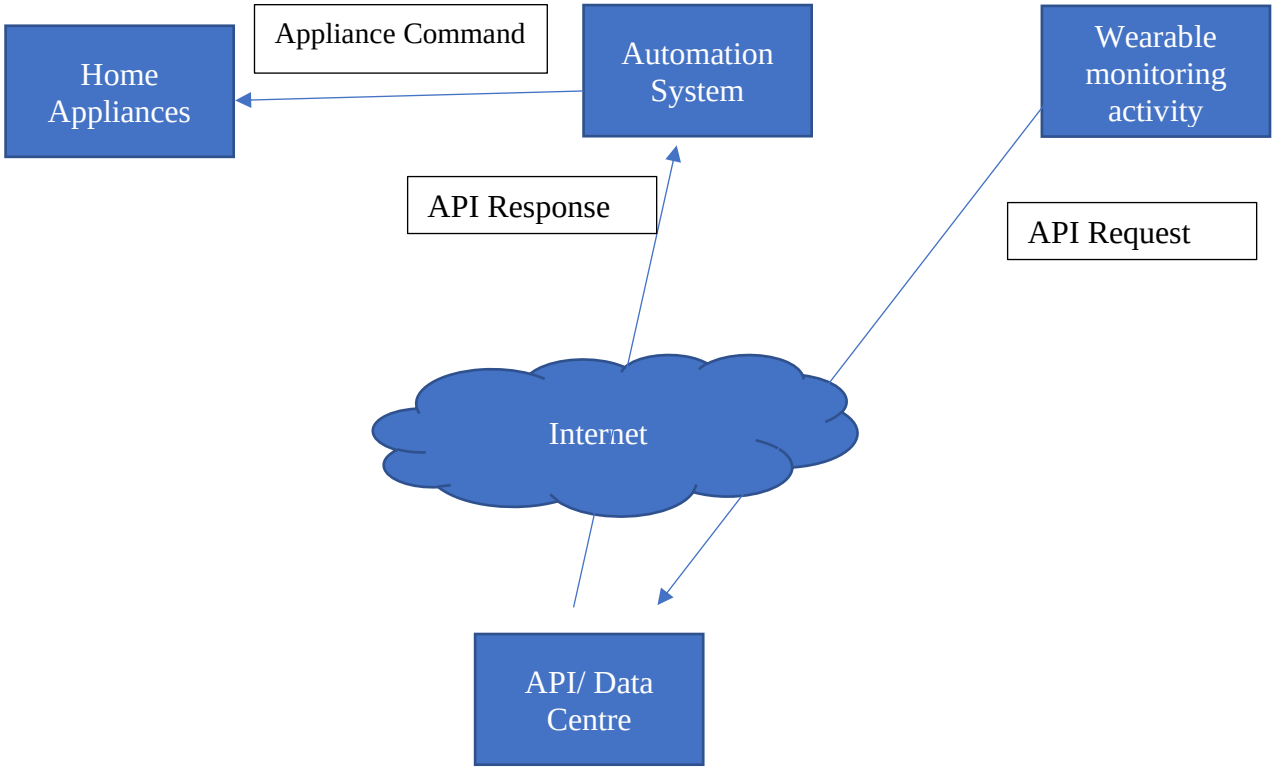
Disadvantages

- They require a large amount of computational power. An example of this is seen in the famous CNN AlexNet for image classification it took six days to train two GPU's.

Overall, the comparisons of different models show some interesting facts about how the models that I will be working on operate. As a result of this, I will look to build upon this knowledge and use my findings to evaluate the hyperparameter tuning carried out on each model later in this project. In the next chapter how, the project is implemented will be discussed alongside the code I used to carry out certain experiments.

4.6 – API Deployment Research

By carrying out research on how to deploy the ML model it has become apparent that we can use FastAPI to build an API that when can predict the activity classes. Fast API is a web framework for building APIs with Python. I gained knowledge on how to use FAST API using a YouTube tutorial (Krish Naik, 2020) as Fast API is a new solution to building APIs rather than the old school way of using flask.



An API is a software intermediary which allows two applications to talk to each other. The reason why I will be looking into creating an API is to enable me to explore how I will be able to deploy my ML model and use it in real life post-graduation. The API will be created to predict human behaviour activities and based on what is predicted transmit messages to home appliances. This can theoretically lead to the automation of user's homes based on what activity is being performed and their daily habits. Above is a diagram that shows how believe this system could possibly work in real life. Whilst it is a quite general diagram further research will be carried out after University to see how it can work in more depth. Due to the lack of time available we will be focussing on building an API using FAST API which will be focussed on in more depth in Chapter 7.

5 – Implementation

In this section we will be discussing how the implementation of this project has been carried out alongside our results.

5.1 What will be done

Considering all research that has been carried out prior to this chapter the Machine Learning Pipeline for the Investigation of different Machine Learning Models on Human Behaviour Activity will be carried out in several ways. First, it is evident that this project is a classification problem this is because we will be classifying whether data fits into a specific category which will be the Activity. In addition to this because the data is labelled this means that we will be using supervised algorithms to match the data to its label. The algorithms used in this project are the algorithms discussed above in the previous chapter K-Nearest Neighbour, Logistic Regression, Support Vector Machine, Decision Tree, Random Forest, and Neural Networks. We're interested to see the impact Random Forest has, as two papers in the Gap analysis stated that Random Forest was the best Algorithm.

Furthermore, as sensors are not available during this project, I will be using an open-source data set from Kaggle more about this dataset can be found in the next section.

I will then carry out the cleaning of the data by viewing how many non-entries of the data there are and looking at its visual representations.

Once this is completed this project will begin by training all models discussed in Chapter 4.5, analysing the accuracies and get performance measures such as Accuracy, F1- Score, Precision and recall. This study will carry out Train Test Split to get the training and testing score and will then carry out K-fold cross- validation to verify results. Next hyperparameter tuning and model evaluation will be carried out to understand how different parameters impact the accuracy of the models. At this point the model that is the best will be deployed as an API that can be used in mobile apps, or websites as a method to deploy the machine learning model for production use.

5.1.1 Resources that will be used

To carry out the implementation of this project effectively as discussed previously the programming language that will be used is Python. In addition to this, the IDE that will be used is PyCharm. PyCharm is a hybrid platform developed by JetBrains as an IDE for Python.

The Dataset that will be used will be discussed in the next chapter.

5.2 – Dataset

The dataset we will be using is called "Human Activity Recognition with Smartphones" this Dataset can be found on Kaggle (UCI Machine Learning, 2012). This data set contains the recordings of 30 participants carrying out activities related to daily living. The activities they carried out were Walking, Walking Upstairs, Walking Downstairs, Sitting, Standing and Laying. This data was captured wearing a smartphone at the waist

5.2.1 – Understanding our Dataset

In this chapter we will discuss how we can begin to understand our data in greater depth.

```
3   import pandas
4
5   # Got the Training/Test Dataset file
6   filename = 'Dataset/trainsaved.csv'
7   filename1 = 'Dataset/test.csv'
8
9   # Creating a DataFrame called DF that
10  df = pandas.read_csv(filename)
11  df1 = pandas.read_csv(filename1)
12  # Printed the columns, head of the da
13  print(df.columns)
14  print(df.head)
15  print(df.shape)
16  df.info()
17  # Printed the columns, head of the da
18  print(df1.columns)
19  print(df1.head)
20  print(df1.shape)
21  df1.info()
```

Figure 14 Code Used to get a general understanding of the Dataset

In line 3 this code begins by importing **Pandas.** Pandas is a Python library that is used for data analysis. Pandas is used because it provides the ability to import data from various formats such as JSON, SQL and Excel. Due to the fact, the Datasets being used are .CSV files which are Comma- Separated Values Files (Pydata.org, 2013)

Next, a Data Frame is created which allows data to be easily manipulated. To read the data file **pandas*.read_csv***(‘….’) could be written but doing this would make the task tedious and stop code from easily being reused in our class. As a result of this, a variable is created that stores the file paths, which can be seen on lines 6 and 7. Next a variable called df and df1 is created which contains the read training and test datasets this can be seen on **line 10 and 11** of **Figure 14**.

The columns of the data sets can be viewed using ***df. columns*** and ***df1.columns*** which can be found on **line 13 and 18** of **Figure 14** doing this gives the column features of the data set which are:

'tBodyAcc-mean()-X' 'tBodyAcc-mean()-Y' 'tBodyAcc-mean()-Z'
'tBodyAcc-std()-X' 'tBodyAcc-std()-Y' 'tBodyAcc-std()-Z'
'tBodyAcc-mad()-X' 'tBodyAcc-mad()-Y' 'tBodyAcc-mad()-Z'
'tBodyAcc-max()-X' 'tBodyAcc-max()-Y' 'tBodyAcc-max()-Z'
'tBodyAcc-min()-X' 'tBodyAcc-min()-Y' 'tBodyAcc-min()-Z' 'tBodyAcc-sma()'
'tBodyAcc-energy()-X' 'tBodyAcc-energy()-Y' 'tBodyAcc-energy()-Z'
'tBodyAcc-iqr()-X' 'tBodyAcc-iqr()-Y' 'tBodyAcc-iqr()-Z'
'tBodyAcc-entropy()-X' 'tBodyAcc-entropy()-Y' 'tBodyAcc-entropy()-Z'
'tBodyAcc-arCoeff()-X,1' 'tBodyAcc-arCoeff()-X,2'
'tBodyAcc-arCoeff()-X,3' 'tBodyAcc-arCoeff()-X,4'
'tBodyAcc-arCoeff()-Y,1' 'tBodyAcc-arCoeff()-Y,2'
'tBodyAcc-arCoeff()-Y,3' 'tBodyAcc-arCoeff()-Y,4'
'tBodyAcc-arCoeff()-Z,1' 'tBodyAcc-arCoeff()-Z,2'

'tBodyAcc-arCoeff()-Z,3' 'tBodyAcc-arCoeff()-Z,4'
'tBodyAcc-correlation()-X,Y' 'tBodyAcc-correlation()-X,Z'
'tBodyAcc-correlation()-Y,Z' 'tGravityAcc-mean()-X'
'tGravityAcc-mean()-Y' 'tGravityAcc-mean()-Z' 'tGravityAcc-std()-X'
'tGravityAcc-std()-Y' 'tGravityAcc-std()-Z' 'tGravityAcc-mad()-X'
'tGravityAcc-mad()-Y' 'tGravityAcc-mad()-Z' 'tGravityAcc-max()-X'
'tGravityAcc-max()-Y' 'tGravityAcc-max()-Z' 'tGravityAcc-min()-X'
'tGravityAcc-min()-Y' 'tGravityAcc-min()-Z' 'tGravityAcc-sma()'
'tGravityAcc-energy()-X' 'tGravityAcc-energy()-Y'
'tGravityAcc-energy()-Z' 'tGravityAcc-iqr()-X' 'tGravityAcc-iqr()-Y'
'tGravityAcc-iqr()-Z' 'tGravityAcc-entropy()-X' 'tGravityAcc-entropy()-Y'
'tGravityAcc-entropy()-Z' 'tGravityAcc-arCoeff()-X,1'
'tGravityAcc-arCoeff()-X,2' 'tGravityAcc-arCoeff()-X,3'
'tGravityAcc-arCoeff()-X,4' 'tGravityAcc-arCoeff()-Y,1'
'tGravityAcc-arCoeff()-Y,2' 'tGravityAcc-arCoeff()-Y,3'
'tGravityAcc-arCoeff()-Y,4' 'tGravityAcc-arCoeff()-Z,1'
'tGravityAcc-arCoeff()-Z,2' 'tGravityAcc-arCoeff()-Z,3'
'tGravityAcc-arCoeff()-Z,4' 'tGravityAcc-correlation()-X,Y'
'tGravityAcc-correlation()-X,Z' 'tGravityAcc-correlation()-Y,Z'
'tBodyAccJerk-mean()-X' 'tBodyAccJerk-mean()-Y' 'tBodyAccJerk-mean()-Z'
'tBodyAccJerk-std()-X' 'tBodyAccJerk-std()-Y' 'tBodyAccJerk-std()-Z'
'tBodyAccJerk-mad()-X' 'tBodyAccJerk-mad()-Y' 'tBodyAccJerk-mad()-Z'
'tBodyAccJerk-max()-X' 'tBodyAccJerk-max()-Y' 'tBodyAccJerk-max()-Z'
'tBodyAccJerk-min()-X' 'tBodyAccJerk-min()-Y' 'tBodyAccJerk-min()-Z'
'tBodyAccJerk-sma()' 'tBodyAccJerk-energy()-X' 'tBodyAccJerk-energy()-Y'
'tBodyAccJerk-energy()-Z' 'tBodyAccJerk-iqr()-X' 'tBodyAccJerk-iqr()-Y'
'tBodyAccJerk-iqr()-Z' 'tBodyAccJerk-entropy()-X'
'tBodyAccJerk-entropy()-Y' 'tBodyAccJerk-entropy()-Z'
'tBodyAccJerk-arCoeff()-X,1' 'tBodyAccJerk-arCoeff()-X,2'
'tBodyAccJerk-arCoeff()-X,3' 'tBodyAccJerk-arCoeff()-X,4'
'tBodyAccJerk-arCoeff()-Y,1' 'tBodyAccJerk-arCoeff()-Y,2'
'tBodyAccJerk-arCoeff()-Y,3' 'tBodyAccJerk-arCoeff()-Y,4'
'tBodyAccJerk-arCoeff()-Z,1' 'tBodyAccJerk-arCoeff()-Z,2'
'tBodyAccJerk-arCoeff()-Z,3' 'tBodyAccJerk-arCoeff()-Z,4'
'tBodyAccJerk-correlation()-X,Y' 'tBodyAccJerk-correlation()-X,Z'
'tBodyAccJerk-correlation()-Y,Z' 'tBodyGyro-mean()-X'
'tBodyGyro-mean()-Y' 'tBodyGyro-mean()-Z' 'tBodyGyro-std()-X'
'tBodyGyro-std()-Y' 'tBodyGyro-std()-Z' 'tBodyGyro-mad()-X'
'tBodyGyro-mad()-Y' 'tBodyGyro-mad()-Z' 'tBodyGyro-max()-X'
'tBodyGyro-max()-Y' 'tBodyGyro-max()-Z' 'tBodyGyro-min()-X'
'tBodyGyro-min()-Y' 'tBodyGyro-min()-Z' 'tBodyGyro-sma()'
'tBodyGyro-energy()-X' 'tBodyGyro-energy()-Y' 'tBodyGyro-energy()-Z'
'tBodyGyro-iqr()-X' 'tBodyGyro-iqr()-Y' 'tBodyGyro-iqr()-Z'
'tBodyGyro-entropy()-X' 'tBodyGyro-entropy()-Y' 'tBodyGyro-entropy()-Z'
'tBodyGyro-arCoeff()-X,1' 'tBodyGyro-arCoeff()-X,2'
'tBodyGyro-arCoeff()-X,3' 'tBodyGyro-arCoeff()-X,4'
'tBodyGyro-arCoeff()-Y,1' 'tBodyGyro-arCoeff()-Y,2'
'tBodyGyro-arCoeff()-Y,3' 'tBodyGyro-arCoeff()-Y,4'
'tBodyGyro-arCoeff()-Z,1' 'tBodyGyro-arCoeff()-Z,2'
'tBodyGyro-arCoeff()-Z,3' 'tBodyGyro-arCoeff()-Z,4'
'tBodyGyro-correlation()-X,Y' 'tBodyGyro-correlation()-X,Z'
'tBodyGyro-correlation()-Y,Z' 'tBodyGyroJerk-mean()-X'
'tBodyGyroJerk-mean()-Y' 'tBodyGyroJerk-mean()-Z' 'tBodyGyroJerk-std()-X'
'tBodyGyroJerk-std()-Y' 'tBodyGyroJerk-std()-Z' 'tBodyGyroJerk-mad()-X'
'tBodyGyroJerk-mad()-Y' 'tBodyGyroJerk-mad()-Z' 'tBodyGyroJerk-max()-X'
'tBodyGyroJerk-max()-Y' 'tBodyGyroJerk-max()-Z' 'tBodyGyroJerk-min()-X'
'tBodyGyroJerk-min()-Y' 'tBodyGyroJerk-min()-Z' 'tBodyGyroJerk-sma()'
'tBodyGyroJerk-energy()-X' 'tBodyGyroJerk-energy()-Y'
'tBodyGyroJerk-energy()-Z' 'tBodyGyroJerk-iqr()-X'
'tBodyGyroJerk-iqr()-Y' 'tBodyGyroJerk-iqr()-Z'
'tBodyGyroJerk-entropy()-X' 'tBodyGyroJerk-entropy()-Y'
'tBodyGyroJerk-entropy()-Z' 'tBodyGyroJerk-arCoeff()-X,1'
'tBodyGyroJerk-arCoeff()-X,2' 'tBodyGyroJerk-arCoeff()-X,3'
'tBodyGyroJerk-arCoeff()-X,4' 'tBodyGyroJerk-arCoeff()-Y,1'
'tBodyGyroJerk-arCoeff()-Y,2' 'tBodyGyroJerk-arCoeff()-Y,3'
'tBodyGyroJerk-arCoeff()-Y,4' 'tBodyGyroJerk-arCoeff()-Z,1'
'tBodyGyroJerk-arCoeff()-Z,2' 'tBodyGyroJerk-arCoeff()-Z,3'
'tBodyGyroJerk-arCoeff()-Z,4' 'tBodyGyroJerk-correlation()-X,Y'
'tBodyGyroJerk-correlation()-X,Z' 'tBodyGyroJerk-correlation()-Y,Z'
'tBodyAccMag-mean()' 'tBodyAccMag-std()' 'tBodyAccMag-mad()'
'tBodyAccMag-max()' 'tBodyAccMag-min()' 'tBodyAccMag-sma()'

'tBodyAccMag-energy()' 'tBodyAccMag-iqr()' 'tBodyAccMag-entropy()'
'tBodyAccMag-arCoeff()1' 'tBodyAccMag-arCoeff()2'
'tBodyAccMag-arCoeff()3' 'tBodyAccMag-arCoeff()4' 'tGravityAccMag-mean()'
'tGravityAccMag-std()' 'tGravityAccMag-mad()' 'tGravityAccMag-max()'
'tGravityAccMag-min()' 'tGravityAccMag-sma()' 'tGravityAccMag-energy()'
'tGravityAccMag-iqr()' 'tGravityAccMag-entropy()'
'tGravityAccMag-arCoeff()1' 'tGravityAccMag-arCoeff()2'
'tGravityAccMag-arCoeff()3' 'tGravityAccMag-arCoeff()4'
'tBodyAccJerkMag-mean()' 'tBodyAccJerkMag-std()' 'tBodyAccJerkMag-mad()'
'tBodyAccJerkMag-max()' 'tBodyAccJerkMag-min()' 'tBodyAccJerkMag-sma()'
'tBodyAccJerkMag-energy()' 'tBodyAccJerkMag-iqr()'
'tBodyAccJerkMag-entropy()' 'tBodyAccJerkMag-arCoeff()1'
'tBodyAccJerkMag-arCoeff()2' 'tBodyAccJerkMag-arCoeff()3'
'tBodyAccJerkMag-arCoeff()4' 'tBodyGyroMag-mean()' 'tBodyGyroMag-std()'
'tBodyGyroMag-mad()' 'tBodyGyroMag-max()' 'tBodyGyroMag-min()'
'tBodyGyroMag-sma()' 'tBodyGyroMag-energy()' 'tBodyGyroMag-iqr()'
'tBodyGyroMag-entropy()' 'tBodyGyroMag-arCoeff()1'
'tBodyGyroMag-arCoeff()2' 'tBodyGyroMag-arCoeff()3'
'tBodyGyroMag-arCoeff()4' 'tBodyGyroJerkMag-mean()'
'tBodyGyroJerkMag-std()' 'tBodyGyroJerkMag-mad()'
'tBodyGyroJerkMag-max()' 'tBodyGyroJerkMag-min()'
'tBodyGyroJerkMag-sma()' 'tBodyGyroJerkMag-energy()'
'tBodyGyroJerkMag-iqr()' 'tBodyGyroJerkMag-entropy()'
'tBodyGyroJerkMag-arCoeff()1' 'tBodyGyroJerkMag-arCoeff()2'
'tBodyGyroJerkMag-arCoeff()3' 'tBodyGyroJerkMag-arCoeff()4'
'fBodyAcc-mean()-X' 'fBodyAcc-mean()-Y' 'fBodyAcc-mean()-Z'
'fBodyAcc-std()-X' 'fBodyAcc-std()-Y' 'fBodyAcc-std()-Z'
'fBodyAcc-mad()-X' 'fBodyAcc-mad()-Y' 'fBodyAcc-mad()-Z'
'fBodyAcc-max()-X' 'fBodyAcc-max()-Y' 'fBodyAcc-max()-Z'
'fBodyAcc-min()-X' 'fBodyAcc-min()-Y' 'fBodyAcc-min()-Z' 'fBodyAcc-sma()'
'fBodyAcc-energy()-X' 'fBodyAcc-energy()-Y' 'fBodyAcc-energy()-Z'
'fBodyAcc-iqr()-X' 'fBodyAcc-iqr()-Y' 'fBodyAcc-iqr()-Z'
'fBodyAcc-entropy()-X' 'fBodyAcc-entropy()-Y' 'fBodyAcc-entropy()-Z'
'fBodyAcc-maxInds-X' 'fBodyAcc-maxInds-Y' 'fBodyAcc-maxInds-Z'
'fBodyAcc-meanFreq()-X' 'fBodyAcc-meanFreq()-Y' 'fBodyAcc-meanFreq()-Z'
'fBodyAcc-skewness()-X' 'fBodyAcc-kurtosis()-X' 'fBodyAcc-skewness()-Y'
'fBodyAcc-kurtosis()-Y' 'fBodyAcc-skewness()-Z' 'fBodyAcc-kurtosis()-Z'
'fBodyAcc-bandsEnergy()-1,8' 'fBodyAcc-bandsEnergy()-9,16'
'fBodyAcc-bandsEnergy()-17,24' 'fBodyAcc-bandsEnergy()-25,32'
'fBodyAcc-bandsEnergy()-33,40' 'fBodyAcc-bandsEnergy()-41,48'
'fBodyAcc-bandsEnergy()-49,56' 'fBodyAcc-bandsEnergy()-57,64'
'fBodyAcc-bandsEnergy()-1,16' 'fBodyAcc-bandsEnergy()-17,32'
'fBodyAcc-bandsEnergy()-33,48' 'fBodyAcc-bandsEnergy()-49,64'
'fBodyAcc-bandsEnergy()-1,24' 'fBodyAcc-bandsEnergy()-25,48'
'fBodyAcc-bandsEnergy()-1,8.1' 'fBodyAcc-bandsEnergy()-9,16.1'
'fBodyAcc-bandsEnergy()-17,24.1' 'fBodyAcc-bandsEnergy()-25,32.1'
'fBodyAcc-bandsEnergy()-33,40.1' 'fBodyAcc-bandsEnergy()-41,48.1'
'fBodyAcc-bandsEnergy()-49,56.1' 'fBodyAcc-bandsEnergy()-57,64.1'
'fBodyAcc-bandsEnergy()-1,16.1' 'fBodyAcc-bandsEnergy()-17,32.1'
'fBodyAcc-bandsEnergy()-33,48.1' 'fBodyAcc-bandsEnergy()-49,64.1'
'fBodyAcc-bandsEnergy()-1,24.1' 'fBodyAcc-bandsEnergy()-25,48.1'
'fBodyAcc-bandsEnergy()-1,8.2' 'fBodyAcc-bandsEnergy()-9,16.2'
'fBodyAcc-bandsEnergy()-17,24.2' 'fBodyAcc-bandsEnergy()-25,32.2'
'fBodyAcc-bandsEnergy()-33,40.2' 'fBodyAcc-bandsEnergy()-41,48.2'
'fBodyAcc-bandsEnergy()-49,56.2' 'fBodyAcc-bandsEnergy()-57,64.2'
'fBodyAcc-bandsEnergy()-1,16.2' 'fBodyAcc-bandsEnergy()-17,32.2'
'fBodyAcc-bandsEnergy()-33,48.2' 'fBodyAcc-bandsEnergy()-49,64.2'
'fBodyAcc-bandsEnergy()-1,24.2' 'fBodyAcc-bandsEnergy()-25,48.2'
'fBodyAccJerk-mean()-X' 'fBodyAccJerk-mean()-Y' 'fBodyAccJerk-mean()-Z'
'fBodyAccJerk-std()-X' 'fBodyAccJerk-std()-Y' 'fBodyAccJerk-std()-Z'
'fBodyAccJerk-mad()-X' 'fBodyAccJerk-mad()-Y' 'fBodyAccJerk-mad()-Z'
'fBodyAccJerk-max()-X' 'fBodyAccJerk-max()-Y' 'fBodyAccJerk-max()-Z'
'fBodyAccJerk-min()-X' 'fBodyAccJerk-min()-Y' 'fBodyAccJerk-min()-Z'
'fBodyAccJerk-sma()' 'fBodyAccJerk-energy()-X' 'fBodyAccJerk-energy()-Y'
'fBodyAccJerk-energy()-Z' 'fBodyAccJerk-iqr()-X' 'fBodyAccJerk-iqr()-Y'
'fBodyAccJerk-iqr()-Z' 'fBodyAccJerk-entropy()-X'
'fBodyAccJerk-entropy()-Y' 'fBodyAccJerk-entropy()-Z'
'fBodyAccJerk-maxInds-X' 'fBodyAccJerk-maxInds-Y'
'fBodyAccJerk-maxInds-Z' 'fBodyAccJerk-meanFreq()-X'
'fBodyAccJerk-meanFreq()-Y' 'fBodyAccJerk-meanFreq()-Z'
'fBodyAccJerk-skewness()-X' 'fBodyAccJerk-kurtosis()-X'

'fBodyAccJerk-skewness()-Y' 'fBodyAccJerk-kurtosis()-Y'
'fBodyAccJerk-skewness()-Z' 'fBodyAccJerk-kurtosis()-Z'
'fBodyAccJerk-bandsEnergy()-1,8' 'fBodyAccJerk-bandsEnergy()-9,16'
'fBodyAccJerk-bandsEnergy()-17,24' 'fBodyAccJerk-bandsEnergy()-25,32'
'fBodyAccJerk-bandsEnergy()-33,40' 'fBodyAccJerk-bandsEnergy()-41,48'
'fBodyAccJerk-bandsEnergy()-49,56' 'fBodyAccJerk-bandsEnergy()-57,64'
'fBodyAccJerk-bandsEnergy()-1,16' 'fBodyAccJerk-bandsEnergy()-17,32'
'fBodyAccJerk-bandsEnergy()-33,48' 'fBodyAccJerk-bandsEnergy()-49,64'
'fBodyAccJerk-bandsEnergy()-1,24' 'fBodyAccJerk-bandsEnergy()-25,48'
'fBodyAccJerk-bandsEnergy()-1,8.1' 'fBodyAccJerk-bandsEnergy()-9,16.1'
'fBodyAccJerk-bandsEnergy()-17,24.1' 'fBodyAccJerk-bandsEnergy()-25,32.1'
'fBodyAccJerk-bandsEnergy()-33,40.1' 'fBodyAccJerk-bandsEnergy()-41,48.1'
'fBodyAccJerk-bandsEnergy()-49,56.1' 'fBodyAccJerk-bandsEnergy()-57,64.1'
'fBodyAccJerk-bandsEnergy()-1,16.1' 'fBodyAccJerk-bandsEnergy()-17,32.1'
'fBodyAccJerk-bandsEnergy()-33,48.1' 'fBodyAccJerk-bandsEnergy()-49,64.1'
'fBodyAccJerk-bandsEnergy()-1,24.1' 'fBodyAccJerk-bandsEnergy()-25,48.1'
'fBodyAccJerk-bandsEnergy()-1,8.2' 'fBodyAccJerk-bandsEnergy()-9,16.2'
'fBodyAccJerk-bandsEnergy()-17,24.2' 'fBodyAccJerk-bandsEnergy()-25,32.2'
'fBodyAccJerk-bandsEnergy()-33,40.2' 'fBodyAccJerk-bandsEnergy()-41,48.2'
'fBodyAccJerk-bandsEnergy()-49,56.2' 'fBodyAccJerk-bandsEnergy()-57,64.2'
'fBodyAccJerk-bandsEnergy()-1,16.2' 'fBodyAccJerk-bandsEnergy()-17,32.2'
'fBodyAccJerk-bandsEnergy()-33,48.2' 'fBodyAccJerk-bandsEnergy()-49,64.2'
'fBodyAccJerk-bandsEnergy()-1,24.2' 'fBodyAccJerk-bandsEnergy()-25,48.2'
'fBodyGyro-mean()-X' 'fBodyGyro-mean()-Y' 'fBodyGyro-mean()-Z'
'fBodyGyro-std()-X' 'fBodyGyro-std()-Y' 'fBodyGyro-std()-Z'
'fBodyGyro-mad()-X' 'fBodyGyro-mad()-Y' 'fBodyGyro-mad()-Z'
'fBodyGyro-max()-X' 'fBodyGyro-max()-Y' 'fBodyGyro-max()-Z'
'fBodyGyro-min()-X' 'fBodyGyro-min()-Y' 'fBodyGyro-min()-Z'
'fBodyGyro-sma()' 'fBodyGyro-energy()-X' 'fBodyGyro-energy()-Y'
'fBodyGyro-energy()-Z' 'fBodyGyro-iqr()-X' 'fBodyGyro-iqr()-Y'
'fBodyGyro-iqr()-Z' 'fBodyGyro-entropy()-X' 'fBodyGyro-entropy()-Y'
'fBodyGyro-entropy()-Z' 'fBodyGyro-maxInds-X' 'fBodyGyro-maxInds-Y'
'fBodyGyro-maxInds-Z' 'fBodyGyro-meanFreq()-X' 'fBodyGyro-meanFreq()-Y'
'fBodyGyro-meanFreq()-Z' 'fBodyGyro-skewness()-X'
'fBodyGyro-kurtosis()-X' 'fBodyGyro-skewness()-Y'
'fBodyGyro-kurtosis()-Y' 'fBodyGyro-skewness()-Z'
'fBodyGyro-kurtosis()-Z' 'fBodyGyro-bandsEnergy()-1,8'
'fBodyGyro-bandsEnergy()-9,16' 'fBodyGyro-bandsEnergy()-17,24'
'fBodyGyro-bandsEnergy()-25,32' 'fBodyGyro-bandsEnergy()-33,40'
'fBodyGyro-bandsEnergy()-41,48' 'fBodyGyro-bandsEnergy()-49,56'
'fBodyGyro-bandsEnergy()-57,64' 'fBodyGyro-bandsEnergy()-1,16'
'fBodyGyro-bandsEnergy()-17,32' 'fBodyGyro-bandsEnergy()-33,48'
'fBodyGyro-bandsEnergy()-49,64' 'fBodyGyro-bandsEnergy()-1,24'
'fBodyGyro-bandsEnergy()-25,48' 'fBodyGyro-bandsEnergy()-1,8.1'
'fBodyGyro-bandsEnergy()-9,16.1' 'fBodyGyro-bandsEnergy()-17,24.1'
'fBodyGyro-bandsEnergy()-25,32.1' 'fBodyGyro-bandsEnergy()-33,40.1'
'fBodyGyro-bandsEnergy()-41,48.1' 'fBodyGyro-bandsEnergy()-49,56.1'
'fBodyGyro-bandsEnergy()-57,64.1' 'fBodyGyro-bandsEnergy()-1,16.1'
'fBodyGyro-bandsEnergy()-17,32.1' 'fBodyGyro-bandsEnergy()-33,48.1'
'fBodyGyro-bandsEnergy()-49,64.1' 'fBodyGyro-bandsEnergy()-1,24.1'
'fBodyGyro-bandsEnergy()-25,48.1' 'fBodyGyro-bandsEnergy()-1,8.2'
'fBodyGyro-bandsEnergy()-9,16.2' 'fBodyGyro-bandsEnergy()-17,24.2'
'fBodyGyro-bandsEnergy()-25,32.2' 'fBodyGyro-bandsEnergy()-33,40.2'
'fBodyGyro-bandsEnergy()-41,48.2' 'fBodyGyro-bandsEnergy()-49,56.2'
'fBodyGyro-bandsEnergy()-57,64.2' 'fBodyGyro-bandsEnergy()-1,16.2'
'fBodyGyro-bandsEnergy()-17,32.2' 'fBodyGyro-bandsEnergy()-33,48.2'
'fBodyGyro-bandsEnergy()-49,64.2' 'fBodyGyro-bandsEnergy()-1,24.2'
'fBodyGyro-bandsEnergy()-25,48.2' 'fBodyAccMag-mean()'
'fBodyAccMag-std()' 'fBodyAccMag-mad()' 'fBodyAccMag-max()'
'fBodyAccMag-min()' 'fBodyAccMag-sma()' 'fBodyAccMag-energy()'
'fBodyAccMag-iqr()' 'fBodyAccMag-entropy()' 'fBodyAccMag-maxInds'
'fBodyAccMag-meanFreq()' 'fBodyAccMag-skewness()'
'fBodyAccMag-kurtosis()' 'fBodyBodyAccJerkMag-mean()'
'fBodyBodyAccJerkMag-std()' 'fBodyBodyAccJerkMag-mad()'
'fBodyBodyAccJerkMag-max()' 'fBodyBodyAccJerkMag-min()'
'fBodyBodyAccJerkMag-sma()' 'fBodyBodyAccJerkMag-energy()'
'fBodyBodyAccJerkMag-iqr()' 'fBodyBodyAccJerkMag-entropy()'
'fBodyBodyAccJerkMag-maxInds' 'fBodyBodyAccJerkMag-meanFreq()'
'fBodyBodyAccJerkMag-skewness()' 'fBodyBodyAccJerkMag-kurtosis()'
'fBodyBodyGyroMag-mean()' 'fBodyBodyGyroMag-std()'
'fBodyBodyGyroMag-mad()' 'fBodyBodyGyroMag-max()'

```
'fBodyBodyGyroMag-min()' 'fBodyBodyGyroMag-sma()'
'fBodyBodyGyroMag-energy()' 'fBodyBodyGyroMag-iqr()'
'fBodyBodyGyroMag-entropy()' 'fBodyBodyGyroMag-maxInds'
'fBodyBodyGyroMag-meanFreq()' 'fBodyBodyGyroMag-skewness()'
'fBodyBodyGyroMag-kurtosis()' 'fBodyBodyGyroJerkMag-mean()'
'fBodyBodyGyroJerkMag-std()' 'fBodyBodyGyroJerkMag-mad()'
'fBodyBodyGyroJerkMag-max()' 'fBodyBodyGyroJerkMag-min()'
'fBodyBodyGyroJerkMag-sma()' 'fBodyBodyGyroJerkMag-energy()'
'fBodyBodyGyroJerkMag-iqr()' 'fBodyBodyGyroJerkMag-entropy()'
'fBodyBodyGyroJerkMag-maxInds' 'fBodyBodyGyroJerkMag-meanFreq()'
'fBodyBodyGyroJerkMag-skewness()' 'fBodyBodyGyroJerkMag-kurtosis()'
'angle(tBodyAccMean,gravity)' 'angle(tBodyAccJerkMean),gravityMean)'
'angle(tBodyGyroMean,gravityMean)' 'angle(tBodyGyroJerkMean,gravityMean)'
'angle(X,gravityMean)' 'angle(Y,gravityMean)' 'angle(Z,gravityMean)'
'subject' 'Activity'
```

From the above results it is apparent these datasets contain an extremely large number of features and columns and because of this means reducing dimensions of the data set when visualising the data will have to be considered to understand the relationship between the data instances.

Looking at the head of each dataset which is **on line 14** and **19** of **Figure 14.** It is possible to look at the head of our dataset to see if the data that is in each column has the correct type in it. Doing this verified that the data in it was correct as it contained the correct data (Pydata.org, 2021).

Next we can use

```
df.info()
```

The .info () method on the data frame gives us a concise summary of the Data Frame.

Doing this for the training data gives the following results:

RangeIndex: 7352 entries, 0 to 7351
Columns: 563 entries, tBodyAcc-mean()-X to Activity
dtypes: float64(561), int64(1), object(1)
memory usage: 31.6+ MB

This means that the Training set contains 7352 data entries, the dataset contains 563 columns where 561 are direct features of the Activity of the participants, one is the participant's ID which is of type 64-bit integer for the 30 participants, and one is the feature column for the Activity label that responds to the 561 features. This data takes 31.6 MB of in-memory storage.

The information on the Test data is:

RangeIndex: 2947 entries, 0 to 2946
Columns: 563 entries, tBodyAcc-mean()-X to Activity
dtypes: float64(561), int64(1), object(1)
memory usage: 12.7+ MB

This means that the Test set contains 2947 data entries, the dataset contains 563 columns where 561 are direct features of the Activity of the participants, one is the participant's ID

which is of type 64-bit integer for the 30 participants, and one is the feature column for the Activity label that responds to the 561 features. This data takes 12.7 MB of in-memory storage.

From this chapter, it is evident that it is possible to gain an understanding of the dataset. In addition to this, the column features show all the information on the X-axis, Y-axis, and Z-axis sensors. The X-axis in sensors is used to indicate sideways or horizontal movement. The Y-axis is used to indicate upwards or downwards movement and the Z-axis is used to indicate forward or backward movement.

In the next chapter, greater in-depth understanding of what activities the dataset is made out of will be explored doing this enables us to see if the dataset is skewed to a specific activity or not.

4.1.2 – Getting Descriptive Statistics on our classifiers

Here I will discuss how we can get an understanding of our categorical data and know how much of a specific category is within our data. Doing this allows us to understand the distribution of the dataset as these are the values that models will be predicting and if the distribution is not fair this can cause us not having the ability to train the model well which causes the model to be unable to detect activities well (Cogito, 2019).

```
6   #region Loading Data
7   filename = 'Dataset/trainsaved.csv'
8   filename1 = 'Dataset/test.csv'
9   df = pandas.read_csv(filename)
10  df1 = pandas.read_csv(filename1)
```

Figure 15 Loading in our data

First, we begin by loading in our data. This is done using the code shown in **Figure 15**.

```
25  #region Used to calculate number of unique values alongside th
26  print(df.Activity.value_counts())
27  print((df.Activity.value_counts()/df.Activity.count())*100)
28  #endregion¢
```

Figure 16 Getting the total of a specific Behaviour Activity and its percentage for Trainset

```
30  #region Used to calculate number of unique values alongside thei
31  print(df1.Activity.value_counts())
32  print((df1.Activity.value_counts()/df1.Activity.count())*100)
33  #endregion¢
```

Figure 17 Getting the total of a specific Behaviour Activity and its percentage for Test set

To get the total of a specific activity we can use the ***.value_counts()*** method to count the number of unique values in the Activity column. Then divide this number by the total number of values in the Activity column and then multiply it by 100 which gives us each

unique value as a percentage. Doing this for the training set shown in **Figure 16** and the test set in **Figure 17** enables us to calculate this data for both data sets.

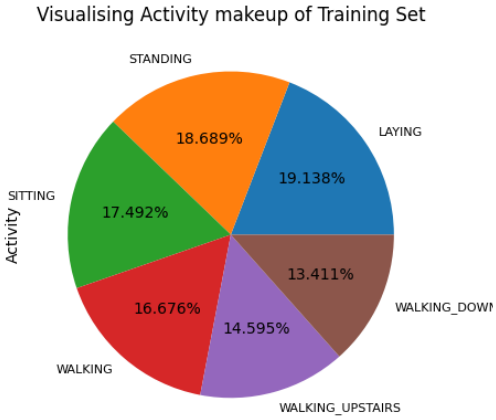


Figure 18 Pie chart showing how many of a specific Activity make up the Training set

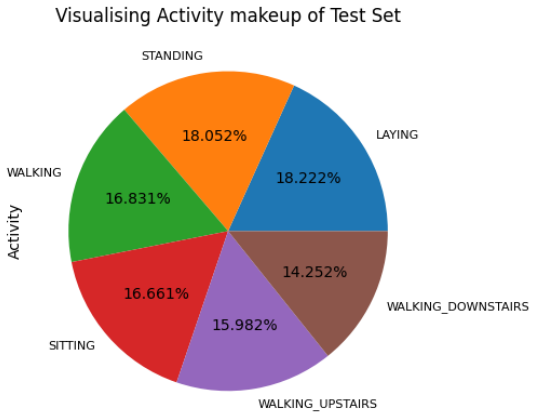


Figure 19 Pie chart showing how many of a specific Activity make up the Test set

Doing this for the 7,352 Training set entries with 563 features show us that. There are 1407 (19.137650%) data entries labelled Laying; There are 1374 (18.688792) data entries labelled Standing; 1226 (16.675734%) data entries labelled Walking; 1286 (17.491839%) data entries labelled Sitting; 1073 (14.594668%) data entries labelled Walking upstairs and 986 (13.411317%) of data labelled Walking Downstairs. A visual representation of this can be seen in **Figure 18**.

Doing this for the 2,947 Test set entries with 563 features show us that there are 537 (18.221921%) data entries labelled Laying. There are 532(18.052257) data entries labelled Standing; 496 (16.830675%) data entries labelled Walking; 491 (16.661011%) data entries labelled Sitting; 471(15.982355%) data entries labelled Walking upstairs and 420 (14.251781%) of data labelled Walking Downstairs. A visual representation of this can be seen in **Figure 19**.

To conclude this Chapter, it can be seen there is a quite fair distribution of the activities and as a result, the ML model that will be developed will be able to give a relatively fair prediction.

In addition to this as seen in the chapter before it can be said that the feature columns of the dataset contain characters that could later be a problem for us. This is because it contains characters that could be invalid in Python and cause issues in the creation of the API as a result in the next chapter, we will look to rectify. We will also check if the dataset contains any missing values and carry out some more data visualisation to see the distinct similarities and differences of the different activities.

5.3 – Data Pre-processing

Here the Data Pre-processing stage as discussed in chapter 4.2.2 will occur to carry this out we will check for any missing values, change the names of the columns to that of something better and simpler.

The processes carried out in this chapter was done for both the training and test set.

5.3.1 – Checking for null values

As the data is already gathered and analysed it is now important to prepare the data, this can be done by first handling any missing data. To do this I have decided to run the code below. Doing this enables me to see how many columns contain null values.

```
8    print(df.isna().sum())
```

Figure 20 Code used to see if any columns in the data frame contain null values

```
tBodyAcc-mean()-X        0
tBodyAcc-mean()-Y        0
tBodyAcc-mean()-Z        0
tBodyAcc-std()-X         0
tBodyAcc-std()-Y         0
                        ..
angle(X,gravityMean)     0
angle(Y,gravityMean)     0
angle(Z,gravityMean)     0
subject                  0
Activity                 0
Length: 563, dtype: int64
```

Figure 21 Result of figure 20

Doing this gives the result shown in figure 19, therefore, showing that there are no null values and therefore no methods to fix null values need to be carried out.

5.3.2 – Changing column names

As the dataset contains columns that have characters that if put into the IDE can cause python warnings and errors, we should now change the column names to something more appropriate.

```
3    from sklearn import preprocessing
```

Figure 22 importing pre-processing package from sklearn

To do this alongside using **Pandas** like used in chapter 5.2.1 we must use the **pre-processing** library from **Sklearn. Sklearn** is a machine learning library that is used to carry out various algorithms. The pre-processing package allows us to change data into a better representation.

To change the column labels, we use the code in **Figure 23.**

```
10  # Removing invalid characters
11  columns = df.columns
12  columns = columns.str.replace('[-]', '')
13  columns = columns.str.replace('[()]','')
14  columns = columns.str.replace('[,]','')
15  columns = columns.str.replace('[.]','')
16  df.columns = columns
17  print(df.head(1))
18  # Savng the new changed datatframe as a .CSV file
19  df.to_csv(r'Dataset/Training Changed.csv')
```

Figure 23 Code used to change characters

Here in **Figure 23**, a variable called **columns** is created which stores all the data frames columns. After this a new variable called **columns** is created for each character we would like to replace and replaces it with space. Each line after line 13 gets the changed columns and searches to find the next character that is needed to be changed. In total 4 changes are carried out to remove the invalid characters. On **line 16** the completed changed array of column names is assigned back to the columns on the data frame.

The changed data frame is then saved as a new Data frame.

5.3.3 – Assigning the labels as numbers

As some models can only be used to classify numeric values, label encoding will be used to replace the activity labels as numbers. Label Encoding is where we convert labels into numeric form to allow them to be easily read by a machine (GeeksforGeeks, 2018) to do this we use the code in **Figure 24**.

```
23  # Label encoding our Activities to remove their string values
24  print("***** Unique Values *****")
25  print(df.Activity.unique())
26  le = preprocessing.LabelEncoder()
27  df['Activity'] = le.fit_transform(df['Activity'])
28  print(df.Activity)
29  print("***** Unique Values Encoded *****")
30  print(df.Activity.unique())
```

Figure 24 Label Encoding Activities

Label Encoding the activities gives the below result shown in **Figure 26**.

```
***** Unique Values *****
['STANDING' 'SITTING' 'LAYING' 'WALKING' 'WALKING_DOWNSTAIRS'
 'WALKING_UPSTAIRS']
```

Figure 25 Activity Labels before Label Encoding

```
***** Unique Values Encoded *****
[2 1 0 3 4 5]
```

Figure 26 Activity Labels after Label Encoding

We can now see that the data has been encoded successfully. Laying is now 0. Sitting is now 1. Standing is now 2. Walking is now 3. Walking Downstairs is now 4 and Walking Upstairs is now 5.

Now that labels have been encoded, when carrying out our training and testing we should not face any problems as they will be able to be easily read by the model.

As we have now ensured there are no missing values, columns contain no characters that can later be a problem for the models and the labels for activities are correctly encoded we have cleaned our data. In the next chapter we will discuss how we visualise data to to pick out patterns.

5.4 – Data Visualization

According to Kayur Patel, Kanit Wongsuphasawat of Apple Inc and other researchers, data visualization is important in enabling us to know more about our data and how we can reach our goal (Acm.org, 2020). Therefore, this section will explore how data visualization is carried out and what conclusions we can make from it.

Due to the dataset having many data features, it is evident that the training set has high dimensionality and if plotted correlations will be hard to see. As a result of this if the dimensionality is reduced and plotted it is clearer to see correlations between the different activities.

5.4.1 – Principal Component Analysis

Principal Component Analysis otherwise known as PCA is used to reduce the dimensionality of a data set and show the correlation between different features. It does this by first identifying a hyperplane that exists closest to the data and then projects it onto it (Aurélien Géron, 2017).

```
1   # Import pandas which allows the importing of
2   # data from different sources such as JSON, CSV also allows manipulation of data
3   import pandas as pd
4    # Using Standard Scaler normalizes data individually so that it has mean 0 and SD 1
5    from sklearn.preprocessing import StandardScaler
6    # this library stored as plt allows us to plot figures easily
7    from matplotlib import pyplot as plt
8    # PCA is used to reduce dimensionality
9    from sklearn.decomposition import PCA
10   #Numpy allows mathematival operations to be easily done
11  import numpy as np
```

Figure 27 Imported libraries to carry out PCA

As discussed before **Pandas** is used to manipulate data from various sources such as. JSON and .CSV. **StandardScaler** from **sklearn.preprocessing** is used to transform data such that our distribution has a mean value of 0 and a Standard Deviation value of 1. Standardizing the values before we carry out PCA ensures the magnitude of variance for some features does not dominate other features in the dataset. This is because during PCA the maximal amount of variance is kept. After this **pyplot** is imported enabling us to plot the PCA figure. After this **PCA** is imported from the **sklearn.decomposition** library enabling us to carry out the PCA algorithm. Finally, **numpy** is imported enable mathematical operations to be easily done.

```
17  # Removing Activity column from dataset and storing it in variable labels
18  labels = df.pop('Activity')
19  no_labels = labels.value_counts()
20
21  # scale data by assigning StandardScalar method to variable called scaler and applying it on the data
22  scaler = StandardScaler()
23  scaler.fit(df)
24  scaled_data = scaler.transform(df)
25  # carry out PCA
26  pca = PCA(n_components=2)
27  pca.fit(scaled_data)
```

Figure 28 Removing Activity Column, Scaling and fitting PCA

After importing scaling can begin, next PCA can occur on the dataset and the indexes of the PCA values are matched to their labels. To begin doing this the data is loaded this is done the same way as shown in **Figure 15.** Next **df.pop is used** on the Activity column and assigned to a variable called **labels** as shown on **line 18** of **Figure 28**. Doing this preserves their Label value and removes the column and assigns it to the variable **labels** ensuring it can be used later without removing it from the dataset.
Next, the data is scaled . To do this a variable called a scaler is created and assigned to it the method **StandardScaler().** After this the scaler is fit to the data frame using **scaler.fit(df).** Next we must transform the dataset using **scaler.tranform(df)** and assigning it to the variable called **scaled_data** . Next PCA is carried out to do this we create a variable called **pca** and give it **PCA** with **n_components** being 2 meaning that the number of components it keeps is 2. We then fit **pca** to the scaled data which is held in **scaled_data.**

```
29  # We are looking at where the data for certain activities exist and if they do we are
30  # getting their indexes and assigning to an array for that activity and storing them as an integer
31  # But due to the fact as we are looking the ones that do not exist there will come back as nan we
32  #  are saying we do not want them
33  # This therefore allows us to get what we want
34  # WALKING UPSTAIRS
35  walkingup = labels.index.where(labels == 'WALKING_UPSTAIRS').values
36  walkingupcleaned = [x for x in walkingup if str(x) != 'nan']
37  walkingupcleanedINT = np.array(walkingupcleaned).astype(int)
38
39  # STANDING
40  standing = labels.index.where(labels == 'STANDING').values
41  standingcleaned = [x for x in standing if str(x) != 'nan']
42  standingcleanedINT = np.array(standingcleaned).astype(int)
43
44  # LAYING
45  laying = labels.index.where(labels == 'LAYING').values
46  layingcleaned = [x for x in laying if str(x) != 'nan']
47  layingcleanedINT = np.array(layingcleaned).astype(int)
48
49  # SITTING
50  sitting = labels.index.where(labels == 'SITTING').values
51  sittingcleaned = [x for x in sitting if str(x) != 'nan']
52  sittingcleanedINT = np.array(sittingcleaned).astype(int)
53
54  # WALKING
55  walking = labels.index.where(labels == 'WALKING').values
56  walkingcleaned = [x for x in walking if str(x) != 'nan']
57  walkingcleanedINT = np.array(walkingcleaned).astype(int)
58
59  # WALKING DOWN STAIRS
60  walkingdown = labels.index.where(labels == 'WALKING_DOWNSTAIRS').values
61  walkingdowncleaned = [x for x in walkingdown if str(x) != 'nan']
62  walkingdowncleanedINT = np.array(walkingdowncleaned).astype(int)
```

Figure 29 Algorithm to get indexes of Activities

Next, an algorithm that looks at the labels array for specific Activities and get their indexes which occurs in line 35, 40, 45 50,55 and 60 of **Figure 29 is created**. If that specific activity is not found the index value will be returned as **nan**. Therefore, only the index values that are not **nan** which is done in line 36, 41, 46,51, 56, and 61 of **Figure 29** are taken**. After this** they are stored in a **cleanedINT** variable which is done in line 37, 42,47, 52, 57, 62 of **Figure 29**. This process is carried out for each activity.

```
66  x_pca = pca.transform(scaled_data)
67
68  # printing to see the scaled values
69  # print(x_pca)
70  # Printing to see labelled values
71  # print(labels.values)
72
73  # Printing to see walking cleaned o see what it looked like
74  # print(walkingcleaned)
75  fig = plt.figure()
76  ax1 = fig.add_subplot(111)
77  #                       PLOTTING POINTS BASED ON ACTIVITY
78  # Here we are going through the array of avtivties containig their index and matching them to their pca values and plotting them
79  # Doing this makes us able to see how distinct activities are
80  # Walking
81  for i in walkingcleanedINT:
82      wlk = ax1.scatter(x=x_pca[i][0], y=x_pca[i][1], s=10, c='b', marker="s")
83  # WalkingUp
84  for i in walkingupcleanedINT:
85      #
86      wlkup = ax1.scatter(x=x_pca[i][0], y=x_pca[i][1], s=10, c='c', marker="s")
87  # WalkingDown
88  for i in walkingdowncleanedINT:
89      wlkdwn = ax1.scatter(x=x_pca[i][0], y=x_pca[i][1], s=10, c='k', marker="s")
90  # Standing
91  for i in standingcleanedINT:
92      stnding = ax1.scatter(x=x_pca[i][0], y=x_pca[i][1], s=10, c='r', marker="s")
93  #      Laying
94  for i in layingcleanedINT:
95      lay = ax1.scatter(x=x_pca[i][0], y=x_pca[i][1], s=10, c='g', marker="s")
96  #     Sitting
97  for i in sittingcleanedINT:
98      sit = ax1.scatter(x=x_pca[i][0], y=x_pca[i][1], s=10, c='y', marker="s")
99
100 plt.legend((wlk, wlkup, wlkdwn, stnding, lay, sit), ('Walking', 'Walking Up', 'Walking Down', 'Standing', 'Laying', 'Sitting'))
101 plt.show()
```

Figure 30 Plot each activity on graph

After this, the indexes of the activities in the **cleanedINT** array for each activity are taken and create a for loop that looks at **i** in the array and matches them to their PCA value stored in **x_pca** and match I to the X and Y values PCA gives us then plot them. This can be seen from line 81 to 101 of **Figure 30**.

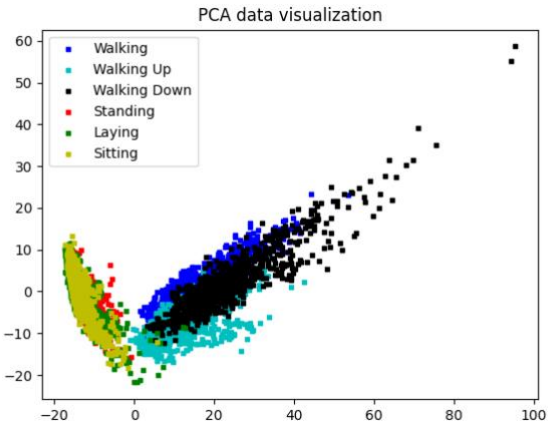


Figure 31 PCA graph

By doing this the resulting PCA graph is plotted in **Figure 31**. From this graph it is evident that there are distinct differences between Walking, Walking Up, Walking Down and Standing, Laying and Sitting. From this data, it can be concluded that the differences between these data point are because Walking activities are not stationary activities and because of this are clustered differently compared to Standing, Laying and Sitting which are activities which require the human body to be stationary.

As a result of this, it can be assumed that during model predictions if models are incorrectly classified it will only be because the data points are too similar to each other and the model cannot tell. An example of this can be seen by Standing, Laying and Sitting overlapping each other and some walking activities overlapping each other therefore if an activity is predicted incorrectly we can assume that it has predicted something from its group. For example, if

the model returns Walking and it is not Walking, we can assume that it will predict it as one of the other Walking activities.

5.4.2 – T-Distributed Stochastic Neighbour Embedding (TSNE)

TSNE is another tool that can be used to visualize high dimensionality data. It works by converting similarities between data points to probabilities (Scikit-learn.org, 2014). As a result of this, it clusters data that are similar together and can be defined as a clustering technique. Unlike PCA it is a non-linear data visualization tool.

```
12    # TSNE is used to map mult dimnsional to two or more dimensions
13    from sklearn.manifold import TSNE
```

Figure 32 Adding TSNE package

To do this the libraries imported for PCA is used but this time TSNE is added which can be seen in **Figure 29**.

```
27    # scale data by assigning scalar method to variable called scaler and applying it on the data
28
29    scl = StandardScaler()
30    tsne_data = scl.fit_transform(df)
31
32    # Reduce dimensions (speed up)
33    pca = PCA(n_components=2)
34    tsne_data = pca.fit_transform(tsne_data)
35
36    # Transform data using TSNE
37    tsne = TSNE(random_state=3)
38    tsne_transformed = tsne.fit_transform(tsne_data)
```

Figure 33 Scaling, Using PCA, then using TSNE

The data is loaded in and **.pop()** is used the same way as shown in **Figure 28**. The data is then scaled which can be seen in line 30 by creating a variable called **scl** and assigning the object StandardScaler to it. Next PCA is carried out on the scaled data before TSNE. Scikit-learn recommends that we do this because this reduces the number of dimensions to a reasonable amount. Due to the fact 561 features exist PCA must be used to suppress the noise and reduce the time taken for the algorithm to run. Next a variable is created called **tsne** containing the TSNE method where its random state is set to 3, which ensures that the result is the same at each run. After this **fit_tranform is called on** the **tsne_data** using **tsne** and assign it to a variable called **tsne_transformed.**

After this, the search for the indexes where the specific activities occur and store their indexes in a variable the same way as shown in figure 27.
Next, the indexes of the activities in the **cleanedINT** array is taken for each activity and a for loop is created that looks at **i** in the array and matches them to their TSNE value stored in **tsne_transformed** and match I to the X and Y values TSNE gives us. These values are then plotted the same way as shown in figure 28. But instead, we use **tsne_transformed** instead of **x_pca.**

```
92        wlk = ax1.scatter(x=tsne_transformed[i][0], y=tsne_transformed[i][1], s=10, c='b', marker="s")
```

Figure 34 Using tsne_tranformed instead of x_pca

Doing this results in the graph in **Figure 35** being plotted.

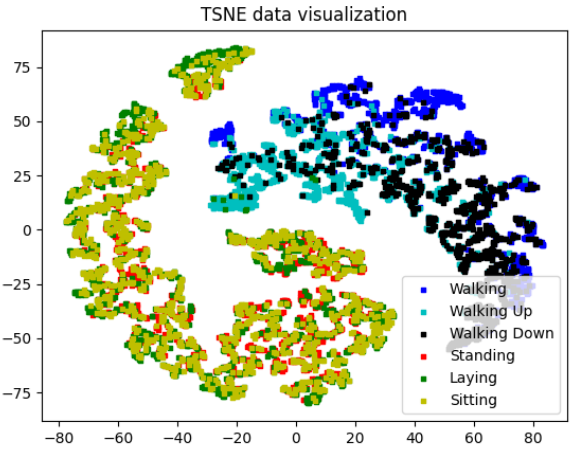


Figure 35 TSNE data visualization

From the graph plotted in **Figure 35**. Just like PCA, there are very close similarities between all Walking activities whilst on the other hand, the plots for Standing, Laying and Sitting are plotted close together. As discussed in Chapter 5.4.1 it can be concluded that because when you are Standing, Laying or Sitting a human is stationary whilst if you are carrying out any walking activity you are not stationary.

It is also evident that unlike PCA which plots points linearly. TSNE does not and instead clusters points that are very similar to each other because of this there is a subsection of certain classifiers that are extremely close to each other and therefore it can be assumed these clusters have the possibility to give us false – positive predictions.

To conclude this chapter, it can be said that based on the above results there are two groups of data which are distinctly different from each other. Group A containing Walking, Walking Up, and Walking Down and Group B containing Standing, Laying and Sitting. As a result, training on the data the data can occur, a careful examination into how successful the different supervised, classification algorithms perform first using Train Test Split and then with K Folds Cross-validation to validate choices made from Train Test Split.

5.5 – Training and Testing our Models

In this chapter how to train the data and plot the data will be discussed this will enable the model to be trained how to classify certain instances. We can then use the model once trained to evaluate unseen test data. This allows a thorough comparison to be made on what model works best when trying to classify different data instances.

5.5.1 – Train the model using Train test split.

In this section of this section, how Train Test Split to train the data and then test it will be discussed. Train Test Split works by splitting arrays of data into random train and test sets

(Scikit-learn.org, 2020), as a result of this provides the ability to evaluate the performance of a machine learning algorithm.

To do this several libraries must be imported to achieve this. This can be seen in **Figure 36**. **Pandas** is used to import the dataset. **Sklearn.metrics** is used to measure different metrics, **accurarcy_score** is used to measure the accuracy of the model. **f1_score** is a way to combine both the recall and precision of a model and can be described as the harmonic mean of the model's precision and recall, according to Thomas Wood an expert in Data Science for more than 10 years (Wood, 2019). **Recall_score** is used to compute the recall of a model and **precision_score** is used to calculate how precise a score is.

Next **pyplot** is used to enable graphs to be plotted and **numpy** to enable mathematical calculations to be easily done. After this the different models which can be seen from **line 7 to 12 of Figure 36 are imported** and then **train_test_split is imported** to enable the measurement of the performance of different algorithms.

```
3   import pandas as pd
4   from sklearn.metrics import accuracy_score,f1_score,recall_score,precision_score
5   from matplotlib import pyplot as plt
6   import numpy as np
7   from sklearn.tree import *
8   from sklearn.linear_model import LogisticRegression
9   from sklearn.neighbors import KNeighborsClassifier
10  from sklearn.svm import SVC
11  from sklearn.ensemble import RandomForestClassifier
12  from sklearn.neural_network import MLPClassifier
13  from sklearn.model_selection import train_test_split
```

Figure 36 Imported libraries for train test split

After this, the data is loaded in and columns we do not need from the data set are removed and empty arrays that can store the model metric values are created. Next the process begun by loading in the data the same way as before, using **Pandas** and drop the *subject* axis as it is not needed when making predictions. This is because this column contains the ID of the participants, and we do not want this to impact our model. Not removing this column could give incorrect results as the model would begin to learn from a feature that does not impact the prediction of the Activity, this is done on **line 17 of Figure 37**. Next empty arrays are created that will be used to store our metric values to enable this project to make comparisons and assigning them to variables which highlight its purpose which can be seen in **line 18 to 22 of Figure 37**.

```
filename = 'Dataset/TrainingCleaned.csv'
df = pd.read_csv(filename)
df.drop(['subject'], axis=1)
trainingScores = []
testingScores = []
f1Scores = []
precisionScores = []
recallScores = []
```

Figure 37 Loading data, dropping column, and creating new arrays

```
#region  Create fuctions for each model

def decision_trees_model(x_train, y_train):
    # creating linear regression object
    decision_trees = DecisionTreeClassifier()
    decision_trees.fit(x_train, y_train)
    return decision_trees

def knn_model(x_train, y_train):
    # creating linear regression object
    KNN = KNeighborsClassifier()
    KNN.fit(x_train, y_train.ravel())
    return KNN

def logistic_model(x_train, y_train):
    # creating linear regression object
    logistic = LogisticRegression(_max_iter=100)
    logistic.fit(x_train, y_train.ravel())
    return logistic

def svm_model(x_train, y_train):
    # creating linear regression object
    svm = SVC()
    svm.fit(x_train, y_train.ravel())
    return svm

def randomforest_model(x_train, y_train):
    # creating linear regression object
    random = RandomForestClassifier()
    random.fit(x_train, y_train.ravel())
    return random

def nn_model(x_train, y_train):
    # creating linear regression object
    nn = MLPClassifier()
    nn.fit(x_train, y_train.ravel())
    return nn

# endregion
```

Figure 38 Creating functions for the models

After loading in the data and creating empty variables to store the model metrics data, the functions that will contain the model and fit it and then returns the fitted model is created. This is done by creating a function for each model which can be seen in **line 32,38, 44,50,56, and 62 of Figure 38**, and passing it the arguments of **x_train, y_train.** Doing this allows the **x_train and y_train** data to be called in the function for use. After this, a variable is created within each function and assigned to the model method for that model this can be seen in line **34, 40, 46, 52, 58, and 64 of Figure 36.** Next .**fit()** is used to fit the x train and y train data resulting in the model being able to learn from the data which can be seen in **line 35, 41, 47, 53,59 and 65 of Figure 36**. Finally, the models that have been fitted is returned allowing the model to be used further which can be seen at the end of the functions in **line 36,42,48,54, 60 and 66** for each model which can be seen in **Figure 38**.

After creating functions for each model, the process of training and testing the model can begin. This is done by creating a **build_and_train_model** function which contains the arguments **df** which it uses to assign the data frame to the models; **target_name** which is a parameter used to define the column containing our classification targets which are *Activity* and **reg_fn,** which is used to define the model function being used which is the same ones defined above in **Figure 38**.

Next global variables are created which match the empty arrays created in **line 18 to 22 of Figure 37** which can be seen in **line 72 to 76 of Figure 39**. By doing this the variable names outside the function can be used allowing graphs to be easily created. Next the code defines X as the data frame when the Activity column is dropped by using **X = df.drop(target_name, axis = 1)** as these are the data features that will be used to make our predictions and use **Y = df[target_name]** as this is the labels of the data. Next **line 80 of Figure 39 is used** to reshape the Y values. The reason for this is because the Y values contain single features.

```
70  def build_and_train_model(df, target_name, reg_fn):
71      # create global variables
72      global trainingScores
73      global testingScores
74      global f1Scores
75      global precisionScores
76      global recallScores
77      # Create X and Y dataframe
78      X = df.drop(target_name, axis=1)
79      Y = df[target_name]
80      Y = Y.values.reshape(-1, 1)
81      # use traintest split
82      x_train, x_test, y_train, y_test = \
83          train_test_split(X, Y, test_size=0.2, random_state=0)
84      # Trai models, get their score and then print the scores
85      model = reg_fn(x_train, y_train)
86      score = model.score(x_train, y_train)
87      print("Training Score :", score)
88      # Predict what the label will be bases on X test data
89      y_pred = model.predict(x_test)
90      # Calculate Accuracy, F1, precision and recall scores
91      accuracy= accuracy_score(y_test, y_pred)
92      f1 = f1_score(y_test,y_pred, average='macro')
93      precision = precision_score(y_test, y_pred, average='macro')
94      recall = recall_score(y_test,y_pred, average='macro')
95      print("Testing Score: ", accuracy)
96      print("F1 Score: ", f1)
97      print("Precision Score: ", precision)
98      print("recall: ", recall)
99
100     # append them to array created above
101     testingScores = np.append(testingScores, accuracy)
102     trainingScores = np.append(trainingScores,score)
103     f1Scores = np.append(f1Scores, f1)
104     precisionScores = np.append(precisionScores, precision)
105     recallScores = np.append(recallScores, recall)
106
107     # get parameters to look at when parameter tuning
108     print(model.get_params())
```

Figure 39 Build and training model

After this **train_test_split** as shown in **line 82 and 83 of Figure 39** is used to divide the dataset into a train and test set. The parameters passed into it which is **X** is used to define that our **X** data which will be the data frame where the target column name is dropped that contains our features. **Y** as a parameter defines what the **Y** data will be which is the reshaped column data frame containing the labels. **test_size = 0.2** means that only 20% of the training data will be split as the test data to validate the model. **random_state = 0** is used to set the seed of the random generator, meaning that the train test splits will always be the same and not randomize and as a result, if 0 is used every time, the same results will be outputted.

After this, a variable called **model** is created which calls the model that has been passed into the function and trains the model with the **x_train** and **y_train** data. The score of the model which is stored in a variable called score using the .**score()** is obtained and then print the score. This can be seen on **line 84 -86** of **Figure 39**.

After this, a variable called **y_pred** which gets the model and predicts what the label will be based on the x_test given to it is created. Next the Accuracy, F1, Precision and Recall Score is calculated by using the libraries imported as discussed at the beginning of this section. This all occurs between **line 88** to **94 of Figure 39**. Next, the scores for each model is printed which is shown from **line 95 to 98 of Figure 39** and then append these scores to the global arrays created earlier using **line 101 to 105 of Figure 39**. Then print the parameter for each model which is shown in line **108 of Figure 39** to get the static parameters.

After building and training the model the models and their scores are printed by creating a variable that stores the build and train function gives it the arguments required for the function shown in **Figure 41**. The arguments given to the function is the target name which is **Activity** and the model that should be used, **df** which is the data frame being used and the name of the function for the model we will be using which is shown in **Figure 38**, this is done for each model. As a result of this, the build and train function shown in **Figure 39** works together with the model functions shown in **Figure 38** and the scores are printed for each model using the code in **Figure 39**. As a result, the scores are obtained which will be discussed at the end of this chapter.

```
112  # Print out model with its metrics based on it classifying activities
113  print('Decision Tree')
114  decision_tree = build_and_train_model(df, 'Activity', decision_trees_model)
115
116  print('K- Nearest Neighbour')
117  KNN = build_and_train_model(df, 'Activity', knn_model )
118
119  print('Logistic Regression')
120  logistic = build_and_train_model(df, 'Activity', logistic_model)
121
122  print('Support Vector Machine')
123  svm = build_and_train_model(df, 'Activity', svm_model)
124
125  print('Random Forest')
126  random = build_and_train_model(df, 'Activity', randomforest_model)
127
128  print('Neural Network')
129  nn = build_and_train_model(df, 'Activity', nn_model)
```

Figure 41 Printing of metrics for each model

```
131  # Creating x Axis
132  algorithms = ['DT', 'KNN ', 'LogR', 'SVM', 'RF', 'NN']
133
134  # Done with help using source[1]
135  fig, ax = plt.subplots()
136  ax.set_ylabel('Accuracy %')
137  ax.set_title('Algorithms')
138  ax.plot(algorithms, trainingScores, 'o-', color="r",
139              label="Training score")
140  ax.plot(algorithms, testingScores, 'o-', color="g",
141              label="Testing score")
142  ax.legend(loc="best")
143  ax.legend()
144  plt.xticks(rotation=90)
145  plt.figure(figsize=(3,4))
146  plt.show()
```

Figure 40 Plotting the graphs

Next, graphs are created to visualize the training set against the test set created by Train Test Split to see what algorithm performs best. This is done by first creating an array that contains the labels for the X-axis which is located on line 132 of **Figure 40**. Next a figure with subplots on line 135 of **Figure 40** are created and set the y label, which is accuracy, the title and plot two-line graphs one for the training score and another for the testing score. This is shown on line 138 to 141 of **Figure 40.** Next the legend is positioned in the best place which is shown on line 142 of **Figure 40**. Instantiate the legend which is shown on line 143 of **Figure 40** and rotate the x labels by 90degrees to reduce the amount of space it takes up. Next the figure size is reduced by passing the parameter **figsize = (3,4)** to **plt.figure** and then show the figure using **plt.show().**

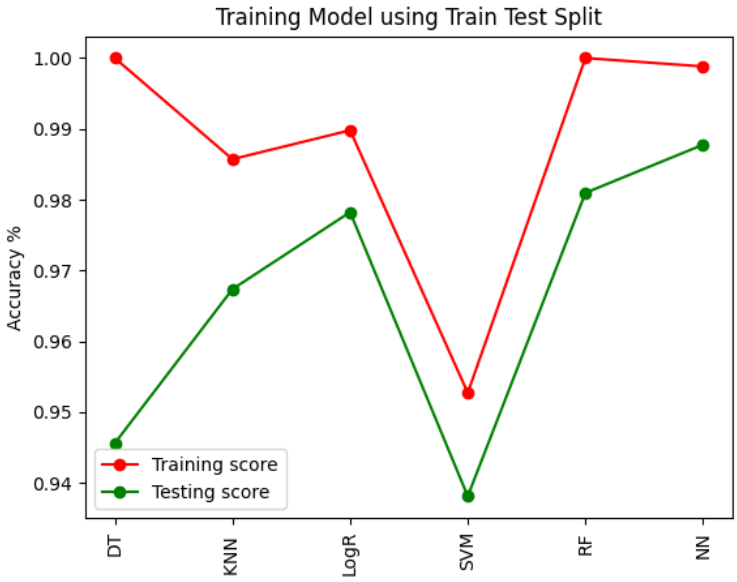


Figure 42 Train test split graph

Carrying out Train Test Split it is evident in **Figure 42** the model that trained the best was Decision tree and Random Forrest with a Training accuracy of 100%. This shows there are problems with the model. It can be said that the reason why Decision Tree a Random Forest has achieved 100% training accuracy is that these models a prone to overfitting. This is because using 80% of the training data as the training set with train test split allows the trees and decision nodes of Random Forest and Decision tree to learn too much about the data and as a result causes overfitting. The worst performing model on training was the Support Vector Machine with 95.27% accuracy. When looking at the test accuracy scores the best performing model was Neural Networks with an accuracy of 97.78% and the worst again was Support Vector Machine with 93.81%. This, therefore, shows that overall, it can be said that Train Test Split highlights that Support Vector Machine is the worst. However surprisingly the 2nd worst model on the training set was Decision Tree with 94.56% test accuracy therefore overall showing us that there is a possibility that it overfits as it is not extremely close to its Training score which was 100%. All figures discussed here can be found in Appendix A Table 7.

From this chapter, it can be seen how to use train test split to train ML models however whilst this is one form of training models. It can be argued to be not the best since it can take a biased approach to training and testing as it does not shuffle the data and just calculates the train and test score based on the test percentage specified. Due to the fact we get 100% for the accuracy of training data for some models, it is important to use a better method to get our scores. One method that can be said to be better is K Fold cross-validation as doing this gives a mean average from training, testing and splitting subsections of the data based on the number of splits as a result in the next section K Fold Cross-Validation will be discussed.

5.5.2 – Validating our models using K-fold Cross validation

According to Jeff Schneider at Carnegie Melon University K-fold Cross-validation works by dividing the data into K subsets. The algorithm then picks out one of the K subsets as the test set and then uses the other K-1 subsets as the training set. After this, the average of the number of K trials is calculated and this is the score that is given for the training and test score. The benefit of this means that it does not matter how the data is being divided, unlike train test split. In addition to this, every data entry gets tested and nothing is missed and therefore provides a more trustworthy score (Cmu.edu, 2021).

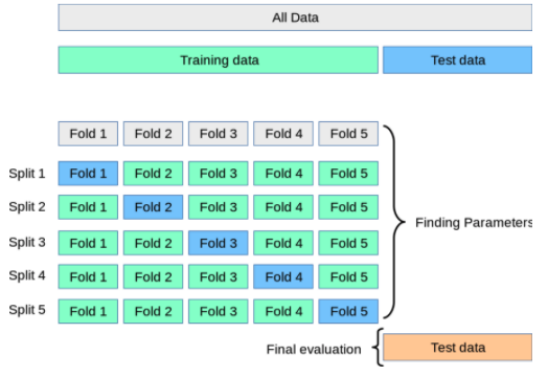


Figure 43 How K-fold Cross Validation Works

Just like Train Test Split the same libraries are imported as highlighted in **Figure 36** however this time **cross_val_score** is added to evaluate model scores by cross-validation and import K-Fold to split the data into consecutive folds this can be seen in **Figure 43**.

```
14    from sklearn.model_selection import cross_val_score, KFold
```

Figure 44 Importing Cross validation and K fold

Next the data is imported and drop the subject column as it is not needed the same way as shown in **Figure 39**.

After this the X variable is created to store all our feature data that will be used to predict our labels and a Y variable to store Activity labels the same way which is shown in line 78 to 80 of **Figure 39**.

Next 4 variables are created scoring, scoring2, scoring3 and scoring4 to store our metrics which are accuracy, precision-recall, and f1 score. This is shown in **line 35 to 38** respectively of **Figure 45**.

```
35    scoring = 'accuracy'
36    scoring2 = 'precision_macro'
37    scoring3 = 'recall_macro'
38    scoring4 ='f1_macro'
```

Figure 45 Score metrics

Next an array is created called models which stores the models being worked on and append them to an array called models which can be seen in **Figure 46**.

```
#region Spot check algorithms/ Create array of all algorithms you'd like to train on
models =[]
models.append(('Decision Tree', DecisionTreeClassifier()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('LR', LogisticRegression()))
models.append(('SV', SVC()))
models.append(('Random Forest', RandomForestClassifier()))
models.append(('Neural Network', MLPClassifier()))
```

Figure 46 Adding models to array called models

```
#region Evaluate models and print results
results =[]
name = []
testingScores = []
f1Scores = []
precisionScores = []
recallScores = []
```

Figure 47 Array to store results

Next arrays are created which are used to store all results and metrics which is shown in **Figure 47**.

Then the **name** which is the specific **model** in the array of **models** is looked at and initialise K- Fold and assign it to a variable called K- Fold. Here the number of K splits which is defined using the parameter **num_folds** is set to 5 with shuffle being set to False. This is because, in Train Test Split, the dataset was not shuffled therefore to ensure a fair evaluation of train test split vs K Fold cross-validation. Next four cross-validation result variables are created which are seen with **cv_results, cv_results2, cv_results3** and **cv_results4** for each model. These variables store the cross-validation score of each metric for each model in the array of models based on the data given to it. This can be seen with the **cross_val_score** method which is used to calculate the cross-validation score has the arguments and parameters **model** we are using in the array of models. The **X** data which is the data being used to predict the label and **Y** which is the label, the **cv** parameter takes the argument **kfold** which store the cross-validation strategy which in our case is K Fold and the scoring parameter takes the scoring variables which represents different metrics as shown in **Figure 47**.

```
for name, model in models: # look through all models in model array
    kfold = KFold(n_splits= 5, shuffle=False) #for each iteration define a KFold instance with a set K value on the random state
    # use cross_val_score four times it is a handy way to both train and evaluate data
    cv_results = cross_val_score(model, X, Y, cv=kfold, scoring = scoring)
    accuracyScore = np.append(accuracyScore, cv_results.mean())
    cv_results2 = cross_val_score(model, X, Y, cv=kfold, scoring=scoring2)
    precisionScores = np.append(precisionScores, cv_results2.mean())
    cv_results3 = cross_val_score(model, X, Y, cv=kfold, scoring=scoring3)
    recallScores = np.append(recallScores, cv_results3.mean())
    cv_results4 = cross_val_score(model, X, Y, cv=kfold, scoring=scoring4)
    f1Scores = np.append(f1Scores, cv_results4.mean())

    msg = "%s: Accuracy: %f, Precision: %f, Recall: %f,F1-Score: %f" % (name,cv_results.mean(), cv_results2.mean(), cv_results3.mean(), cv_results4.mean())

    print(msg)
```

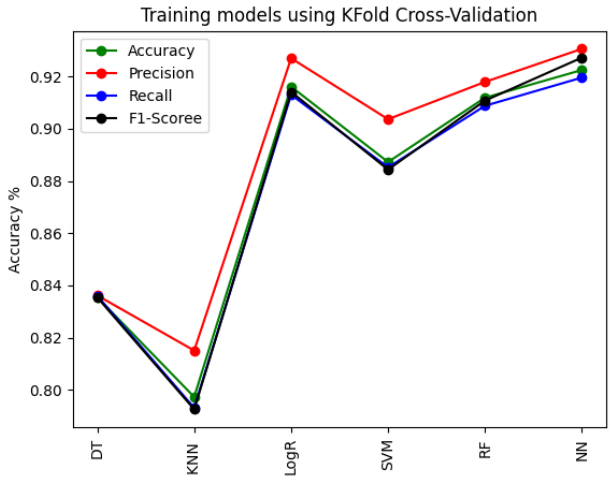
Figure 48 Cross-Validation results printed

Next the code appends the result of each scoring method to their relative scoring array as shown in **Figure 47** and take the mean of the result to get a general view of the score instead of the scores at each K-Fold.

Next a variable called message which prints out the values alongside their score names is created and the model name as shown in line 76 of **Figure 48**.

After this a graph is created that plots the results the same way we did in **Figure 42** for Train Test Split.

By using K- Fold Cross-validation lower CV accuracy is obtained compared to Train Test Split this is expected because as discussed above these scores represent test scores once the K -1 folds are used to train the model and the 1 fold is used to test the model. The score from each split is then taken and the final score outputted is the mean across the number of splits.



From this, we can see that the model with the top three CV scores in order is Neural Networks with 92.42%, Logistic Regression with 91.61% and Random Forest with 90.91%. The model with the worst CV accuracy score was KNN with a score of 79.73%. These results show that based on train test split and K- Fold Cross-validation there is a similarity with NN being the best as they both gained the highest scores on their experiments. However, K-Fold Cross-validation gives a different model which is the worst compared to what Trian Test Split gives us. However, it can be concluded that the CV score should be trusted more than the train test split. This is because the test set and train set is fairly used due to the folding of the dataset and the use of splits. As a result, it can be said that going forward we will prioritise the CV scores.

In addition to this it is evident that the model with the greatest precision was the Neural Network with a precision score of 92.74% and the worst was KNN with a score of 81.54%. Precision shows us the ratio between the True Positives and all Positives. For this projects problem statement it shows the measurement of the labels we correctly identify out of all the labels. As the precision score is high this shows that it is quite precise.

The model with the greatest recall score was Logistic Regression with a CV recall score of 91.3% and the model with the worst CV recall score was KNN with a score of 79.32%. The recall is the measure of the model correctly identifying True Positives. As a result of this, it makes it known that out of all data instances where a specific label occurs. How many did we correctly identify of this label (Analytics Vidhya, 2020)?

It is also evident that the model with the greatest F1- Score was Neural Network with a CV F1- Score of 92.12% and the model with the worst F1 Score is KNN with an F1 – Score of 79.26%. F1 Score can be described as the harmonic mean of Precision and Recall and because of this gives a better measurement of incorrectly classified case since in real – life classifications imbalances can occur and because of this the F1 score is better to be used.

$$recall = \frac{true\ positives}{true\ positives\ + false\ negatives} \quad precision = \frac{true\ positives}{true\ positives + false\ positives} \quad F_1 = 2 * \frac{precision * recall}{precision + recall}$$

Figure 49 Equations for Recall, Precision and F1

To conclude cross-validation score will continue being used as a true representation of how good the models are performing. As this is a unbiased metric so we can evaluate models fairly. In addition to this it can be seen that the models that perform the best are NN and Logistic Regression as a result in the next chapter it will be explored how to fine-tune these models further to get greater accuracy.

5.6 - Hyperparameter tuning

In this chapter how we carry out hyperparameter tuning to try and optimize our models and fine-tune them to try and help us to get the model with the best accuracy possible will be discussed. Hyperparameter tuning helps to evaluate the impact different parameter values have on a model. From this this project will be able to analyse what parameter values give the highest accuracy score. To do this GridSearchCV is used. GridSearchCV is an exhaustive search method used to find out what parameters work best for a model. To begin fine-tuning models' libraries must be imported which are like that which was used in **Figure 36**. This time GridSearchCV is imported to carry out GridSearch. **plot** is imported from **sklearn_evaluation** to plot the results of our GridSearch, therefore, allowing results to be visualized to make a thorough evaluation. **ExcelWriter** from pandas is imported to create tables of results after each model has been run.

```
12  from sklearn.model_selection import GridSearchCV
```

Figure 50 Importing KFold and GridSearchCV

```
14  from sklearn_evaluation import plot
```

Figure 51 Importing plot from sklearn_evaluation

```
15  from pandas import ExcelWriter
```

Figure 52 Importing ExcelWriter from pandas

Next data is imported and drop the subject column as it is not needed the same way as shown in **Figure 37**.

```
29  def decision_trees_model(x_train, y_train):
30      # creating Decision Tree Classifier method
31      decision_trees = DecisionTreeClassifier()
32      # Create parameters to search
33      max_depth = [1, 10, 50, 100, 1000]
34      min_samples_leaf = [1, 25, 50, 100, 300]
35      # create GridSearch Dictionary
36      grid = dict(
37          max_depth=max_depth, min_samples_leaf=min_samples_leaf
38      )
39      grid_search = GridSearchCV(estimator=decision_trees, param_grid=grid, error_score=0, cv = 5)
40      grid_results = grid_search.fit(x_train, y_train)
41      # Plot GridSearch Results
42      ax1 = plot.grid_search(grid_results.cv_results_, change='max_depth', kind='bar')
43      ax1.set_title('When maximum depth and minimum samples leaf')
44      plt.show()
45
46      # Create CSV file to hold table of results.
47      df1 = pd.concat([pd.DataFrame(grid_results.cv_results_["params"]),
48                       pd.DataFrame(grid_results.cv_results_["mean_test_score"], columns=["Accuracy"])], axis=1)
49      print("DataFrame for Decision Tree")
50      print(df1.values)
51      writer = ExcelWriter('Decision Finetuned.xlsx')
52      df1.to_excel(writer, 'sheet2')
53      writer.save()
54
55      return grid_results
```

Figure 53 Structure of how GridSearch is done for models

After this, functions for each model is created an example of how models in this project are fine-tuned with GridSearch is highlighted in Figure 53 of this projects Decision Tree function. Here a function is created which takes the arguments of **x_train**, **y_train**. The method for the Decision Tree classifier is assigned to a variable called **decision_trees**. Next, the parameters we want GridSearch to run on for the Decision Tree are defined. Next these parameters are added to the grid dictionary which can be seen in **line 37 of Figure 45**, then a variable called **grid_search** is created and within this variable assign it the method **GridSearchCV ()**. This method takes the arguments of the estimator which is equal to the **decision_trees** variable as the estimator takes in the model. The argument **param_grid** takes in the GridSearchCV grid and the argument **error_score** defines when there is an error what should the score be which is set to zero. **Cv** is used to take the number of cross-validation folds.

After this the code fits the **x_train** and **y_train** to the GridSearch. After this, bar chart showing the visual impacts of different parameters on the model is plotted, by creating a variable called ax1 and assigning it **plot.grid_search**. **ax1** takes in the argument values of the GridSearch results which are done using **grid_results.cv_resuts**. The parameter **change** takes the parameters of the model that will be changing and is also the parameter we will be tracking on **line 42 of Figure 53.** In this method the parameter of **kind** is set to **bar** which makes a bar graph be plotted.
Next the title of the graph is set and the graph shown on **line 45** and **46** respectively of **Figure 53**.

Next **Pandas** is used to create a new data frame and concatenate the parameter of the grid results alongside the accuracy and create a table of results for it. This is done by writing it to a file defined as a string within the **ExcelWriter** method and then call the **.save()** method on the writer to save the file then return **grid_results** so it can be used at a later date if needed.

This process is carried out for all the different models.

As GridSearchCV uses cross-validation already there is no need to specify a splitter as a result the X and Y data as follows in **Figure 54** is passed as arguments to each model.

```
global best

X = df.drop(target_name, axis=1)
Y = df[target_name]
Y = Y.values.reshape(-1, 1)

model = reg_fn(X, Y)
```

Figure 54 Passing X and Y data to model

Next, the models best score alongside the parameters which gave us the best result are printed this can be seen in **Figure 55**.

```
253     print("Best: %f using %s" % (model.best_score_, model.best_params_))
```

Figure 55 Print best model score alongside best parameters

```
207     means = model.cv_results_['mean_test_score']
208     stds = model.cv_results_['std_test_score']
209     params = model.cv_results_['params']
210     for mean, stdev, param in zip(means, stds, params):
211         print("%f (%f) with: %r" % (mean, stdev, param))
212     best = np.append(best, model.best_score_)
```

Figure 56 Get metrics

Further metrics from GridSearch such as the mean test score alongside its standard deviation and the exact parameters used throughout the entire GridSearch are obtained. This can be seen from **line 207 to 212 of Figure 56** respectively with **line 210 and 211 of Figure 56** being used to print these metrics and **line 212 of Figure 56** being used to append the best score from each model to an array called best which is later used to plot a graph of the best scores with their models to make a comparison.

The plotting of the graph was done the same way as shown in **Figure 40**.

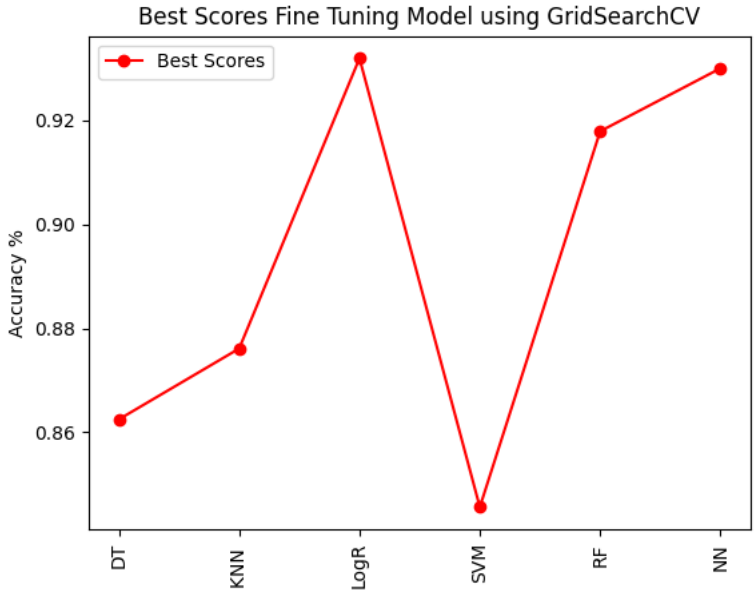


Figure 57 Model scores when fine tuned

To conclude this chapter, it can be seen that hyperparameter tuning allows the best parameters for each model to be found resulting in an increase in accuracy. The scores of the hypermeter tuning will be discussed further in Chapter 6 where the results of the hyperparameter tuning and the meaning of their results will be evaluated.

The next section will discuss the last step before we deploy the best model which is to evaluate the best overall hypermeter tuned model on the test set which we have been holding back only for testing purposes. As shown in **Figure 57** the model that performed the best was Logistic Regression as a result, this model will be used to evaluate our Test set in the next chapter.

5.7 – Evaluating model on our Test set

According to Aureilien Geron in his book, ~~he states~~ once hyperparameter tuning is carried out we should now go on evaluating our model on the Test data to see how well it performs on data it has never seen. This is because when our machine learning models are being deployed our model will be predicting values it has never seen. Therefore how we evaluate our best model on our test set will be discussed.
The same code used in section 5.5.1 is used this time the parameters for each model that yielded the best result is used. The model is trained using only the train set and tested using the entire Test set.
Doing this yielded the results shown below in **Figure 58**.

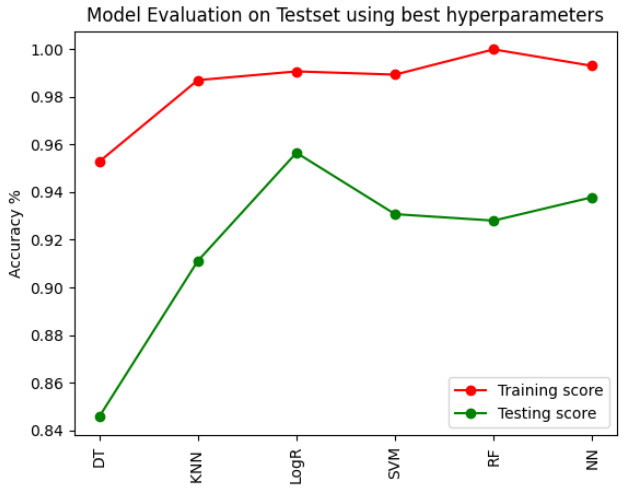


Figure 58 Code used to evaluate model on TestSet

Doing this it can be seen that Logistic regression still performs the best with an approximate score of 95% test score compared. The second best was Neural Networks that gained a test score of approximately 93.8% and the third best was SVM with an approximate score of 93%. Unlike before hyperparameter tuning was carried out when using train test split. The best was Neural Network, then Random Forest Logistic regression this shows that the tuning of a model can have a profound impact on the performance and because of this proves that by the fine-tuning model we can yield greater more accurate results.

6 – Evaluation of the different Models

In this chapter the effect different parameters have on the accuracy of our Human Behaviour Activity dataset will be discussed. Whilst carrying out hyperparameter tuning the test set was not included as doing this could lead to a model that is good for the entire dataset and could lead to overfitting.
The scores shown for each model were carried out using GridSearchCV using a cross-validation split of 5. This means the score that is shown will be the cross-validation score of 5 K-folds. The mean test score is what will be evaluated as the average score across all splits will be analysed because would like to see how different models perform with an unseen dataset based on changes to their parameters.

6.1 – Decision Trees Model Evaluation

For the evaluation, the Decision Tree model will be defined with different parameters to see how well they perform based on their accuracies. To do this only the training set will be used. To carry out our evaluation of different parameters GridSearchCV was used which is shown in Chapter 5.6.

The following table shows Decision Trees Static Parameters (this is parameters that will not be changed) for the tuning of the models:

Static Parameters	Value
Minimal Cost-Complexity Pruning	0
Class Weight	None
Criterion	Gini
Max Features	None
Max Leaf Nodes	None
Min Impurity Decrease	0
Min Impurity Split	None
Min Samples Leaf	1
Min Weight Fraction Leaf	0
Random State	None
Splitter	Best

The following table shows Decision Trees Model Evaluation variable Parameters(this is parameters that will not be changed) for the tuning of the models:

Variable Parameters	Possible Values
Max Depth	1, 10, 50, 100,1000
Min Samples Split	1, 10, 50, 100,1000

The decision to change these parameters was because my prior research in section 4.5.4 highlighted that the depth and minimum samples split can cause the accuracy of the Decision Tree to change. Therefore, these parameters were picked out to evaluate how do they impact the Decision Tree.

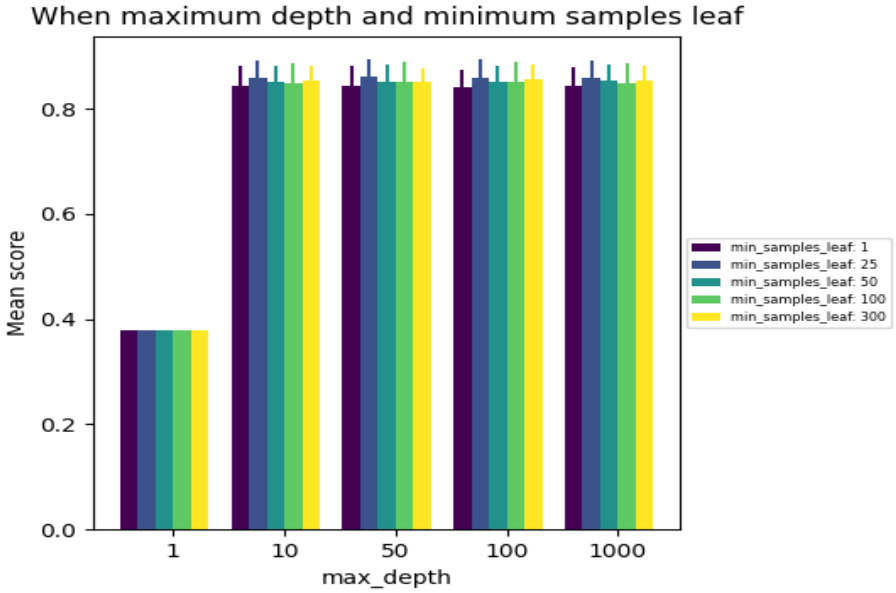


Figure 59 Effect of Max depth and Min samples split on Decision tree

By using GridSearchCV it is evident the best score was 86.09% with reference to Table 1 in Appendix A, with a maximum depth of 1000 and minimum samples split of 25 and the worst CV score was 37.79% with a maximum depth of 1 and Minimum samples depth of 300.

It can be said that the reason for this is because the greater the depth of the Decision tree the more decision nodes available to be created meaning that the algorithm filters the data for the classifier more because this leads to a greater amount of accuracy. However, this is not always the case this is because our table of results in Appendix A Table 1 shows that the second highest CV score was 85.96% with a maximum depth of 10 and minimum samples split of 25 and the third-highest score being 85.93% with a maximum depth of 50 and minimum samples split of 25. This, therefore, shows that the greater the depth does not necessarily mean the better the score.
In addition to this it is evident that when the maximum depth of our tree is 1 this causes the accuracy of the DT model to be low. This is because as the depth is only one it does not have enough decision nodes to make a solid decision on what label the data belongs to and because of this across the 5 experiments where the Maximum depth was one it had an average score of 37.80%

In addition to this, it can be seen there is an optimal value of the number of samples split which is 25 due to the fact our top four results all have a minimum simple split of 25.

To conclude it can be said that higher the maximum depth does not necessarily mean that the score will increase. The fact that out of the 5 experiments that were conducted where the minimum samples split was 25 four of the results were the top four apart from one where the max depth was 1. This shows there is an optimal value of the number of samples split. In addition to this the highest score gained from GridSearchCV was greater than the test score than that that was given during our K Fold Cross-validation shown in section 5.5.2 and as a result of this shows that it is possible to get better results for Decision Tree by fine-tuning its parameters.

6.2 – K- Nearest Neighbour Model Evaluation

For the evaluation of KNN, the KNN model will have different parameters to see how well they perform, based on their accuracies. To do this only the training set was used. To carry out our evaluation of different parameters GridSearchCV was used which is shown in chapter 5.6.

The following table shows KNN Static Parameters (this is parameters that will not be changed) for the tuning of the models:

Static Parameters	**Value**
Algorithm	Auto
Leaf Size	30
Metric Params	None
Number of Jobs	None

Power Parameter	2
Weights	Uniform

The following table shows KNN variable Parameters (this is parameters that will not be changed) for the tuning of the models:

Variable Parameters	**Possible Values**
Number of Neighbours	1, 10, 50, 100,500
Metric	Manhattan, Euclidean

The decision to change these parameters was because prior research in section 4.5.1 highlighted that the number of neighbours and distance metric can cause the accuracy of the KNN model to change. Therefore, these parameters were picked out to evaluate how do they impact KNN.

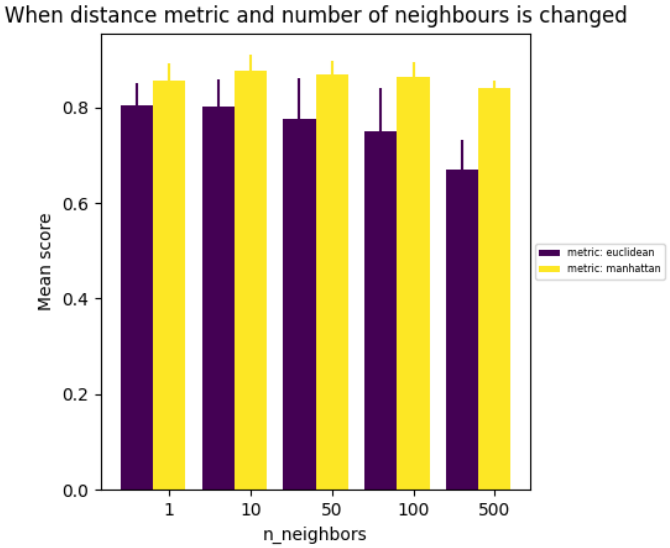


Figure 60 Effect of Number of neighbours and distance metric on KNN

By using GridSearchCV it is evident the best score was 87.60% with the distance metric parameter being the Manhattan distance and the number of neighbours being 10 and the lowest score being 67.08% with the distance metric parameter being used being the Euclidean distance and the number of neighbours being used is 500.

By looking at the graph shown in Figure 63 it can be seen that the use of the Manhattan distance performed better than results that used the Euclidean distance. In a paper written by Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Kiem they concluded that *"for a given problem with a fixed (high) value of the dimensionality d, it may be preferable to use lower values of p. This means that the L1 distance metric (Manhattan Distance metric) is the most preferable for high dimensional applications".* (Aggarwal, Hinneburg and Keim, n.d.) Because of this, it can be concluded that KNN works better when the Manhattan distance is used rather than the Euclidian distance because there are many dimensions in our dataset which is evident as our dataset contains 563 features.

In addition to this based on Table 3 of Appendix A, it is evident that when the Manhattan distance was used the algorithm had the best accuracy when 10 neighbours were used and then when 50 neighbours were used. As a result of this, it is evident that the lower the number of neighbours but several neighbours more than 1 the algorithm can make a solid comparison. This is because higher the value of K past its optimal value which in our case is 10. The algorithm will have to evaluate more data points and decide whether the K data instance belongs to a class based on what are the nearest neighbours around it. If it has lots of neighbours around it can lead to it making a generalised prediction and not a specific one.
However, when the Euclidean distance is used our best accuracy was 80.52% where the nearest neighbour was one. Just like when the Manhattan distance is used, the best result is obtained when the number of neighbours are low but surprisingly 1 gives the best with the Euclidian distance. This can be said due to the fact of how the Euclidian distance works as it gives us the distance of two close point on a plane.

To conclude we can say that it is evident to us that the Manhattan distance outperforms the Euclidian distance and as highlighted in a paper written by Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Kiem this is because the Manhattan distance performs better on high dimensional data. However, further experiments should be carried out on how the Euclidian distance performs on a dataset with low dimensionality to to make this solid conclusion on the dataset being used. Nonetheless, it can be concluded that the Manhattan distance outperforms the Euclidian distance. Also, it can be concluded that the greater the number of neighbours does not directly lead to an increase of accuracy and there is an optimal value which in our experiments is 25. This can be said because the more neighbours to a data point the more generalised the prediction becomes as it must evaluate more neighbours which can be positioned anywhere on the plane.

6.3 – Logistic Regression Model Evaluation

For the Logistic Regression evaluation, the Logistic Regression model will be defined with different parameters to see how well they perform based on their accuracies. To do this only the training set will be used. To carry out the evaluation of different parameters GridSearchCV was used which is shown in Chapter 5.6.

The following table shows Logistic Regressions Static Parameters (this is parameters that will not be changed) for the tuning of the models:

Static Parameters	**Value**
Class Weight	None
Dual	False
Fit Intercept	True
Intercept Scaling	1

L1 Ratio	None
Max Iterations	100
Multi Class	Auto
Number of Jobs	None
Penalty	L2
Random State	None
Tolerance Stopping Criteria	0.0001
Verbose	0
Warm Start	False

The following table shows Logistic Regressions variable Parameters (this is parameters that will not be changed) for the tuning of the models:

Variable Parameters	**Possible Values**
C value	0.01, 0.1, 1, 10,100,1000
Solver	Sag, Saga

The decision to change these parameters was because the prior research in section 4.5.2 highlighted that the strength of regularization which is the C value. The reason why the solver has been changed is that Scikit-learn documentation states that SAGA is often the best choice and wanted to challenge this idea. Therefore, these parameters were picked out to evaluate how do they impact Logistic Regression.

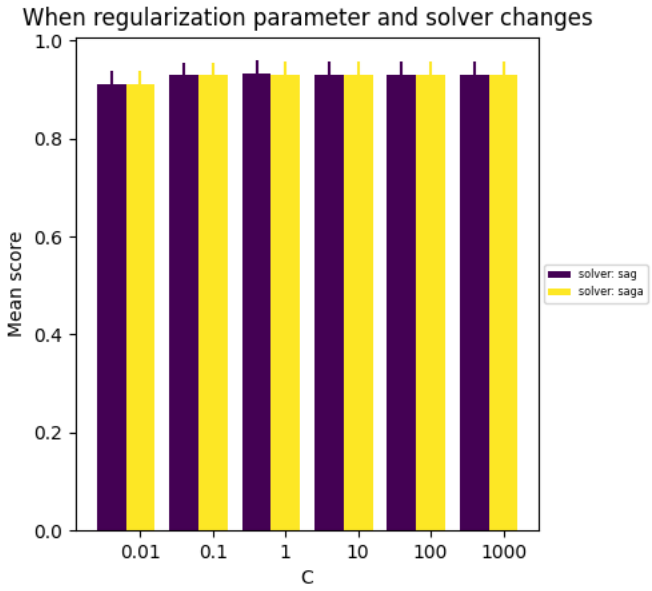


Figure 61Effect of value of C and solver on logistic regression

From using GridSearchCV it can be seen the best score was 93.19% with a C value of 1 and solver of SAG and the worst result was 91.10% with a C value of 0.01 and solver of SAG.

By looking at Table 2 of appendix A it can be seen that on average the SAG solver outperforms the SAGA solver despite Scikit Documentation saying that SAGA is often the best choice.

In addition to this, it can be said that the lower the value of C but more than or equal to one the better the algorithm performs. This can be seen across the two solvers where when the C value was equal to 1 Logistic regression performed the best whereas when the value of C was large it did not perform the best. In addition to this when the value of C was less than 0 this caused the model to not perform well at all. As C represents the regularization parameter it can be said that the higher the value of the more regular and acceptable the model performs and because of this prevents overfitting as it stops the model from becoming overly complex and specific. As a result of this, it can be said that the reason why when C is one it performs at its optimal best is that there is less regularization and because of this the model must perform precisely. The problem with this is that this can lead to overfitting which is when the model learns a lot of detail from the training data and as a result performs poorly on the test data. As the CV score is used, we can assume that it did overfit slightly but still produced a high CV score.

To conclude it can be said that the higher the value of C the more regularization occurs and as a result is less specific and complex and therefore gives a lower amount of accuracy. Whilst even though Scikit learn states that SAGA is the proffered method to use for linear regression. It is evident that this is not always the case but instead SAG is.

6.4 – Support Vector Machine Model Evaluation.

For the Support Vector Machine evaluation, the Support Vector Machine model is defined with different parameters to see how well they perform based on their accuracies. To do this only the training set is used. To carry out our evaluation of different parameters GridSearchCV was used which is shown in Chapter 5.6.

The following table shows SVM Static Parameters (this is parameters that will not be changed) for the tuning of the models:

Static Parameters	**Value**
Break Ties	False
Cache Size	200
Class Weight	None
Independent term in kernel function	0.0
Decision Function Shape	OVR
Kernel	Rbf
Max Iterations	-1
Probability	False
Random State	None
Shrinking	True
Toleration	0,001
Verbose	False

The following table shows SVM variable Parameters (this is parameters that will not be changed) for the tuning of the models:

Variable Parameters	Possible Values
C value	0.01, 0.1, 1, 10,100,1000
Degree	1, 2, 3, 4, 5
Gamma	0.01, 0.1, 1, 10,100,1000

The decision to change these parameters was because prior research in section 4.5.3 highlighted that the strength of regularization which is the C value, the degree and gamma all impact the accuracy of an SVM model. Therefore, these parameters were picked out to evaluate how do they impact SVM.

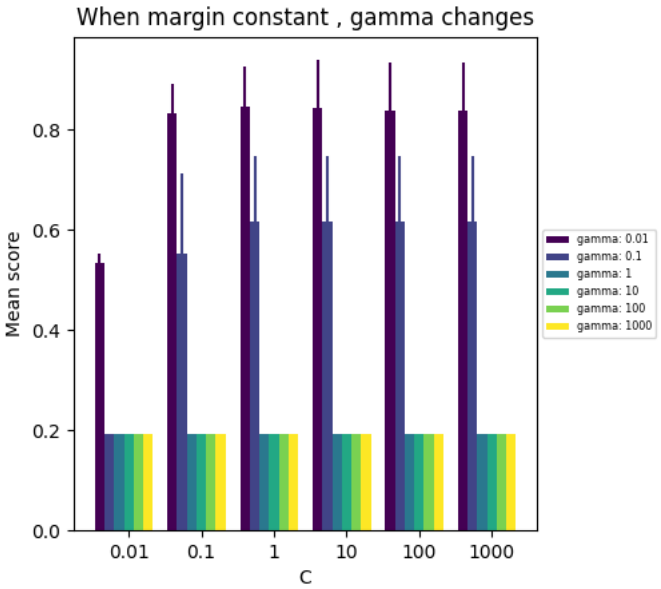


Figure 62 Effect of gamma and changed value of C on accuracy

From using GridSearchCV it is evident that the best score was 84.56% with a regularization C value of 1 and a value of gamma at 0.01. Our worst score was 19.13% with a regularization C value of 0.1 and a gamma value of 10.

By looking at our plotted results in **Figure 62** the lower the value of gamma the higher the accuracy of results. According to Intisar Shadeed Al-Mejibli, Jwan K. Alwan and Dhafar Hamed Abd ((5) (PDF) The effect of gamma value on support vector machine performance with different kernels, 2020). In their paper, they discussed that the lower the value of gamma the broader the decision region becomes, and the higher gamma is the narrower the decision region becomes, and in some cases can generate islands of decision boundaries. They then concluded that the value of gamma on an RBF kernel which in our case is being used, cannot singularly necessarily lead to an increase in accuracy. Instead, it varies from dataset to dataset and whether missing values occurs, the number of attributes and much more. As a result of this, it can be said that during our experiment it is evident that the lower the value of gamma the higher the accuracy on this dataset however with that being the cause it cannot be said to be the case across the board all the time.

It can also be said that just like Logistic Regression on average the lower the value of C the more accurate SVM this is since the regularization parameter is smaller and as a result does not have to regularize and aim to catch many instances. As a result of this, the lower value

of C means lower error and a larger value of C is at more risk of a larger amount of error. However, whilst this is the case it is more prone to overfitting when C is low as less regularization occurs and as a result, the model becomes very complex and specific

From this it can be concluded that on our dataset the lower the value of gamma the higher the accuracy and the lower the value of C the higher the accuracy.

6.5 – Random Forest Model Evaluation

For the Random Forest evaluation, the Random Forest model will be defined with different parameters to see how well they perform based on their accuracies. To do this only the training set will be used. To carry out the evaluation of different parameters GridSearchCV was used which is shown in Chapter 5.6.

The following table shows Random Forest Static Parameters (this is parameters that will not be changed) for the tuning of the models:

Static Parameters	**Value**
Bootstrap	True
Minimal Cost-Complexity Pruning	0
Class Weight	None
Criterion	Gini
Max Depth	None
Max Features	Auto
Max Leaf Nodes	None
Max Samples	None
Min Impurity Decrease	0
Min Impurity Split	None
Min Samples Leaf	1
Min Samples Split	2
Min Weight Fraction Leaf	0
Number of jobs	None
Out of Bag Samples	False
Random State	None
Verbose	0
Warm Start	False

The following table shows Random Forest variable Parameters (this is parameters that will not be changed) for the tuning of the models:

Variable Parameters	**Possible Values**
Number of estimators	10, 20, 50, 100, 500
Max Features	1, 25, 50, 300, 500, 1000

The decision to change these parameters was because prior research in section 4.5.5 highlighted that the number of estimators which represents the number of trees in the forest and the maximum features which represents the maximum number of features to consider when making a split, can impact the accuracy of RF. Therefore, these parameters were picked out to evaluate how do they impact RF.

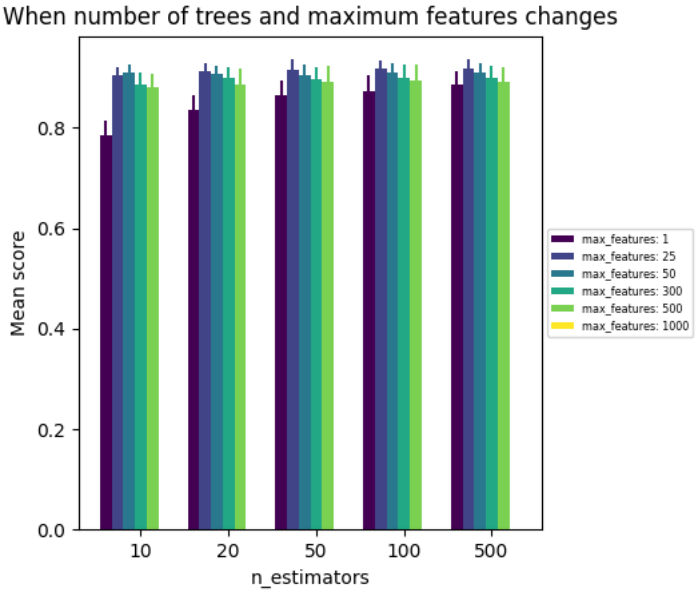


Figure 63 Effect of the change of number of trees and maximum features

By using GridSearchCV it is evident that the best result was 91.77% with a maximum number of features of 25 and several trees of 50. The worst results occurred where all the maximum number of features were 1000 had the score of 0 and as a result, are not shown in figure 55. While discarding all results where the maximum number of features were 0 the worst result was where the maximum features were 1 and the number of estimators was 10 with an accuracy of 78.56%.

It can be said that there does not lie a solid correlation between the maximum number of features and the level of accuracy but instead from our results shown in **Figure 66** and **Table 5 of Appendix A**. It can be seen there is an optimal value of the maximum number of features number which in this case is 25. It can be said that the reason for this is because too many features being evaluated by each tree causes the model to be unable to make a good enough prediction as it will have too many features to evaluate. On the other hand, having too low of a number of features causes it to be unable to make a uniform decision because there are not enough features for it to evaluate. As a result of this, it can be understood that when the maximum features are 1 on average it performs worse and the second worse accuracy occurs when the maximum number of features is 500.

In addition to this, it can be realised that on average when the number of trees is 500 it outperforms the other values of the number of trees, except from the best score which occurs when the number of trees is 50 and max features are 25. It can be said that the reason for this is the greater the number of threes the more individual models there are to make a greater informed decision. This is since each tree contains deals with a specific randomized sample of the dataset and having multiple decision trees means that the model

deals with more samples and because of this can make a more informed prediction. However, this is not exactly the rule of thumb as it is evident from **Table 5 of Appendix A** that the second-highest amount of the number of trees which is 100 is not on average producing the second-best results when comparing it against the other maximum features. As a result of this, it cannot exactly be concluded that the greater the number of trees the higher the accuracy and instead we can say it varies case by case.

To conclude it can be said that there exists an optimal value for the maximum number of features which in this case is 25. But there exists no strong correlation between the number of trees and the accuracy and as a result, it varies case by case.

6.6 – Neural Networks

For the Neural Networks evaluation, the Neural Network model is defined with different parameters to see how well they perform based on their accuracies. To do this only the training set will be used. To carry out our evaluation of different parameters GridSearchCV was used which is shown in Chapter 5.6.

The following table shows Neural Network Static Parameters (this is parameters that will not be changed) for the tuning of the models:

Static Parameters	**Value**
Activation	Relu
Alpha	0.0001
Batch Size	Auto
Beta 1	0.9
Beta 2	0.999
Early Stopping	False
Epsilon	1e-08
Learning Rate	Constant
Initial Learning Rate	0.001
Maximum number of loss function calls	15000
Max Iterations	200
Momentum	0.9
Maximum number of epochs to not meet tolerance	10
Nesterovs Momentum	True
Power of t	0,5
Random State	None
Tolerance	0.0001
Validation Fraction	0.1
Verbose	False
Warm Start	False

The following table shows Neural Network variable Parameters (this is parameters that will not be changed) for the tuning of the models:

Variable Parameters	Possible Values
Hidden Layer Sizes	25, 50, 100, 500, 1000
Solver	Adam, SGD

The decision to change these parameters was because prior research in section 4.5.6 highlights that the hidden layer sizes which are the number of neurons and the solver impact the accuracy of a Neural Network. Therefore, these parameters were picked out to evaluate how do they impact Neural Networks.

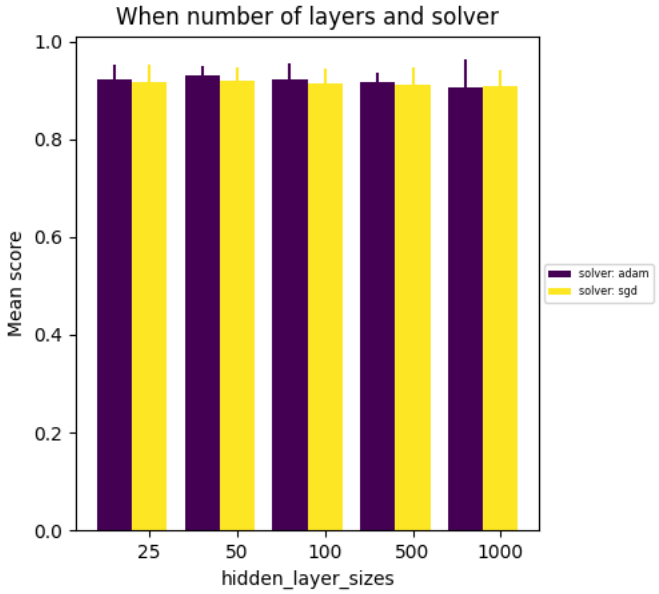


Figure 64 Effect of the change of number of hidden layers and solver

By using GridSearchCV it can be seen the best score was 93.10% with a total number of hidden layers of 25 and the solver being ADAM and the lowest score being 91.29% with the number of hidden layers being 50 and the solver being SGD.

By looking at the results for ADAM we can see that on average the lower the number of hidden layer size the higher the accuracy for ADAM and the higher the number of hidden layer size the less accurate ADAM is. One reason why it can be said that ADAM outperforms SGD is that ADAM makes use of the benefits of both RMSProp and SGD which are also solvers. ADAM uses the squared gradients to scale the learning rate and takes advantage of momentum by using the moving average of the gradient instead of one constant gradient-like SGD. According to a paper written by Jingzhao Zhang, Sai Praneeth Karimireddy, Andreas Veit, Seungyeon Kim, Sashank J Reddi, Sanjiv Kumar and Suvrit Sra (Veit, 2019). One of the reasons why ADAM outperforms SGD is due to the that that ADAM does well when the algorithm has to spend a lot of attention on the dataset with many features or high dimensions. As a result of this ADAM performs gradient clipping to deal with heavy-tailed noise. As a result of this, it can be said that possibly one reason why ADAM outperforms SGD is due to the amount of attention that the model must commit to the dataset as it contains a large amount of dimensionality.

It can also be seen that the increase of hidden layers sizes does not exactly lead to an increase in results as the use of 1000 neurons alone does not lead to an increase in accuracy when SGD or ADAM is used. Instead the lowest amount of hidden layer size is the best for ADAM and 500 hidden layers size comes out to be the best for SGD and not 1000. As a result of this, it can be said that the higher the number of neurons alone does not exactly lead to an increase in accuracy. Instead, there can be said to be an optimal value for the number of neurons for SGD which in our case is 500. For ADAM it can be said that the lower the hidden layer size the greater the accuracy.

Overall, to conclude it can be said that the lower the hidden layer size, when used with ADAM, yields a better result, but this may possibly depend on the dataset used. As a result, experimentation should be carried out when using any other dataset. But we can say with our dataset this is true. On the other hand, when using SGD there is an optimal value for the number of hidden layer size. Whilst the best accuracy is given when ADAM is used, and lower hidden layer size are used.

7 – Deployment

In this chapter the process to deploying our model as an API using FastAPI will be discussed based on the research carried out in Chapter in 4.6.

7.1 - Using FAST API to deploy the model

Once a model has been evaluated on its test data, it can then be deployed. Deploying the model enables it to be put into real-life production, therefore, allowing it to go past just being machine written code and then into being something that can be used. Due to time constraints, a UI was unable to be developed therefore an API was built. An API was decided to be built because an API has the ability to allow two pieces of software to talk to each other, and post- University further exploration will be carried out to bring this solution to market as shown in section 4.6.

This section was carried out with the help of GeeksforGeeks FastAPI tutorial (GeeksforGeeks, 2020).

First like all the other pieces of code we must import libraries that will be used in this section.

```
2   import pandas as pd
3   # Importing Necessary modules
4   from fastapi import FastAPI
5   import uvicorn
6   from pydantic import BaseModel
```

Figure 65 Importing if libraries required

Just like the other chapters Pandas is used. However, in addition to Pandas, FastAPI is imported. FastAPI is a Python framework that enables the use of a REST interface to implement applications. The REST interface is accessed through a REST API. The next library imported is **BaseModel** from **pydantic**. Pydantic is used to carry out data validation and therefore data sent to the API to be validated. For example, if strings are sent to the API a validation message will be emitted. Uvicorn is also used. Uvicorn is a **library** which allows fast ASGI server implementation. As a result, creates an Asynchronous Server Gateway Interface allowing our API containing our model to be ran.

```
# Declaring our FastAPI instance
app = FastAPI()
#  Load Data
filename = 'TrainingCleaned.csv'
df = pd.read_csv(filename)


# Get our X and Y
X = df.drop(['subject', 'Activity'], axis=1)
Y = df['Activity']

# Build and train our model
model = LogisticRegression(C=1, solver='sag')
model.fit(X, Y)
```

Figure 66 Creating model

After this, an instance of FastAPI is created. This enables FastAPI to be used anywhere in the code with the variable app.

Next the data that will be used to train the model is loaded in and X and Y data is chosen. Next the model that is intended to be used for the API to predict the data labels is defined. As the model that gives the best accuracy is Logistic Regression the parameters that achieved this is defined. Next the training data is fit on the model to train it, this can all be seen in Figure 66

```
class BehaviourActivity(BaseModel):
    tBodyAccmeanX: float
    tBodyAccmeanY: float
    tBodyAccmeanZ: float
    tBodyAccstdX: float
    tBodyAccstdY: float
    tBodyAccstdZ: float
    tBodyAccmadX: float
    tBodyAccmadY: float
    tBodyAccmadZ: float
    tBodyAccmaxX: float
    tBodyAccmaxY: float
    tBodyAccmaxZ: float
    tBodyAccminX: float
    tBodyAccminY: float
    tBodyAccminZ: float
    tBodyAccsma: float
    tBodyAccenergyX: float
    tBodyAccenergyY: float
```

Figure 67 Using BaseModel to validate that

After this, Pydantic can be used to validate all fields of the API. In **Figure 67** a class called **BehaviourActivity** which uses BaseModel as an argument is created. BaseModel provides a base class for building a model which takes the specific data values. By setting each column of the dataset to float this ensures all incoming data that matches these field are floats if the incoming data does not match this data type a type error will be shown. Therefore, helping us to validate the incoming data.

Next a path operation for our root endpoint is defined. To do this **@app.get("/") is used** which can be seen in Figure 68. This tells FASTAPI that the function below has the job of dealing with requests that go to the path within quotation marks which in this case is the path **/** with a get operation.

```
591  # # Defining path operation for root endpoint
592  @app.get('/')
593  def main():
594      return {'message': 'Welcome to Khrishawns ML Deployment!'}
```

Figure 68 Defining Path operation for root endpoint

```
597  # # Defining path operation for /name endpoint
598  @app.post('/predict')
599  def predictor(data: BehaviourActivity):
600      test_data = [[
601          data.tBodyAccmeanX,
602          data.tBodyAccmeanY,
603          data.tBodyAccmeanZ,
604          data.tBodyAccstdX,
605          data.tBodyAccstdY,
606          data.tBodyAccstdZ,
607          data.tBodyAccmadX,
608          data.tBodyAccmadY,
609          data.tBodyAccmadZ,
610          data.tBodyAccmaxX,
```

Figure 69 Defining endpoint

Next a function called predictor which is carried out using @app.post() and is found at the path /predict is created. **The app.post()** function is used to handle post requests. Post requests is a method that accepts the data enclosed in a message. In this case, our post request will be accepting the data values we send to it to carry out a prediction. Within this function contains an array that contains the headings created in the BaseModel. As an array is created that takes the values inputted within the API and from there predicts what class it belongs to.

```
1165        class_label = model.predict(test_data)[0]
1166
1167
1168        # If the encoded numbers come back what Activity is it?
1169        if(class_label == 0):
1170            prediction = 'Laying'
1171        if (class_label == 1):
1172            prediction = 'Sitting'
1173        if ( class_label == 2):
1174            prediction = 'Standing'
1175        if ( class_label == 3):
1176            prediction = 'Walking'
1177        if (class_label ==4):
1178            prediction = 'Walking Downstairs/ 4'
1179        if (class_label == 5):
1180            prediction = 'Standing/ 5'
1181
1182        return {prediction}
```

Figure 70 Predicting Values

In Figure 70 the predicted labels are obtained by passing the model **test_data,** which contains the values which have been sent to the API for prediction. Next the value at array position 0 is taken as this is where the encoded label occurs.
Next an if statement is created for each label so that when the label is predicted if it is a certain number, the API can turn it back into its string value. This can be seen from line **1168 to 1180 of Figure 70.**

Next the prediction is returned as a string after the if statement on **line 1182,** as this allows the API to emit the label as a string.

As the API has now been created, the next chapter will discuss the testing of the API.

7.2 – Testing the API

To load up the API type the following command into our IDEs integrated terminal:

uvicorn basic-app:app – reload

basic-app is used to refer to the name of the file which contains the model and API, **app** is used to refer to the FastAPI instance created under the variable and **-reload** is used to restart the server every time we reload.

Figure 71 Running the API on http location

Doing this creates a uvicorn server located at http://127.0.0.1:8000 which can be seen in Figure 71.

Next the API UI is loaded up by using the http link http://127.0.0.1:8000/docs which loads the web page shown in **Figure 72.**

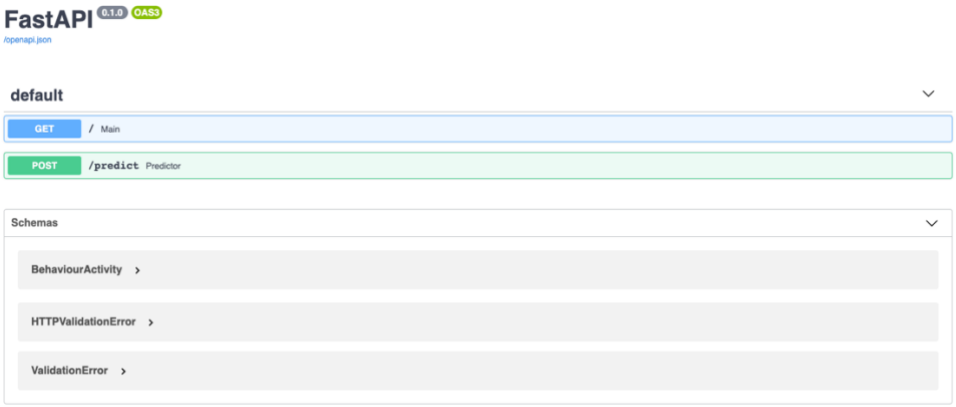


Figure 72 FAST API UI

Within the UI it is possible to post data to the API to do this press the post section of the UI and press **Try it out**.

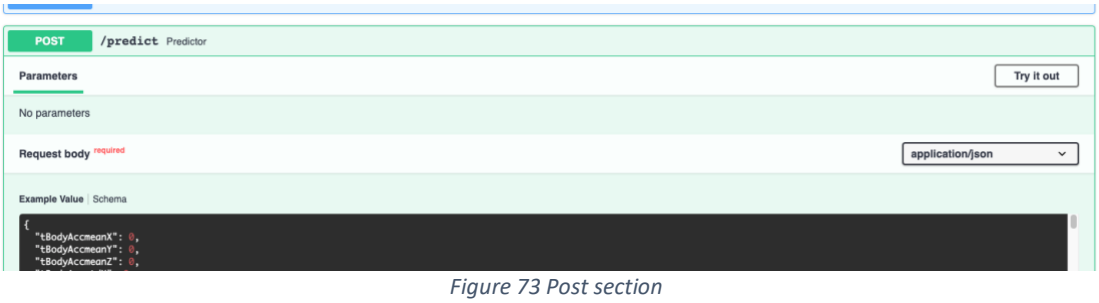


Figure 73 Post section

Doing this presents a field where we can add data into the API once data is added we hit execute. The data we will be using is data from the test set which is for Laying which has been turned into a JSON format for the API.

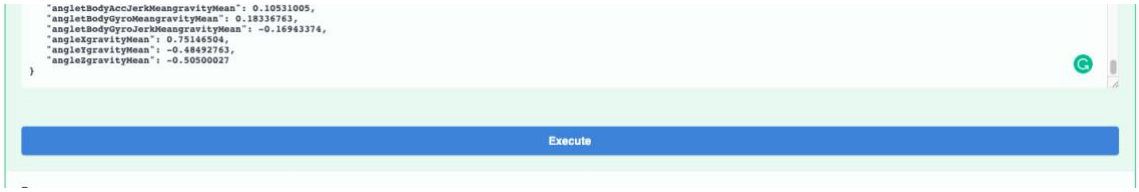


Figure 74 Laying Data instance as JSON

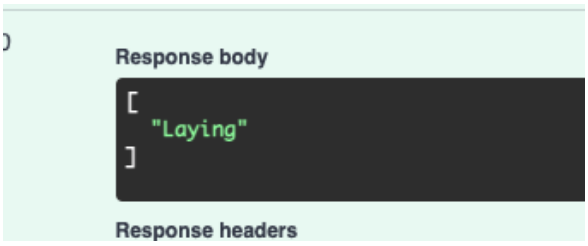


Figure 75 API response

The API is executed pressing by pressing **Execute** it then gives the response body saying Laying has been predicted.

We do this for all activities which show us our API can predict activities.

To conclude this chapter, it can be said that the use of FastAPI made it possible to learn how FastAPI can be used in a much bigger scalable project which will be further explored post-University.

8 – Evaluation of entire project

It can be said that the initial plan of this project was to Investigate different machine learning models on Human Behaviour Activity and then deploy this model as a system that can be used to automate the home based on the activity of a Human being. However, due to time constraints, this was not possible. Before starting this project, it was recommended by Robert Mercas a senior lecturer at Loughborough University that I do not go ahead and do it all but just investigate what machine learning models work best on human behaviour activity and learn how to develop an ML pipeline. I have successfully managed to do this and with the extra time, I had left also produced an API with the use of FastAPI. Overall, I have learned how to program in Python to carry out Machine Learning. In addition to this, I learned how different ML models work and what parameters can have changed to make them yield better results. Furthermore, I am proud to say that even though I was not taught how to carry out an ML pipeline in Python I was able to successfully do it.

In addition to this, I also learned to deploy an ML model as a basic API very and as a result, will be looking further after my degree to carry out greater research. Overall, it can be said that I learned lots of professional skills whilst carrying out this project For example how to carry out a project in an academic style, how to reference well in a project, how to use different sources of information, how to find people that have worked on things similarly in

the past and from findings make decisions on what to make or how to structure a project and build upon what they have done.

I did have problems when carrying out some programming such as grouping the labels to successfully visualize the data in chapter 5.4. However, with my determination and grit I was able to overcome these challenges.

I believe that this project could've gone even better if it was not my first time carrying out ML using Python however despite this, I managed to successfully carry out an ML pipeline. In addition to this, if I was given more time, I believe that I would have been able to do even more. Also, my lack of resources was also a problem as I believe that whilst I was using a dataset from Kaggle, it would have been even better if I had the funds and resources available to get my own data. This would be obtained from my own sensors and would feature me building my own wearable device that can be worn by Humans. The experiment would then be carried out using a wearable device that can record data that can be used to automate the home.

Overall, it can be said that this project was very enjoyable, and I learned a lot and look in the future to build upon my findings and explore the idea of turning my findings into a product with the use of Loughborough Enterprise.

9- Conclusion

With the increase of the of over 65's living longer increasing due to the accessibility of very good healthcare older people are beginning to live longer lives. It is fundamental that we ensure that we use the technology available to protect them and help them, as many of them may in the future be childless or will have children at late periods of their life in pursuit of successful careers and not have the luxury of having family members to look after them. As a result, this project was created to investigate different machine learning methods on Human behaviour activity. To explore the likelihood of using machine learning to predict human behaviour activity and from this then use the predicted result to automate the home. As it can help the elderly and possibly in the future be developed into an ecosystem to support all homes no matter what age. By carrying out this project even though not having any knowledge in Python and ML. We can conclude that with the use of the different libraries, methods, functions, and other resources provided by Python for ML anyone is able to create an ML pipeline that can predict human behaviour activity and then use the model created and deploy it as an API.

We can also be proud of the fact that we were able to deploy a model with an of 95.79% on the Test set created by Kaggle for only testing purposes which were shown in chapter 5.7. Due to the fact we passed our threshold defined in chapter 1.1 of 90%+ this shows us that the model we decided to deploy is sufficient enough to be deployed.

The model that performed the best was Logistic regression with a test accuracy score of 95.66% with a C value of 1 and solver of SAG.

The Second-best performing model was Neural Network with an accuracy of 93.79% on the test set with a hidden layer size of 25 and solver of Adam.

The model with the third-highest accuracy on the test set was 93.07% with SVM with a C value of 1 and a gamma value of 0.01.

The model with the lowest accuracy on the test set was 84.56% using decision trees with a max depth of 1000 and minimum samples leaf of 25.

Overall, the findings of this research are quite interesting considering that our best model was Logistic Regression however in our Gap analysis research papers carried out by Kwon and Choi and by Jitenkumar B Rana, Tanya Jha and Rashmi Shetty they stated Random Forest was the best. However, it can be said the reason why we have discovered that Logistic Regression was the best is due to the fact these research papers did not look to carry out research on Logistic regression and as a result, shows my project fills a gap in the academic paper space on Machine learning on Human behaviour activity. In addition to this, we were able to create an API for this project which none of the research papers managed to do.

Overall, we have reached our aim and objective to investigate different machine learning models on Human Behaviour Activity by evaluating the impact different parameters have on different models and then building an API. Even though we did not build a UI for this project which we said would not be done in Chapter 1.1 we will look to explore this further in the future.

It would be very interesting after this project to explore deep learning on Human Behaviour Activity and look to build a system that does not require human observation and analysis. This is because it has become apparent in this project that if the Human being's activity begins to change and does not follow the conventional activities or routines the system will need to learn about these changes of the human being and act accordingly. As a result, this is also something that will be explored post-graduation. Nonetheless, the project was a great learning experience and I look forward to using the skills gained.

Appendix A- Hyperameter Tuning Table of results

Table 1 Decision Tree Grid Search results

Decision Tree fine-tuned using Grid Search		
max_depth	**min_samples_leaf**	**Accuracy**
1000	25	0.860994
10	25	0.859634
50	25	0.859361
100	25	0.858818
100	300	0.857052
1000	50	0.855146
10	300	0.854739
50	50	0.853785
100	50	0.852561
1000	300	0.852426
10	50	0.852017
50	300	0.851882
1000	100	0.850931
1000	1	0.849845
50	100	0.848346
10	100	0.84821
100	100	0.847938
100	1	0.847533
10	1	0.844269
50	1	0.840459
1	25	0.378264
1	1	0.377992
1	50	0.377992
1	100	0.377992
1	300	0.377992

Table 2 Logistic Regression Grid Search results

Logistic Regression fine-tuned using Grid Search		
C	**solver**	**Accuracy**
1	sag	0.931999
10	sag	0.931863
100	sag	0.931319
1000	sag	0.930775
1	saga	0.930775
10	saga	0.930775
100	saga	0.930231
1000	saga	0.930095
0.1	sag	0.929686
0.1	saga	0.92887
0.01	saga	0.911596
0.01	sag	0.911052

Table 3 K Nearest Neighbour GridSearch results

K- Nearest Neighbour fine-tuned using Grid Search		
metric	**n_neighbors**	**Accuracy**
manhattan	10	0.876099
manhattan	50	0.868888
manhattan	100	0.864944
manhattan	1	0.856106
manhattan	500	0.840046
euclidean	1	0.805237
euclidean	10	0.802926
euclidean	50	0.776547
euclidean	100	0.750841
euclidean	500	0.670854

Table 4 Support Vector Machine Grid Search results

Support Vector Machine fine-tuned using Grid Search		
C	**gamma**	**Accuracy**
1	0.01	0.845643
10	0.01	0.843879
100	0.01	0.839526
1000	0.01	0.839526
0.1	0.01	0.83217
10	0.1	0.618098
100	0.1	0.618098
1000	0.1	0.618098
1	0.1	0.616329
0.1	0.1	0.552818
0.01	0.01	0.535096
0.01	0.1	0.191376
0.01	1	0.191376
0.01	10	0.191376
0.01	100	0.191376
0.01	1000	0.191376
0.1	1	0.191376
0.1	10	0.191376
0.1	100	0.191376
0.1	1000	0.191376
1	1	0.191376
1	10	0.191376
1	100	0.191376
1	1000	0.191376
10	1	0.191376
10	10	0.191376
10	100	0.191376
10	1000	0.191376
100	1	0.191376
100	10	0.191376
100	100	0.191376
100	1000	0.191376
1000	1	0.191376
1000	10	0.191376
1000	100	0.191376
1000	1000	0.191376

Table 5 Random Forest Grid Search results

Random Forest fine-tuned using Grid Search		
max_features	**n_estimators**	**Accuracy**
25	50	0.917714
25	500	0.917713
25	100	0.916488
25	20	0.913497
50	500	0.912408
50	100	0.910912
50	50	0.908327
50	20	0.907375
300	500	0.901257
25	10	0.900574
50	10	0.897987
300	10	0.897311
300	100	0.896905
300	50	0.895682
500	20	0.89201
500	100	0.891874
500	500	0.891193
500	50	0.891193
300	20	0.889697
1	500	0.889693
1	100	0.882621
500	10	0.880992
1	50	0.86222
1	20	0.824409
1	10	0.785641
1000	10	0
1000	20	0
1000	50	0
1000	100	0
1000	500	0

Table 6 Neural Network Grid Search results

Neural Network fine-tuned using Grid Search		
hidden_layer_sizes	**solver**	**Accuracy**
25	adam	0.931048
50	adam	0.928461
100	adam	0.92438
25	sgd	0.916632
1000	adam	0.915674
500	adam	0.915408
500	sgd	0.914456
1000	sgd	0.914046
100	sgd	0.913095
50	sgd	0.91296

Table 7 Train Test Split Results

Train Test Split Results					
Model	Training Score	Testing Score	F1 Score	Precision Score	Recall
Decision Tree	1.0	0.9456	0.9434	0.9446	0.9426
K- Nearest Neighbour	0.9857	0.9674	0.9702	0.9713	0.9699
Logistic Regression	0.9898	0.9782	0.9799	0.9797	0.9803
Support Vector Machine	0.9527	0.9381	0.9430	0.9434	0.9431
Random Forest	1.0	0.9810	0.9818	0.9819	0.9819
Neural Network	0.9988	0.9878	0.9886	0.9886	0.9886

\\

Table 8 K-Fold Cross Validation results

K-Fold Cross Validation Results				
Model	Accuracy	Precision	Recall	F1-Score
Decision Tree	83.72%	83.20%	83.18%	81.81%
K Nearest Neighbour	79.73%	81.51%	79.32%	79.26%
Logistic Regression	91.61%	92.70%	91.30%	91.41%
Support Vector. Machine	88.72	90.36	88.52	88.44%
Random Forest	90.91	91.40	91.00	90.20
Neural Network	92.42	92.74	90.91	92.12

Table 9 Evaluation on Testset

Evaluating parameters on test data					
Model	Training Score	Testing Score	F1 Score	Precision Score	Recall
Decision Tree	95.28%	84.56%	83.84%	84.64%	83.77%
K- Nearest Neighbour	98.71%	91.11%	90.86%	91.69%	90.64%
Logistic Regression	99.07%	95.66%	95.64%	95.91%	95.54%
Support Vector Machine	98.93%	93.07%	92.93%	93.37%	92.76%
Random Forest	1.0%	92.80%	92.61%	92.90%	92.49%
Neural Network	99.31%	93.79%	93.82%	94.51%	93.68%

Appendix B – User Manual

WHEN RUNNING THE PROGRAM, YOU MUST HAVE THE MOST LATEST VERSION OF PYTHON INSTALLED ALONGSIDE ALL OTHER LIBRARIED SPECIFIED IN THIS MANUAL.

MAC

As pointed out above alongside the latest version of Python all libraries used in this project must be installed. This can be done using the following command in terminal:

pip install LIBRARY

LIBRARY must be replaced with the library you wish to install. The list of libraries alongside what chapter they refer to can be found below:

- Chapter 5.2

```
import pandas
import matplotlib.pyplot as plt
```

- Chapter 5.3

```
import pandas as pd
from sklearn import preprocessing
```

- Chapter 5.4

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from matplotlib import pyplot as plt
from sklearn.decomposition import PCA
import numpy as np
from sklearn.manifold import TSNE
```

- Chapter 5.5

```
import warnings
import pandas as pd
from sklearn.metrics import
accuracy_score,f1_score,recall_score,precision_score
from matplotlib import pyplot as plt
import numpy as np
from sklearn.tree import *
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score, KFold
```

- Chapter 5.6

```
import warnings

import pandas as pd
from matplotlib import pyplot as plt
import numpy as np
```

```
from sklearn.tree import *
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import  GridSearchCV
from sklearn_evaluation import plot
from pandas import ExcelWriter
```

- Chapter 5.7

```
import warnings

import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.tree import *
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, f1_score, precision_score,
recall_score
```

Once all libraries are installed the process of running each Python script can begin.

Process

1. Open Terminal on your Mac
2. Navigate to the file directory containing this program using the command below:

cd FILEPATH

Where **FILEPATH** is you must replace this with the file path where the directory is located for example: **cd/ Desktop/University/FYP/B722100-COC257-2**

3. Once the directory for this project has been located run any of the Python scripts using the command below:

python3 SCRIPTNAME

Where **SCRIPTNAME** is the name of the script to run. The different sections with their **SCRIPTNAME** can be found below:

- Loading in data **01LoadData.py**
- Descriptive Stats **02DescriptiveStats.py**
- Data Exploration **03DataExploration.py**
- Data Pre-Processing – Training Set **04DataPre-Processing.py**
- Data Pre-Processing – Test Set **04DataPre-ProcessingTestset.py**

- Data Visualisation PCA **05DataVisualizationPCA.py**
- Data Visualisation TSNE **05DataVisualizationTSNE.py**
- Train Test Split **06TrainWithTrain-Test-Split.py**
- K- Fold cross validation **07TrainWithKFOLD.py**
- Hyperparameter Tuning with GridSearchCV **08Train_Different_Parameters.py**
- Evaluation on Test Set **09EvaluateonTest Set.py**
- API **basic-app.py**

4. To run the API server, you must navigate to the Deployment Directory. For example: **cd/ Desktop/University/FYP/B722100-COC257-2/Deployment** and enter the command below:

uvicorn basic-app:app --reload

HOPE YOU ENJOY!

Glossary

PCA - Principal Component Analysis
SVM – Support Vector Machine
KNN – K Nearest Neighbours
JSON - JavaScript Object Notation
CSV – Comma Separated Values
TSNE - T-Distributed Stochastic Neighbour Embedding
SAG - Stochastic Average Gradient descent
SAGA- Variant of Stochastic Average Gradient descent
RF – Random Forest
ML – Machine Learning
AI – Artificial Intelligence
LR – Logistic Regression

Bibliography

(5) (PDF) Machine Learning Methods for Classifying Human Physical Activity from On-Body Accelerometers. (2021). *ResearchGate*. [online] Available at: https://www.researchgate.net/publication/51970052_Machine_Learning_Methods_for_Classifying_Human_Physical_Activity_from_On-Body_Accelerometers [Accessed 5 Apr. 2021].

(5) (PDF) The effect of gamma value on support vector machine performance with different kernels. (2020). *ResearchGate*. [online] Available at: https://www.researchgate.net/publication/344458945_The_effect_of_gamma_value_on_support_vector_machine_performance_with_different_kernels [Accessed 16 Apr. 2021].

AAL Programme. (2021). *AAL Home 2020 - AAL Programme*. [online] Available at: http://www.aal-europe.eu/ [Accessed 20 Apr. 2021].

Acm.org. (2020a). *Understanding and Visualizing Data Iteration in Machine Learning*. [online] Available at: https://dl.acm.org/doi/fullHtml/10.1145/3313831.3376177 [Accessed 7 Apr. 2021].

Acm.org. (2020b). *Understanding and Visualizing Data Iteration in Machine Learning*. [online] Available at: https://dl.acm.org/doi/fullHtml/10.1145/3313831.3376177 [Accessed 21 Apr. 2021].

Adarsh Menon (2018). *Linear Regression using Gradient Descent - Towards Data Science*. [online] Medium. Available at: https://towardsdatascience.com/linear-regression-using-gradient-descent-97a6c8700931 [Accessed 4 Dec. 2020].

Aggarwal, C., Hinneburg, A. and Keim, D. (n.d.). *On the Surprising Behavior of Distance Metrics in High Dimensional Space*. [online] . Available at: https://bib.dbvis.de/uploadedFiles/155.pdf.

Agilebusiness.org. (2021). *Chapter 10: MoSCoW Prioritisation*. [online] Available at: https://www.agilebusiness.org/page/ProjectFramework_10_MoSCoWPrioritisation [Accessed 20 Apr. 2021].

AltexSoft. (2019). *What is API: Definition, Types, Specifications, Documentation*. [online] Available at: https://www.altexsoft.com/blog/engineering/what-is-api-definition-types-specifications-documentation/ [Accessed 24 Apr. 2021].

Alto, V. (2019). *Neural Networks: parameters, hyperparameters and optimization strategies*. [online] Medium. Available at: https://towardsdatascience.com/neural-networks-parameters-hyperparameters-and-optimization-strategies-3f0842fac0a5 [Accessed 14 Jan. 2021].

Analytics Vidhya (2020). *Precision vs Recall | Precision and Recall Machine Learning*. [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2020/09/precision-recall-machine-learning/ [Accessed 25 Apr. 2021].

AnalyticsSteps. (2020). *How Does K-nearest Neighbor Works In Machine Learning Classification Problem? | Analytics Steps*. [online] Available at: https://www.analyticssteps.com/blogs/how-does-k-nearest-neighbor-works-machine-learning-classification-problem [Accessed 31 Oct. 2020].

Aurélien Géron (2017). *Hands-on machine learning with Scikit-Learn and TensorFlow concepts, tools, and techniques to build intelligent systems*. OREILLY.

Bassey, P. (2019). *Logistic Regression Vs Support Vector Machines (SVM)*. [online] Medium. Available at: https://medium.com/axum-labs/logistic-regression-vs-support-vector-machines-svm-c335610a3d16 [Accessed 4 Nov. 2020].

by RJ Reinhart (2018). *Most Americans Already Using Artificial Intelligence Products*. [online] Gallup.com. Available at: https://news.gallup.com/poll/228497/americans-already-using-artificial-intelligence-products.aspx [Accessed 13 Jan. 2021].

chinmay das (2017). *What is machine learning and types of machine learning — Part-1*. [online] Medium. Available at: https://towardsdatascience.com/what-is-machine-learning-and-types-of-machine-learning-andrews-machine-learning-part-1-9cd9755bc647 [Accessed 14 Jan. 2021].

Cmu.edu. (2021a). *Cross Validation*. [online] Available at: https://www.cs.cmu.edu/~schneide/tut5/node42.html [Accessed 9 Apr. 2021].

Cmu.edu. (2021b). *Cross Validation*. [online] Available at: https://www.cs.cmu.edu/~schneide/tut5/node42.html [Accessed 22 Apr. 2021].

Cogito (2019). *How Much Training Data is Required for Machine Learning Algorithms?* [online] Cogito. Available at: https://www.cogitotech.com/blog/how-much-training-data-is-required-for-machine-learning-algorithms/ [Accessed 7 Apr. 2021].

GeeksforGeeks. (2017a). *K-Nearest Neighbours - GeeksforGeeks*. [online] Available at: https://www.geeksforgeeks.org/k-nearest-neighbours/ [Accessed 31 Oct. 2020].

GeeksforGeeks. (2017b). *Regression and Classification | Supervised Machine Learning - GeeksforGeeks*. [online] Available at: https://www.geeksforgeeks.org/regression-classification-supervised-machine-learning/#:~:text=A%20regression%20problem%20is%20when,which%20goes%20through%20the%20points. [Accessed 5 Apr. 2021].

GeeksforGeeks. (2018). *ML | Label Encoding of datasets in Python - GeeksforGeeks*. [online] Available at: https://www.geeksforgeeks.org/ml-label-encoding-of-datasets-in-python/#:~:text=Label%20Encoding%20refers%20to%20converting,structured%20dataset%20in%20supervised%20learning. [Accessed 7 Apr. 2021].

https://www.facebook.com/MachineLearningMastery (2018). *How to Calculate Principal Component Analysis (PCA) from Scratch in Python*. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/calculate-principal-component-analysis-scratch-python/#:~:text=An%20important%20machine%20learning%20method,same%20number%20or%20fewer%20dimensions. [Accessed 22 Feb. 2021].

https://www.facebook.com/MachineLearningMastery (2019). *A Gentle Introduction to Cross-Entropy for Machine Learning*. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/cross-entropy-for-machine-learning/#:~:text=Cross%2Dentropy%20is%20a%20measure%20of%20the%20difference%20between%20two,encode%20and%20transmit%20an%20event. [Accessed 4 Jan. 2021].

IBM Cloud Education (2020). *What is Machine Learning?* [online] Ibm.com. Available at: https://www.ibm.com/cloud/learn/machine-learning [Accessed 14 Jan. 2021].

IBM Developer. (2017a). *Supervised learning models*. [online] Available at: https://developer.ibm.com/articles/cc-supervised-learning-models/ [Accessed 14 Jan. 2021].

IBM Developer. (2017b). *Unsupervised learning for data classification*. [online] Available at: https://developer.ibm.com/articles/cc-unsupervised-learning-data-classification/ [Accessed 14 Jan. 2021].

Jahnavi Mahanta (2017). *Introduction to Neural Networks, Advantages and Applications*. [online] Medium. Available at: https://towardsdatascience.com/introduction-to-neural-networks-advantages-and-applications-96851bd1a207#:~:text=Key%20advantages%20of%20neural%20Networks%3A&text=ANNs%20have%20the%20ability%20to,linear%20as%20well%20as%20complex. [Accessed 14 Jan. 2021].

Krish Naik (2020). *How To Deploy Machine Learning Models Using FastAPI-Deployment Of ML Models As API's*. *YouTube*. Available at: https://www.youtube.com/watch?v=b5F667g1yCk&t=600s [Accessed 24 Apr. 2021].

Kwon, M.-C. and Choi, S. (2018). Recognition of Daily Human Activity Using an Artificial Neural Network and Smartwatch. *Wireless Communications and Mobile Computing*, [online] 2018, pp.1–9. Available at: https://www.hindawi.com/journals/wcmc/2018/2618045/ [Accessed 5 Apr. 2021].

LearnCode.academy (2018). *Neural Networks Explained - Machine Learning Tutorial for Beginners*. *YouTube*. Available at: https://www.youtube.com/watch?v=GvQwE2OhL8I&t=524s [Accessed 14 Jan. 2021].

Leonel, J. (2019). *Hyperparameters in Machine /Deep Learning - Jorge Leonel - Medium*. [online] Medium. Available at: https://medium.com/@jorgesleonel/hyperparameters-in-machine-deep-learning-ca69ad10b981 [Accessed 14 Jan. 2021].

Machine Learning Algorithms for Human Activity Recognition. (2019). *International Journal of Innovative Technology and Exploring Engineering*, 8(12S), pp.401–403.

Medium. (2019). *Medium*. [online] Available at: https://medium.com/@zxr.nju/what-is-the-kernel-trick-why-is-it-important-98a98db0961d [Accessed 4 Nov. 2020].

Ons.gov.uk. (2018). *Ageing - Office for National Statistics*. [online] Available at: https://www.ons.gov.uk/peoplepopulationandcommunity/birthsdeathsandmarriages/ageing [Accessed 20 Apr. 2021].

Outsource2india.com. (2021). *Machine Learning & its Applications - Outsource2india*. [online] Available at: https://www.outsource2india.com/software/articles/machine-learning-applications-how-it-works-who-uses-it.asp [Accessed 13 Jan. 2021].

OxfordSparks (2017). *What is Machine Learning? YouTube*. Available at: https://www.youtube.com/watch/f_uwKZIAeM0 [Accessed 14 Jan. 2021].

Prashant Gupta (2017). *Decision Trees in Machine Learning - Towards Data Science*. [online] Medium. Available at: https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052 [Accessed 5 Nov. 2020].

Pydata.org. (2013). *IO tools (text, CSV, HDF5, …) — pandas 1.2.3 documentation*. [online] Available at: https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html [Accessed 6 Apr. 2021].

Pydata.org. (2021). *pandas.DataFrame.info — pandas 1.2.3 documentation*. [online] Available at: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.info.html [Accessed 6 Apr. 2021].

Raheel Shaikh (2018). *Choosing the right Encoding method-Label vs OneHot Encoder*. [online] Medium. Available at: https://towardsdatascience.com/choosing-the-right-encoding-method-label-vs-onehot-encoder-a4434493149b [Accessed 7 Apr. 2021].

Rana, J., Jha, T. and Shetty, R. (n.d.). *Applications of Machine Learning Techniques in Human Activity Recognition*. [online] . Available at: https://arxiv.org/pdf/1510.05577.pdf.

Ravindra Parmar (2018). *Common Loss functions in machine learning - Towards Data Science*. [online] Medium. Available at: https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23 [Accessed 14 Jan. 2021].

Readthedocs.io. (2014). *Linear Regression — ML Glossary documentation*. [online] Available at: https://ml-cheatsheet.readthedocs.io/en/latest/linear_regression.html#:~:text=Linear%20Regression%20is%20a%20supervised,Simple%20regression [Accessed 30 Oct. 2020].

Saishruthi Swaminathan (2018). *Logistic Regression — Detailed Overview - Towards Data Science*. [online] Medium. Available at: https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc [Accessed 4 Nov. 2020].

Science in the News. (2017). *The History of Artificial Intelligence - Science in the News*. [online] Available at: http://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/#:~:text=It%20began%20with%20the%20%E2%80%9Cheartless,culturally%20assimilated%20in%20their%20minds [Accessed 14 Jan. 2021].

Sciencedirect.com. (2017). *Machine Learning - an overview | ScienceDirect Topics*. [online] Available at: https://www.sciencedirect.com/topics/psychology/machine-learning [Accessed 14 Jan. 2021].

Scikit-learn.org. (2014). *sklearn.manifold.TSNE — scikit-learn 0.24.1 documentation*. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html [Accessed 8 Apr. 2021].

Scikit-learn.org. (2015). *sklearn.neural_network.MLPClassifier — scikit-learn 0.24.1 documentation*. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier [Accessed 23 Apr. 2021].

Scikit-learn.org. (2020). *sklearn.model_selection.train_test_split — scikit-learn 0.24.1 documentation*. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html [Accessed 8 Apr. 2021].

Shiva Verma (2019). *Understanding different Loss Functions for Neural Networks*. [online] Medium. Available at: https://towardsdatascience.com/understanding-different-loss-functions-for-neural-networks-dd1ed0274718 [Accessed 14 Jan. 2021].

Tatan, V. (2019). *Your Beginner Guide to Basic Classification Models: Logistic Regression and SVM*. [online] Medium. Available at: https://towardsdatascience.com/your-beginner-guide-to-basic-classification-models-logistic-regression-and-svm-b7eef864ec9a [Accessed 4 Nov. 2020].

The Machine Learning Process | Codecademy (2021). *The Machine Learning Process | Codecademy*. [online] Codecademy. Available at: https://www.codecademy.com/articles/the-ml-process#:~:text=The%20Process,spitting%20out%20predictions%20and%20insights.&text=Finding%20and%20Understanding%20the%20Data,the%20Data%20and%20Feature%20Engineering [Accessed 5 Apr. 2021].

Towards AI Team (2020). *Decision Trees Explained With a Practical...* [online] Towards AI — The Best of Tech, Science, and Engineering. Available at: https://towardsai.net/p/programming/decision-trees-explained-with-a-practical-example-fe47872d3b53 [Accessed 5 Nov. 2020].

UCI Machine Learning (2012). *Human Activity Recognition with Smartphones*. [online] Kaggle.com. Available at: https://www.kaggle.com/uciml/human-activity-recognition-with-smartphones [Accessed 6 Apr. 2021].

Ucla.edu. (2020). *Multivariate Regression Analysis | Stata Data Analysis Examples*. [online] Available at: https://stats.idre.ucla.edu/stata/dae/multivariate-regression-analysis/ [Accessed 31 Oct. 2020].

Varghese, D. (2018a). *Comparative Study on Classic Machine learning Algorithms*. [online] Medium. Available at: https://towardsdatascience.com/comparative-study-on-classic-machine-learning-algorithms-24f9ff6ab222 [Accessed 5 Jan. 2021].

Varghese, D. (2018b). *Comparative Study on Classic Machine learning Algorithms*. [online] Medium. Available at: https://towardsdatascience.com/comparative-study-on-classic-machine-learning-algorithms-24f9ff6ab222 [Accessed 5 Jan. 2021].

Varghese, D. (2018c). *Comparative Study on Classic Machine learning Algorithms , Part-2*. [online] Medium. Available at: https://medium.com/@dannymvarghese/comparative-study-on-classic-machine-learning-algorithms-part-2-5ab58b683ec0 [Accessed 8 Jan. 2021].

Veit, A. (2019). *Why ADAM Beats SGD for Attention Models*. [online] OpenReview. Available at: https://openreview.net/forum?id=SJx37TEtDH [Accessed 16 Apr. 2021].

Wikipedia Contributors (2020a). *Hyperparameter (machine learning)*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Hyperparameter_(machine_learning) [Accessed 14 Jan. 2021].

Wikipedia Contributors (2020b). *ID3 algorithm*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/ID3_algorithm [Accessed 5 Jan. 2021].

Wood, T. (2019). *F-Score*. [online] DeepAI. Available at: https://deepai.org/machine-learning-glossary-and-terms/f-score [Accessed 8 Apr. 2021].

Yiu, T. (2019). *Understanding Random Forest - Towards Data Science*. [online] Medium. Available at: https://towardsdatascience.com/understanding-random-forest-58381e0602d2 [Accessed 5 Nov. 2020].

GeeksforGeeks. (2020). *Deploying ML Models as API using FastAPI - GeeksforGeeks*. [online] Available at: https://www.geeksforgeeks.org/deploying-ml-models-as-api-using-fastapi/ [Accessed 4 May 2021].