Using IoT to simulate a parking bay and provide real time updates via an app based on external factors

Student: Tarikh Chouhan

Supervisor: Dimitris Dracopolous

Institute: University of Westminster

# Introduction

(Talk about the rise of cars and AI, how it’ll impact road and road users, congestion etc.

Get rid of sub headings below?

)

This section of the report will cover the general scope of the project and what problem the project aims to tackle.

As car manufacturers continuously unveil new cars to the public every year and as the world’s population continues to grow, a trend can be seen with these two factors. There are more cars appearing on the road every year due to growing population[[1]](#footnote-1). This trend can have negative repercussions in terms of environmental and financial factors (e.g. more money being spent on petrol, insurance claims increasing, pollution, hazardous chemicals entering the atmosphere). However there are positive effects to this trend such as new technology emerging through innovative solutions and moving towards a society revolving around self-driving cars.[[2]](#footnote-2) Due to the trend outlined, finding parking bays to park in will be a challenge as more cars will be on the road hence this project proposes a solution to combat this problem.

This project will be comprised of chapters and segments and each chapter will individually contain highly detailed information in order to fully understand this report. The report will go through an overview of a mathematical concept which is seen every day in our lives and has only became popular in the last 100 years; queueing theory. Furthermore, it will delve a bit into traffic engineering and seeing what’s already in place at the moment. As well that, this report will contain feedback and information I have gathered from companies revolving around transport engineering. This report will also delve into an interesting and highly sophisticated part of computer science; machine learning, as this report will outline an overview of what is essentially machine learning and AI as well as discussing the model I have chosen to use in my solution. Moving onto the technical aspect, this report will show the tech stack behind the proposed solution as well as explaining the choice for the chosen technologies. Diving deeper, it will show the source code behind the solution and explain concepts that might not be familiar with university students with such as dependency injection, the maven build life cycle as well as using GIT for source control.

## Current and future problems of not finding parking bays efficiently – LR?

As more cars will be on the road, available parking bays will be less frequent which in turn would frustrate drivers as they look for an available bay. As a result of this frustration, drivers tend to park illegally and end up having to pay a penalty/fine. Local councils are generating massive amounts of revenue by handing out parking fines. The following statistics paint a picture on how significant the car parking industry is:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Surplus in £(millions) made in parking fines per local council [[3]](#footnote-3) | | | | | |
| Local authority | **2010/11** | **2011/12** | **2012/13** | **2013/14** | **2014/15** |
| Westminster | 38.2 | 41.6 | 39.7 | 51 | 46.4 |
| Kensington & Chelsea | 21.1 | 28.1 | 30.4 | 33.5 | 33 |
| Camden | 21.1 | 25 | 23.5 | 24.9 | 24.5 |
| Hammersmith & Fulham | 16.6 | 19.5 | 19.4 | 23 | 23.8 |
| Wandsworth | 14.4 | 16.1 | 15.9 | 19.7 | 20.4 |
| Brighton & Hove UA | 12.7 | 14.4 | 16.3 | 18.1 | 18.6 |
| Haringey | 3.3 | 5.3 | 5.2 | 5.7 | 16.1 |
| Islington | 5.6 | 10.9 | 8.2 | 10.4 | 13.7 |
| Nottingham City UA | 3.7 | 3.3 | 11.8 | 12.1 | 13.3 |
| Hackney | 4.7 | 5.9 | 7.8 | 8.2 | 10.8 |
| Brent | 3.9 | 2.7 | 2.7 | 8.3 | 10.5 |
| Tower Hamlets | 6 | 5.8 | 7 | 8.3 | 10 |
| Birmingham | 5.1 | 5.5 | 6.9 | 7.8 | 9.7 |
| Lambeth | 7 | 5.8 | 12 | 7.2 | 9.7 |
| Milton Keynes UA | 6 | 6.6 | 6.7 | 8.2 | 9 |
| Cornwall UA | 8.2 | 7.9 | 8.1 | 8 | 8.7 |
| Manchester | 1.9 | 6.3 | 8.8 | 8 | 7.9 |
| Hounslow | 6 | 7.3 | 6.4 | 7.8 | 7.7 |
| Newham | 3.9 | 7.3 | 8.2 | 7.2 | 7.3 |
| Merton | 4.4 | 5.7 | 6.9 | 7 | 7.2 |

Furthermore, unable to find available parking bays could have negative repercussions on a global scale such as the increase of greenhouse gas since harmful emissions would be emitted from the car as it spends more time and fuel to look for a parking bay having the driver arrive at its destination. As well as that, driving around to look for a parking bay will use up fuel hence the driver would need to spend money to fill their cars more frequently compared to finding a parking bay that is readily available.

## Current solutions to the problem

# Literature Review

There have been multiple documents produced and published regarding the issue around car parks; whether it is the mathematics behind it or a simulating the construction of a parking lot. As well as that there are documents providing in depth articles from well-known established bodies such as the RAC foundation. These documents can vary from articles and publications to thesis’s. In this section of the report, you will be updated and be equipped with the latest works currently being undertaken in the parking community within traffic engineering as well as getting a brief overview of the mathematics behind it.

Queuing Theory –

Queues are everywhere. They can be observed in the most obvious places such as customers lining up to pay for their goods in a shop, patients being on a waiting list to see the GP or drivers waiting their turn to fill up their car in a petrol station. Queues can also be found in places where the average person wouldn’t typically realise they’d find a queue as instructions being executed on a CPU or sending and receiving packets of data to browse the internet. Queuing theory was first written by Danish mathematician, Agner Krarup Erlang, back in 1909. Agner worked at a telephone exchange which consisted of using jack plugs and plugging them into a circuit to route phone calls. Agner wanted to know how many circuits was needed to provide a sufficient service to a local village and thus began researching and then published his findings in the paper ‘The Theory of Probabilities and Telephone Conversations’[[4]](#footnote-4).

Queueing theory in its simplest form, deals with problems involved with queues or waiting. Most problems regarding this concept have 2 entities in common; ‘queue’ and an ‘activity’. ‘Queue’ is the current wait and ‘activity’ is the server. So a practical situation would be a queue that represented a queue of customers and activity would represent a staff member at the cash till. The staff member deals with the customers one by one effectively taking care of the queue.

There are some characteristics to these entities that are present in every queuing problem. The activity would need to determine on what the queue discipline would be i.e. FIFO (first in first out), LIFO (last in first out) etc. Another concept present in queuing problems is understanding what type of queue are we dealing with. Baulking; where customers decide not to join the queue if it’s too long, reneging; where customers leave the queue if they have waited for too long to be served or jockeying; customers switch between queues if it’ll help them get served quicker. Furthermore, another important variable to consider when dealing with queuing theory is understanding the behaviour of the arrival process. This means understanding how customers would join the queue; e.g. in fixed timed intervals or variable times, would they join it as a group or as a single entity.

‘Applicability of information technologies in parking area capacity optimization’ written by Maršanić Robert and Pupavac Drago is a research paper how to efficiently design parking areas based on waiting-line models; also known as queuing theory[[5]](#footnote-5). In their paper, they were trying to find an efficient model to use for their car park, ‘Delta’ located in a city in Croatia. They compared their findings with different models of car parks i.e. a car park with a single-channel queueing model and a multichannel queueing model and found out that having a single-channel queueing model is not as efficient as having a multichannel queueing model as the single-channel service deteriorates in peak hours as it cannot cater for all the vehicles arriving during peak hours.

Furthermore, the research paper from Shuguo Yang and Xiaoyan Yang titled ‘The Application of the Queuing Theory in the Traffic Flow of Intersection’ delves into the concept of using queueing theory to analyse traffic conditions on an intersection[[6]](#footnote-6) which is similar to analysing car parks as it revolves around the same concept. This paper uses first hand data as they acquire their data from the intersection. This can be seen on Table 1 on their paper[[7]](#footnote-7). By collecting first hand data, the results generated from this paper would be reliable only in the location that they acquired their results from. They use the data to find the average number of cars arriving to the intersection. This paper then goes onto give the reader a clear and concise conclusion by comparing their results from different scenarios i.e. comparing the overall probability that there will be zero cars left in the queue in an intersection with two, three and four lanes. It further enforces the fact that using queuing theory is a sound and practical approach when dealing with vehicles and roads as this model can give a huge insight onto vehicles waiting on the road.

Machine Learning –

With technology rapidly increasing and most of the technology we use are becoming automated, the rise of Artificial Intelligence (AI) is becoming more and more dominant. AI in essence involves machines that behave, e.g. think, like humans[[8]](#footnote-8). There is then a multitude of categories that fall under AI and ‘Machine Learning’ is one of them. Machine learning is all primarily about detecting patterns in data and identifying future patterns based on historic patterns to make a solid prediction. A common example is the autocorrect feature in Google. If you misspell something on Google, Google will suggest the correct word.

Machine learning is on the rise when it is used in conjunction with vehicles. More and more vehicles are incorporating AI into them. An example of this is the Tesla’s autopilot feature. And as the world progresses further, the further we progress up the autonomous levels. We are moving towards a Level 3 autonomous society. Level 3 autonomousity revolves around the car actively scanning and monitoring the environment by using external sensors such as LiDAR, infrared sensors, ultrasonic sensors etc. And quite recently, a vehicle that aims to provide complete level 4 autonomousity was showcased in CES 2018[[9]](#footnote-9). As you can see, vehicles are getting smarter so it only makes sense to make our roads smart too.

There are many algorithms to use when it comes to incorporating machine learning. An important factor in choosing what algorithm to use depends on the data you are dealing with. There are two main types of machine learning algorithms one can use; classification learning and regression learning. Regression learning revolves around continuous data and is usually used in scenarios where a value is to be predicted such as ‘What will the average house price be in 10 years’ time’ or ‘How much will stock X be worth in Y day’s time’. Classification learning on the other hand is more about predicting something that has a binary output e.g. yes/no, 0/1 etc. A typical question that would use the classification learning would be ‘Will it rain today?’ as there are only 2 possible outcomes to this question; yes or no.

However, classification and regression learning algorithms is used if the scenario revolves around ‘supervised learning'. Supervised learning is used when the algorithm is given the inputs and outputs and the algorithm is algorithm is trained to come up with the best prediction. As Bostjan Kaluza wrote in his book, “…given a set D of learning examples described with features, X, the goal of supervised learning is to find a function that predicts a target variable, Y. The function f that describes the relation between features X and class Y is called a model: f(X) -> Y”[[10]](#footnote-10). Essentially, we are supervising the machine to come up with a model.

On the other hand, unsupervised learning revolves around the computer identifying complex patterns with minimal human guidance. A prime example of unsupervised learning can be seen when products are being suggested to you on e-commerce websites, or relevant ads being displayed to you as you surf the web. Unsupervised learning is more complex than supervised learning as you are not explicitly giving the algorithm as much information and thus it’s up to the machine to come up with relevant labels to the data. The machine may do this in a variety of different ways but the common approach is to group the data based on some properties of the data and see which data is plotted next to each other. As these data gets plotted, there will be clusters forming and depending on how dense the cluster is compared to another cluster and how far a cluster is from another cluster, it will associate a label to these clusters. This approach is used in the ‘k-means clustering’ algorithm which is used in unsupervised learning. Essentially, we are not supervising the machine to come up with a model but rather the machine itself comes up with a model.

In this report, logistic regression will be used as it is the most sound and practical approach with the data that we will be dealing with. The reason for this is because logistic regression is typically used when the dependent variables are binary. Logistic regression is used to predict one outcome out of a possible two outcomes. For example, given a scenario where a patient who was being tested for asthma, the only outcomes would either be ‘yes’ or ‘no’. Likewise, in the context of parking bays, the parking bay is either occupied or vacant. There can be no in between. For this reason alone, logistic regression the ideal algorithm to use.

The logistic regression is modelled with the following equation:

Logistic regression uses maximum likelihood estimation (MLE) to obtain the coefficients in the above equation. This can be thought of as fine tuning the model so that the model will be able to give us a clear and more accurate model to use. In this report, I will be using the stochastic gradient descent to obtain the values of the coefficients.

Machine learning data and analysis -

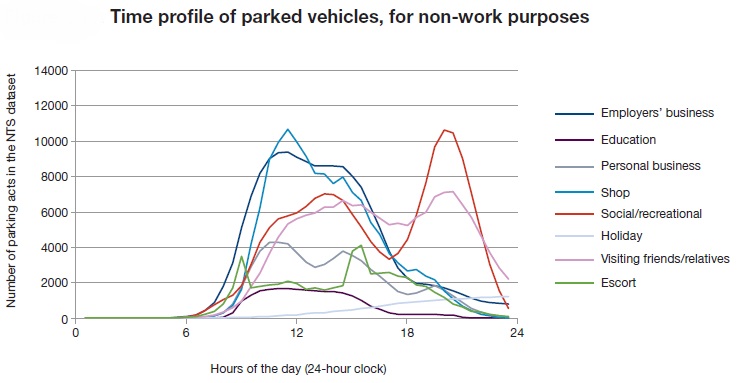
This section of the report will revolve around the data to be used when implementing the machine learning feature of the app.

Referring to the context of parking bay sensors, each sensor will theoretically have its own data of when it is being occupied or vacant. This is because each sensor will be used different depending on where it is located. For example, if a sensor was placed near a school, generally the times that it would be used would range from 3pm to 5pm and if a sensor was placed near a business car park, it the times that it would be in use would vary from 9am to 6pm. Not only the time will make a difference in the data, but the day of the week will also. An example of this is if a parking bay was near a shopping mall, it would be used more heavily on the weekends rather than the weekdays as more people will be out on weekends. It is important to bear in mind of these external social factors. In this report, I will be creating the data for one sensor as generating data for more than sensor will be time consuming. There are a multitude of approaches to creating this sort of data, one is predicting how an individual sensor will be used and creating the data based on it (e.g. if this sensor will be placed in a business car park, expect to see the sensor to show up a ‘occupied’ between the hours of 9am-6pm. Another way of gathering the data to use would be to physically place the sensor on a parking bay. This approach would give the ideal data but has its drawback as the sensor is not built to withstand any weather conditions or intense weight; should a car accidentally draw over it. Finally, a better approach would be to look at a report regarding car parks and base data on the finding of that report which the RAC foundation have done.

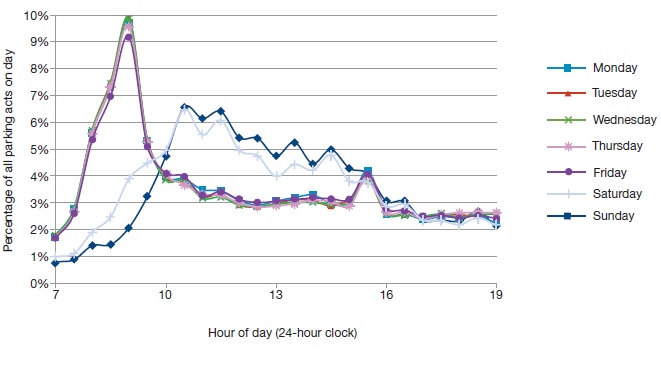
//Introduce report

The RAC Foundation has done intense research and surveys to gather information revolving around car parks. The report has an immense amount of information ranging from supply and demand of car parks to management of parking. It also contains timings on how car parks are being used and for what reason. Below are the graphs corresponding to parking bays from their report.

//introduce graphs



**Variation in start time profile of parking by day of week**



From the graph, we can see a multitude of information which will be relevant to the data to be created to feed to the machine learning algorithm. From the first graph regarding the reason of parked cars, we can deduce that some categories follow the same trend as other categories. You can see this by comparing the trend between ‘shop’ and ‘employers business’. Furthermore, the majority of the trends may have a different pattern to each other but they all usually follow the same downwards trend because after 18:00 hours; ‘Visiting friends/relatives, ‘Escort’, ‘Shop’ and ‘Employers business’ all follow the same downwards trend. And as expected, the social category tends to pick up after 18:00 hours as the report outlines that “…public car parks are especially used by shoppers and those travelling for social and recreational activities…” therefore a lot of people leave work around the 18:00 hours mark and go out and socialise.

Furthermore, as you can see from the second graph, all 5 weekdays follow almost an exact trend as each other whilst the weekends follow the same trend as each other.

//introduce data model structure

Firstly, the data will be created in Microsoft Excel as it is easy to visualise the data in a tabular form as well as creating the data and also most softwares like Matlab will be able import data from this type of file. The data will have 2 corresponding columns; one for time and the other for the status of the bay (occupied/vacant).

Based on the graphs from RAC foundation, the following tables can be deduced to feed the algorithms:

Table regarding shopping hours:

|  |  |
| --- | --- |
| Time (hours) | Status |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 1 |
| 8 | 1 |
| 9 | 1 |
| 10 | 1 |
| 11 | 1 |
| 12 | 1 |
| 13 | 1 |
| 14 | 0 |
| 15 | 0 |
| 16 | 0 |
| 17 | 0 |
| 18 | 1 |
| 19 | 0 |
| 20 | 0 |
| 21 | 0 |
| 22 | 0 |
| 23 | 0 |
| 24 | 0 |

Table regarding social hours:

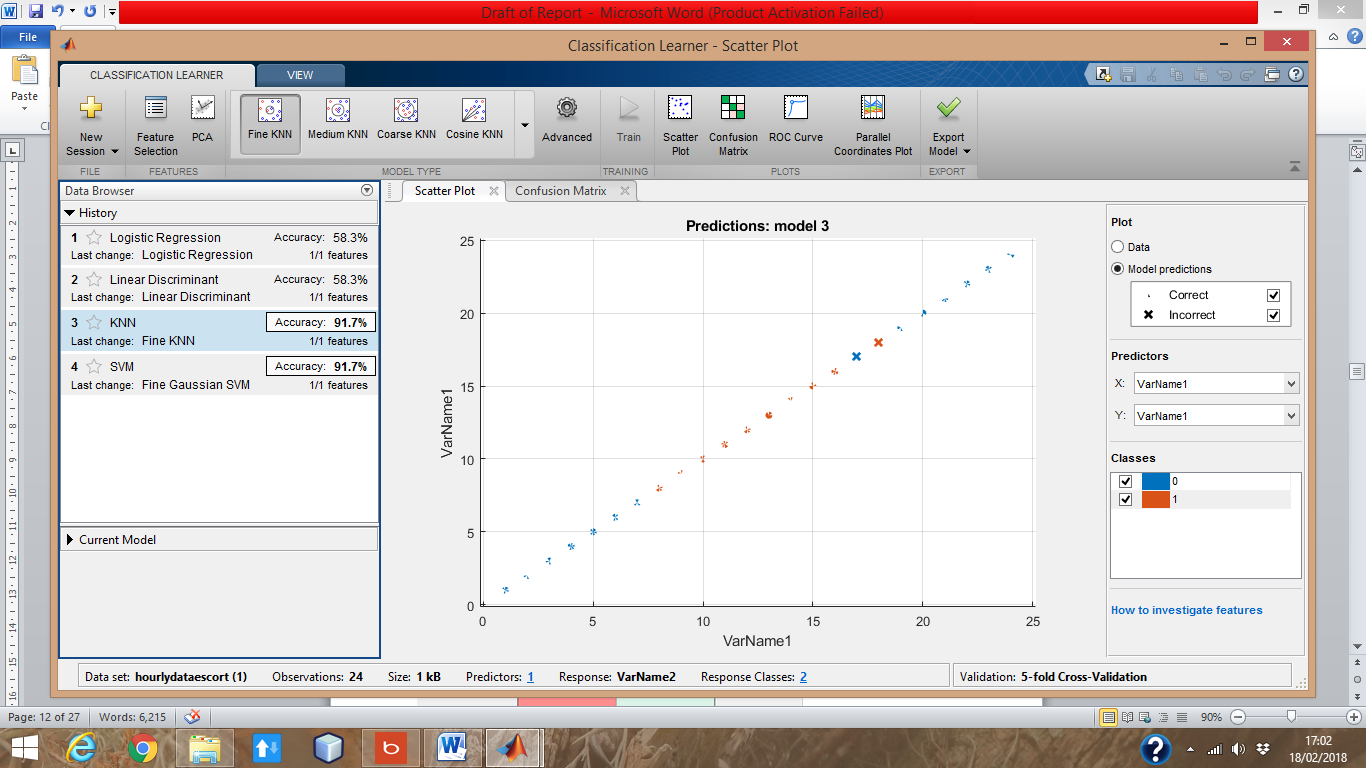
|  |  |
| --- | --- |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 0 |
| 8 | 0 |
| 9 | 0 |
| 10 | 1 |
| 11 | 1 |
| 12 | 1 |
| 13 | 1 |
| 14 | 1 |
| 15 | 0 |
| 16 | 0 |
| 17 | 0 |
| 18 | 0 |
| 19 | 1 |
| 20 | 1 |
| 21 | 1 |
| 22 | 1 |
| 23 | 0 |
| 24 | 0 |

Table regarding escort:

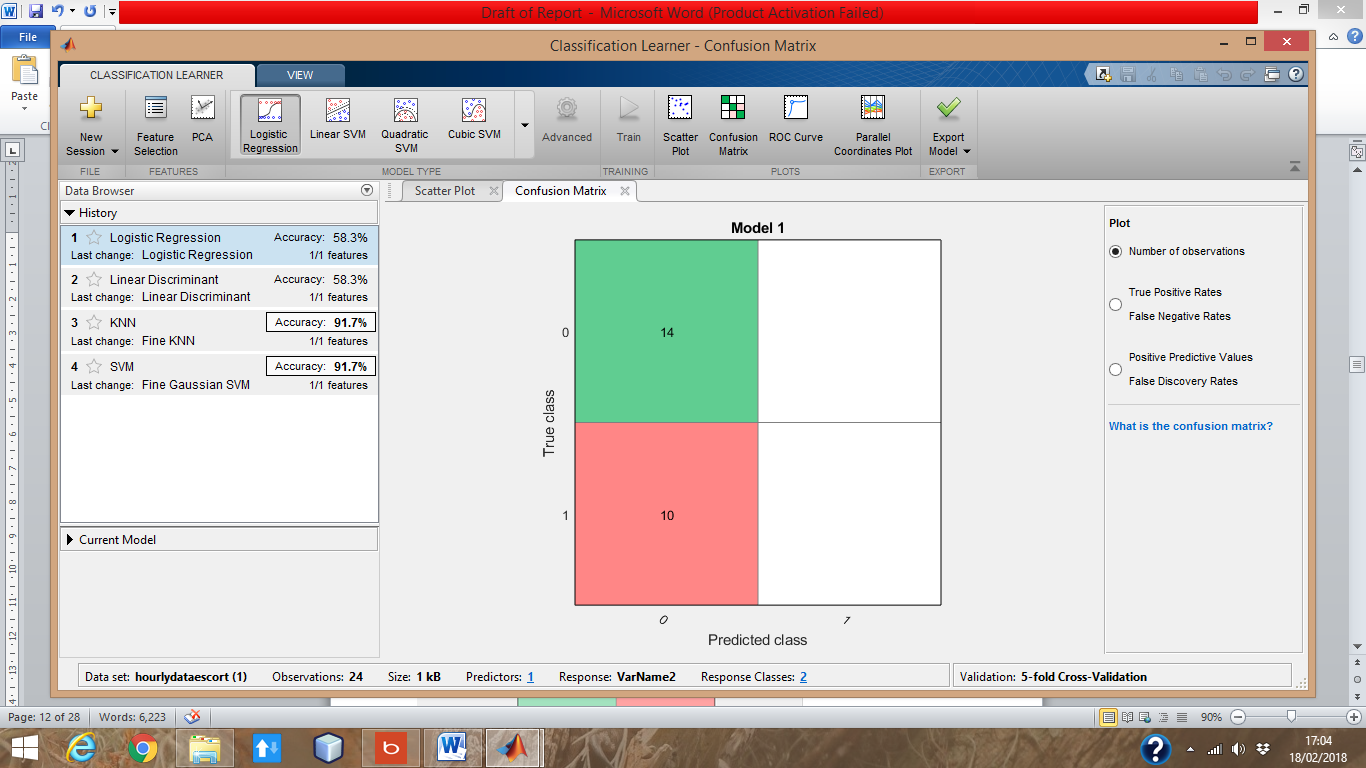
|  |  |
| --- | --- |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 0 |
| 8 | 1 |
| 9 | 1 |
| 10 | 1 |
| 11 | 1 |
| 12 | 1 |
| 13 | 1 |
| 14 | 1 |
| 15 | 1 |
| 16 | 1 |
| 17 | 1 |
| 18 | 0 |
| 19 | 0 |
| 20 | 0 |
| 21 | 0 |
| 22 | 0 |
| 23 | 0 |
| 24 | 0 |

//introduce matlab graphs and analysis

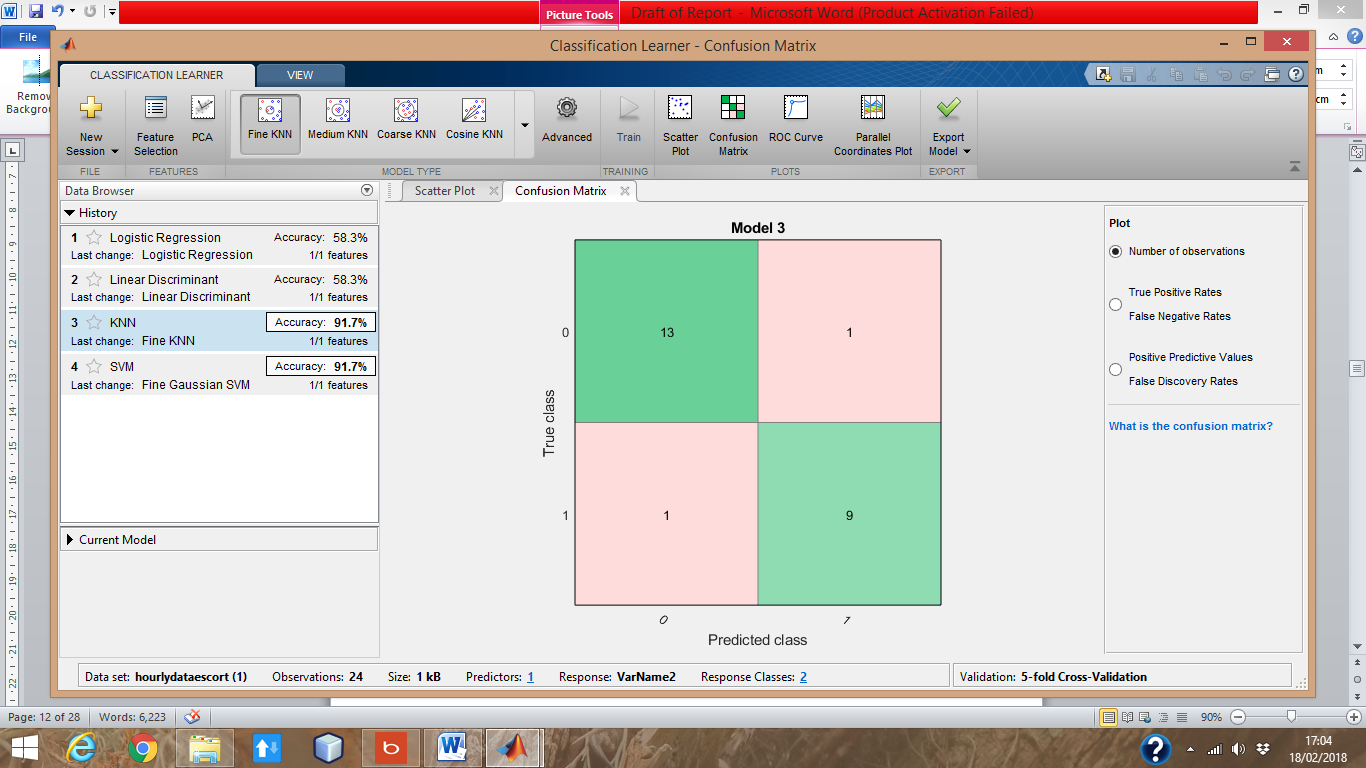
Once we feed these tables into Matlab, we can use the ‘Classification Learner’ app and look at the different algorithms that perform on this data. Furthermore, we can critically analyse the performance of the learning algorithm and decide what algorithm should be implemented in the app to predict the availability of parking bays. There is no right algorithm because every algorithm has its strength and weaknesses ranging from different factors from computability and complexity.

 <- Algorithms being tested on hourly escort data

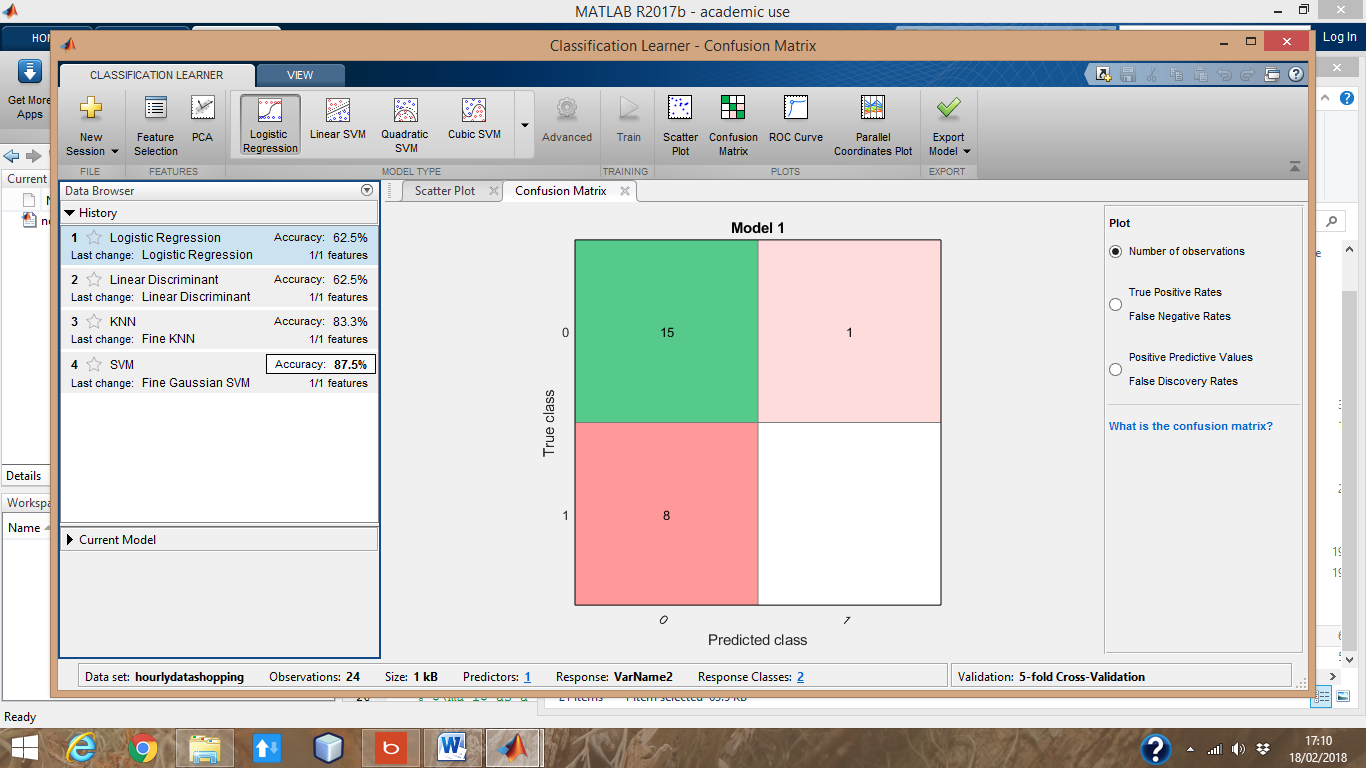
Confusion matrix on logistic regression. Hourly escort data (below).



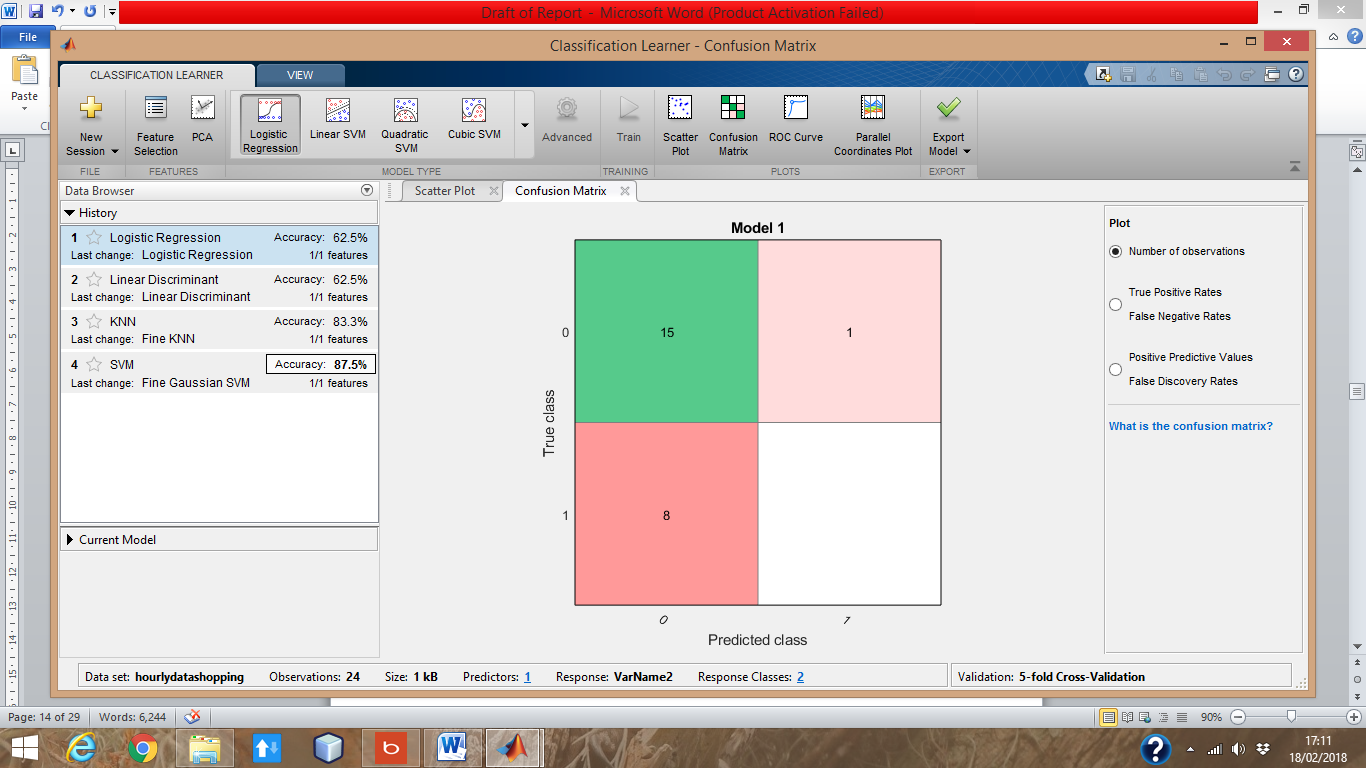
Confusion matrix on KNN. Hourly escort data (below).



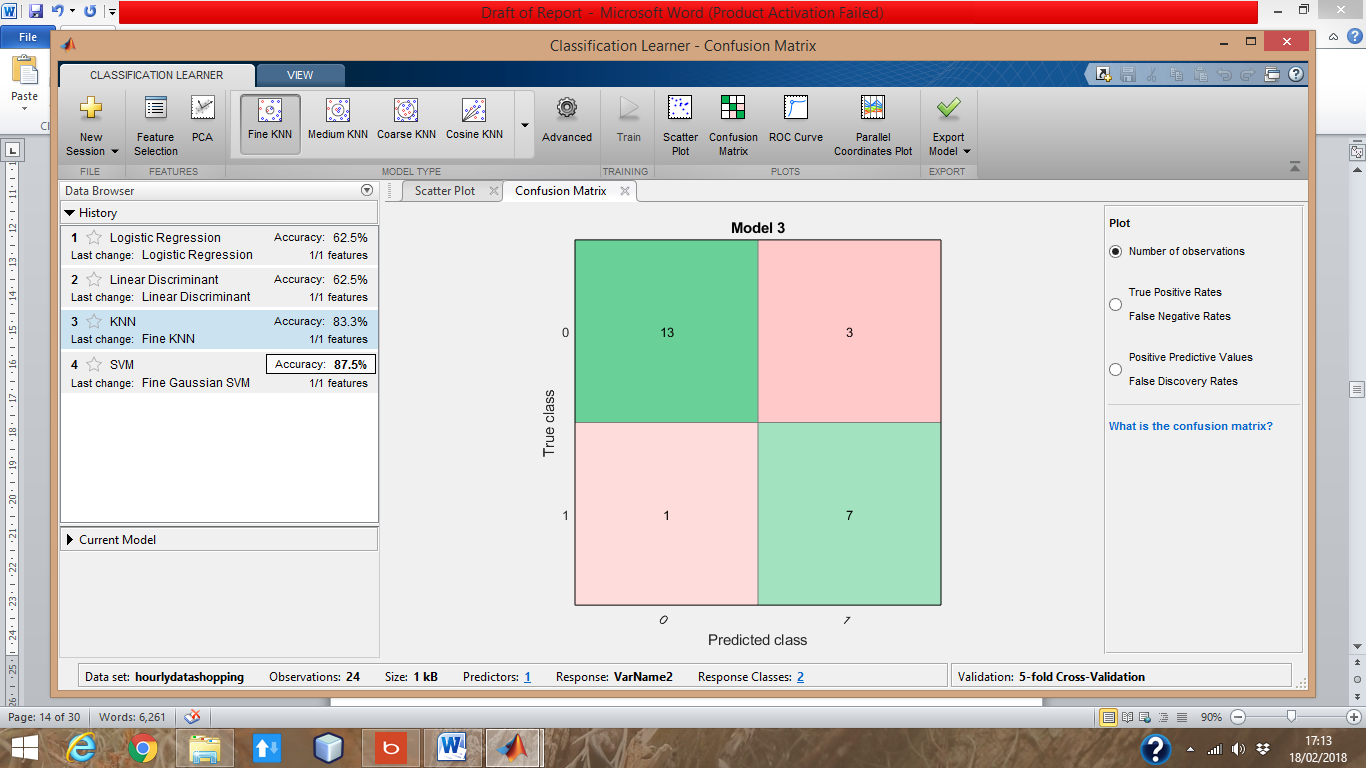
Algorithms being tested on hourly shopping data (below).



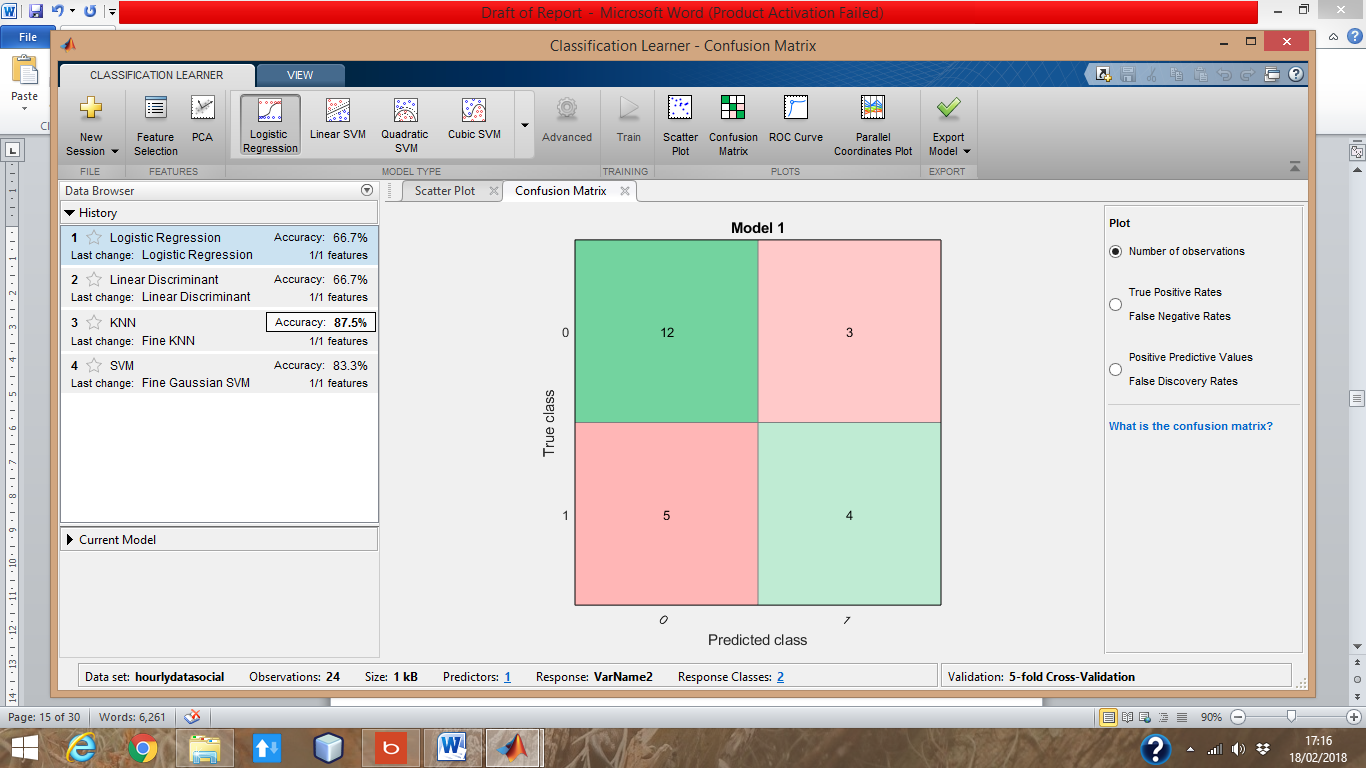
Confusion matrix on logistic regression. Hourly shopping data. (below).



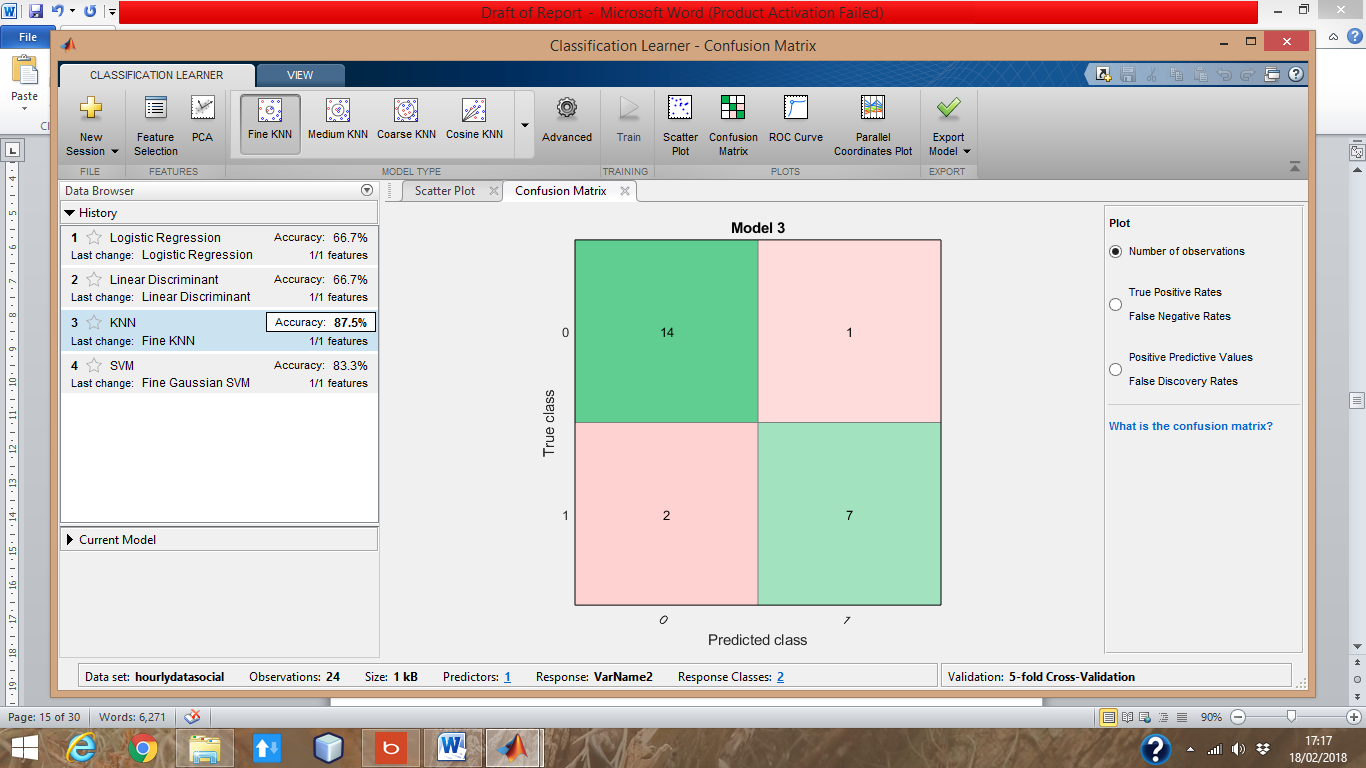
Confusion matrix on KNN. Hourly shopping data (below).



Confusion matrix on logistic regression. Based on hourly social data.

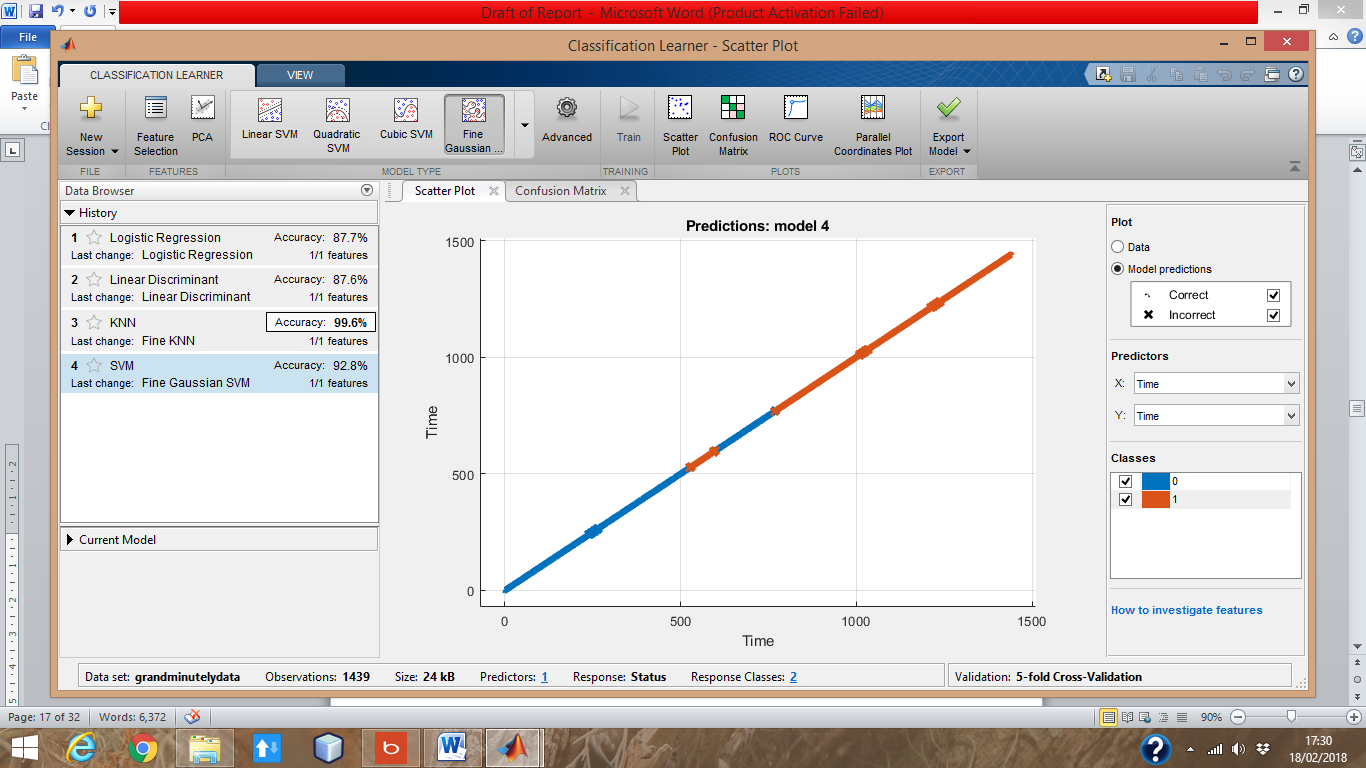


Confusion matrix on KNN. Based on hourly social data.



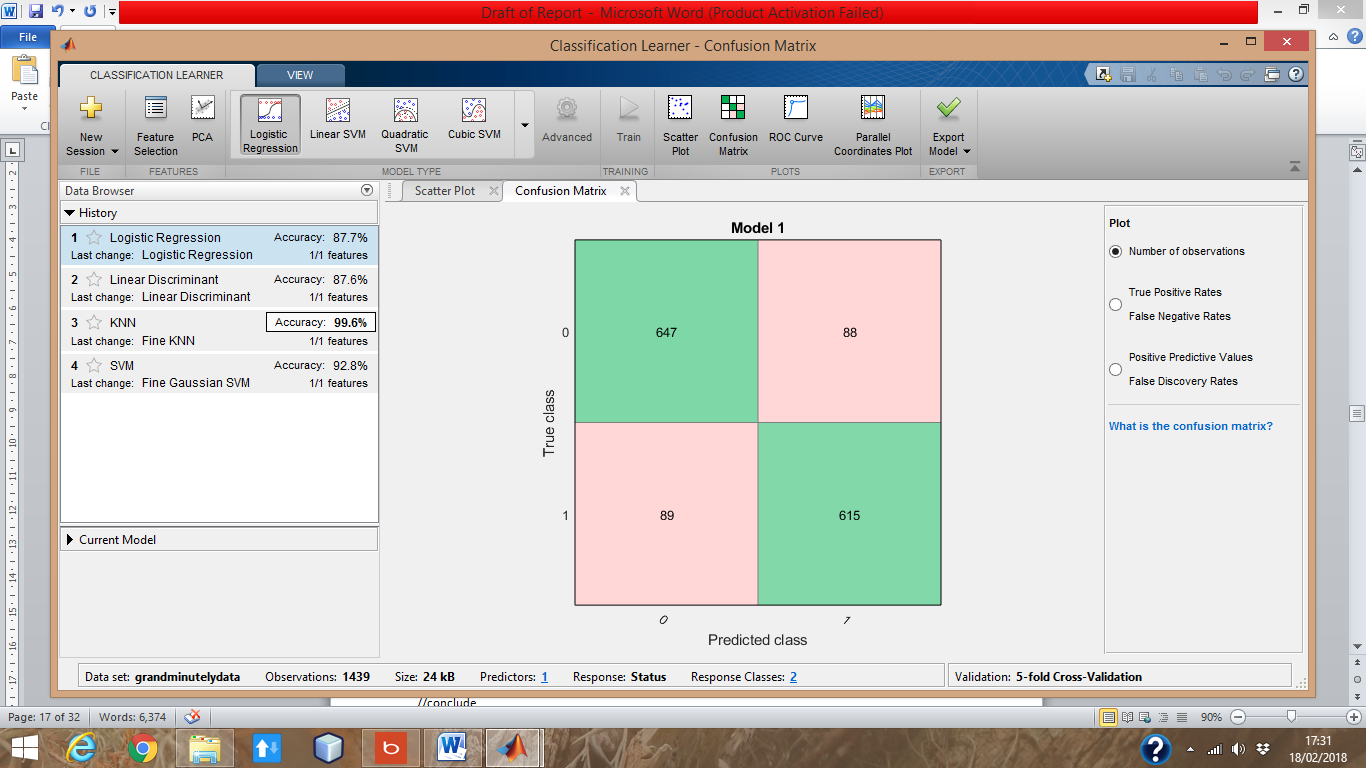
Furthermore, I have created table that ranges from 1-1440 which these numbers corresponds to minutes past 12:00am. I have also made the table follow a linear trend; i.e. the parking bay starts of as vacant but as time passes throughout the day, it gradually becomes occupied. The reason for including the independent variable as minutes is because the sensor will constantly send out signals every few seconds so it’s essential to gather data every time it occurs.

The table is too big to display but these are the results and graph of the findings:

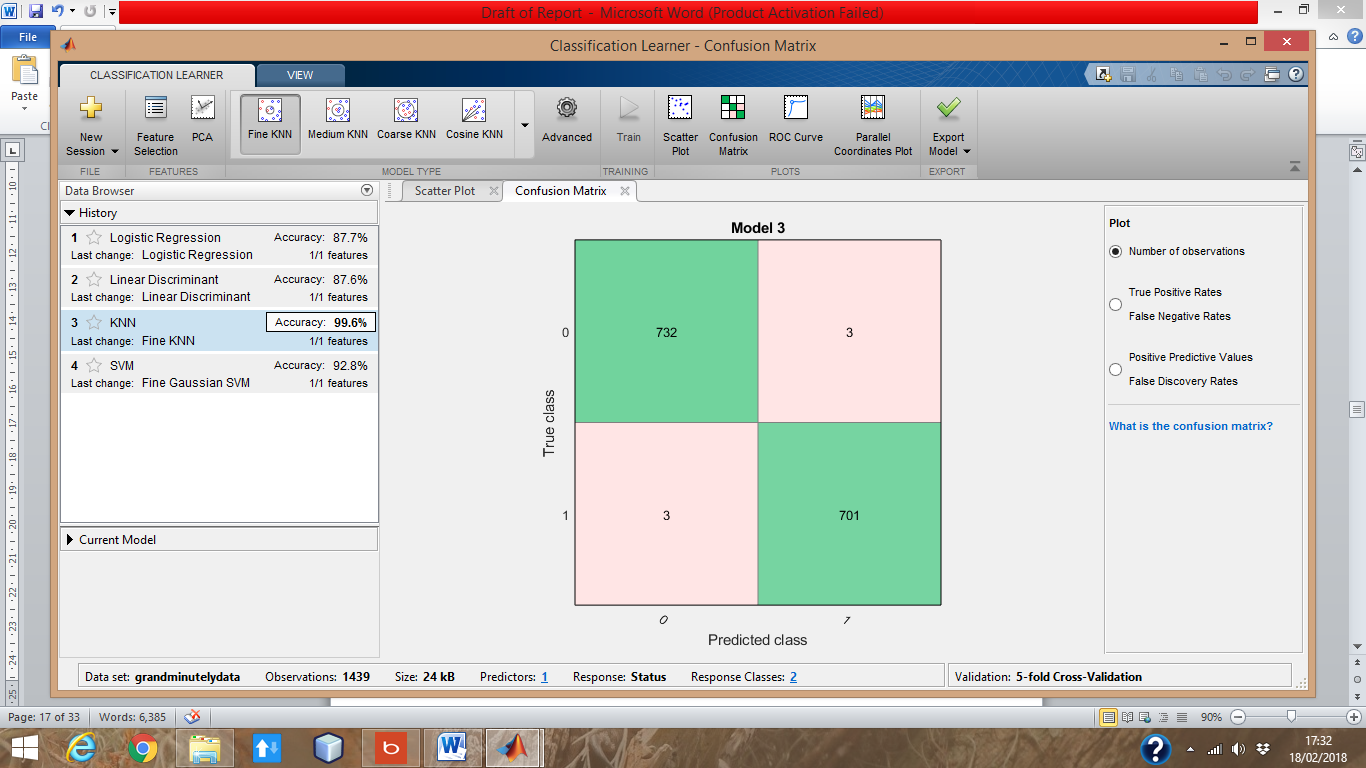


Blue indicates vacant and orange indicates occupied.

Confusion matrix based on logistic regression. Based on minute data (Below).



Confusion matrix based on KNN. Based on minute data (below).



//conclusion of findings

As the above graphs suggests, some algorithms are better than other depending on the data. Logistic regression performed the same as the linear discriminant algorithm. From the tests, they both averaged at an accuracy of 68.8%. Whilst on the other hand, ‘Fine KNN’ scored an average accuracy of 90.5%. The confusion matrix also shows what the algorithm predicted and what it was supposed to predict. On the hourly shopping confusion matrix, logistic regression could predicted a status of 1 but could not actually attain a true value of the status. Therefore, this suggests logistic regression is not typically suitable for data that has a lot of variety and also that does not typically follow a linear trend. On the other hand, if the data follows a linear trend, logistic regression does a good job as the confusion matrix reported healthy numbers on the above picture above.

Furthermore, KNN outperforms logistic regression as this algorithm can be used on data where there necessarily is not a linear trend. On all occasions, Fine KNN managed to get a higher accuracy than logistic regression hence reporting an average accuracy score of 90.5%. The default settings were used when testing the KNN algorithm which is when K = 1. KNN generally works by calculating the distance of the new element from all already stated elements. Once the distance has been calculated, it picks the Kth closest elements the new point it is close to and examines the elements and based on which type of element is more popular within the group of Kth elements; will be the prediction of the algorithm.

In conclusion, logistic regression will be implemented on the app to test the robustness of it and after the KNN algorithm will be implemented.

Microcontrollers

The sensor will be built using an Arduino Uno. The reason for this is because Arduino is an open source hardware microcontroller with a thriving community. As a result of this, it will be easier to learn as there are many guides on how to work with the Arduino Uno. Furthermore, there are guides on how to connect and use other components such as a circuit breadboard, LDR (light dependent resistor) and an ultrasonic sensor.

## Aim and Objectives:

The aim of this project is to help solve an on-going problem most car drivers face daily; finding an available bay to park in. This project will aid and help car drivers find a car parking bay that they’ll be able to park in without driving further than necessary.

The objectives of the project are:

1. Creating the sensor :

This will be achieved by creating a sensor (using an Arduino Uno) that will record the data from an ultrasonic sensor, GPS module and a thermistor module as well as an integrated Wi-Fi module.

1. Code the sensor to detect a change in the environment :

Once the modules acknowledge a drastic change in the environment, the results will be shown to the user via an app hence indicating whether or not the bay is vacant or occupied based on the change in environment.

1. Incorporate machine learning :

As well as that, the app will learn how the data correlates to the bay being vacant or occupied, essentially machine learning. This will be achieved by implementing the concept of neural network as well as using the logistic regression algorithm. The reason for using the logistic regression algorithm is because it is used to answer questions that has binary answers (i.e. two possible answers) and this fits my situation as there are only 2 answers in my context; the parking bay is either occupied or vacant. By implementing this feature, the app will be able to predict whether or not the bay is being used and will come in handy if for some reason the sensor cannot fulfil its purpose (e.g. if there are networking issues, wiring issues, external factors such as rain etc.).

1. Create the server and the database :

The server will be used to send the data to and from; the app and the sensor, over the HTTP/HTTPS protocol. The database will be used to store the sensor’s information it gathers.

1. Design of the sensor :

Designing how the sensor should look aesthetically in order to make the sensor more efficient in terms of accurately reading the changes in environment.

1. Developing the app :

The app will be developed for the android platform. As well as that, it will be developed using the Agile methodology. The app will fetch data from the server and populate a map which will show parking bays that are vacant / occupied near the user. Furthermore, the app will need to be user friendly and not have too much elements for the user to interact with as this app will typically be used whilst the user is behind the wheel of a car so every effort will be made to ensure the user focuses on the road and not on the app as this could pose a danger to the driver. The app will be laid out in such a way that the user should know enough information by interacting with the app by no more than 5 seconds.

# Requirements

How you got these requirements e.g. prototyping, looking at other devices like it

To gather the requirements for this project, I have used a range of requirement gathering techniques such as creating a prototype to see any improvements or to see any missing requirements. By doing this I’ll be able to see if my prototype is practical. Furthermore, I compared different devices/apps already out there in the public, that tries to combat the problem stated. By comparing and contrasting different devices to each other, I will be able to gain a huge insight on the common requirements they fulfil as well as seeing any potential requirements they may have missed out thus making my prototype unique.

Firstly, I have looked at alternative solutions proposed by different companies to combat the problem stated. One of the IT giants; Google, is already making progress in terms finding a solution to this problem. They have implemented a solution and it is available on Google Maps. Their solution works using historic data with machine learning to predict the availability of car parking bays[[11]](#footnote-11). Whilst this is a step in the right direction to solving the problem described, it is not as accurate as having a physical sensor embedded to the parking bays. As the world progresses further in technology, more and more devices are being connected to the internet, the concept of IoT (internet of things) will be greatly beneficial here and will outperform the use of machine learning.

Another solution proposed by the start-up company, AppyParking, is also aiming to tackle the problem. AppyParking uses a sensor that is embedded onto the road and provides real time updates to the user via their app. Their solution is currently implemented in Westminster and Coventry. Their sensors has immensely helped with the problem outlined as it has helped Coventry City Council recognise an approximate £475000 lost revenue due to parking bays that were not either used or placed efficiently[[12]](#footnote-12) whilst providing users real time updates to the available parking bays which in turn has led to 30% fewer miles driven looking for bays and 22% reduction in parking congestion during peak hours[[13]](#footnote-13). Whilst this is similar to my proposed solution, it lacks the use of machine learning which would be greatly beneficial because in the unfortunate event that the sensor stops working, the app would not be able to notify the user if the bay is vacant or occupied.

Moreover, another company that is closely related to this field is Inrix. They work closely with companies such as BMW and Audi. Inrix also gathers data regarding car parks and congestion on the roads in order to provide a huge collection of data in order to understand the current trend and patterns of road usage as well as making the roads more efficient. One idea that they have proposed to combat the problem statement is to use ultrasonic sensors[[14]](#footnote-14). Their method revolves around fitting cars with ultrasonic sensors as opposed to more evasive methods such as physically implanting sensors on the road. This has its pros and cons. It is more economically viable to implant sensors on cars rather than on roads as that would mean less cost since there would be no need to refurbish the roads to incorporate sensors. But a drawback would be the amount of data that would be gathered about the driver and how Inrix would store, or even share, the data.

Comparing the above innovations proposed by these companies, I can see each of them does have its strength and weaknesses therefore I will aim to build my prototype to include the main logical purposes the above serve and also; to include what some of them have missed, i.e. machine learning. I will be using an Arduino Uno to build the prototype and also be using a wifi module for wireless communication as well as wiring up a thermistor and an ultrasonic sensor.

Furthermore, I will need to cater requirements for the UI of the android app that I will be developing. Below are the images of the AppyParking app for Android. My initial reaction at first glance was that there was too much information on one screen. It should’ve been spaced out more and having all the tabs and icons centralised at the bottom of the screen was not efficient. Upon using it, it was pretty confusing to navigate through the app because the tabs on the icon did not represent what the tab would do/perform. Even more, the app became unresponsive as I navigated through the map. This might be due to the fact that it tries to display all the map data at once hence sending multitudes of http requests and receiving responses in a short amount of time whilst the phone is busy constantly busy updating the UI.

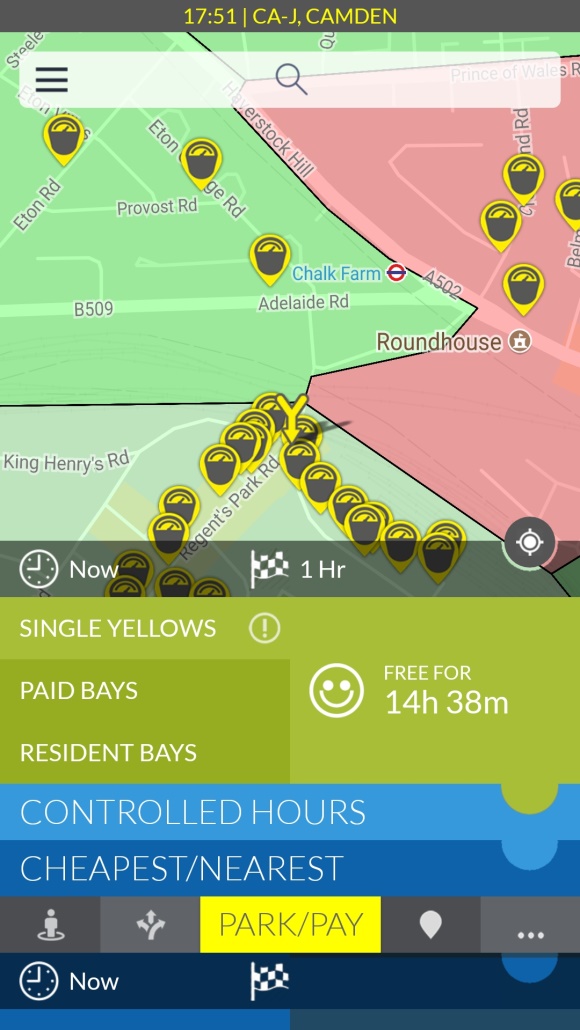
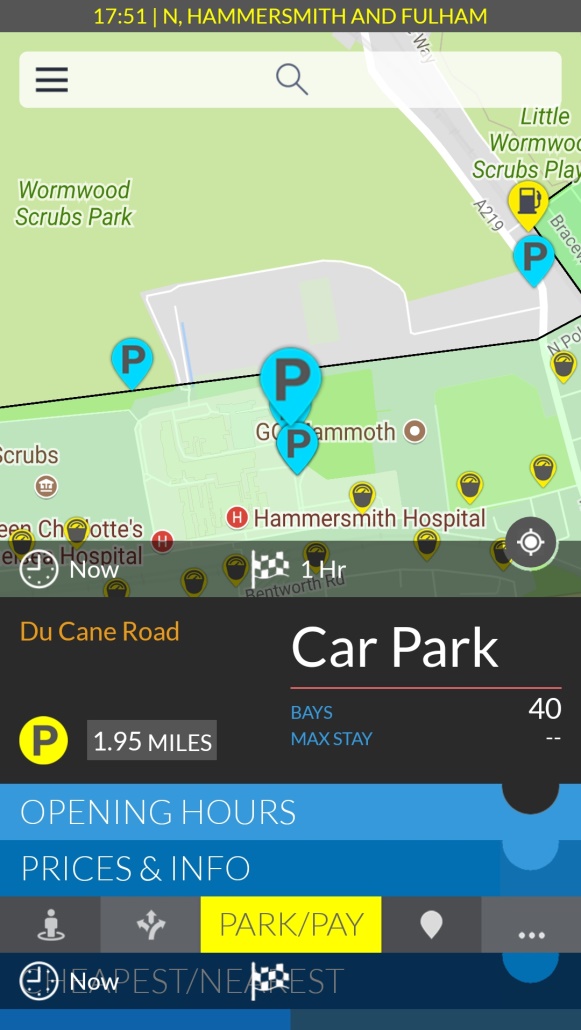
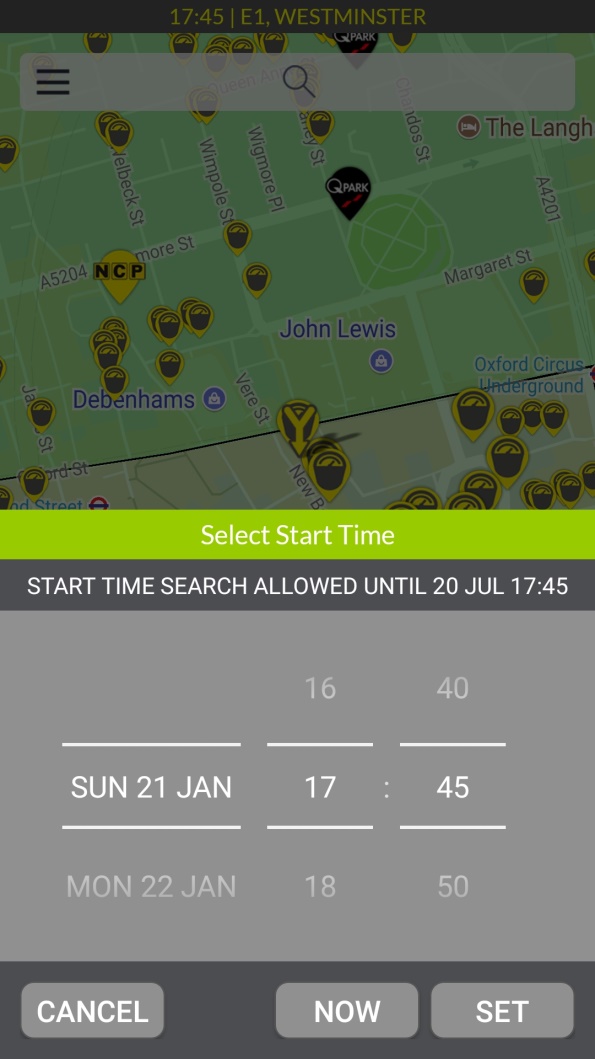
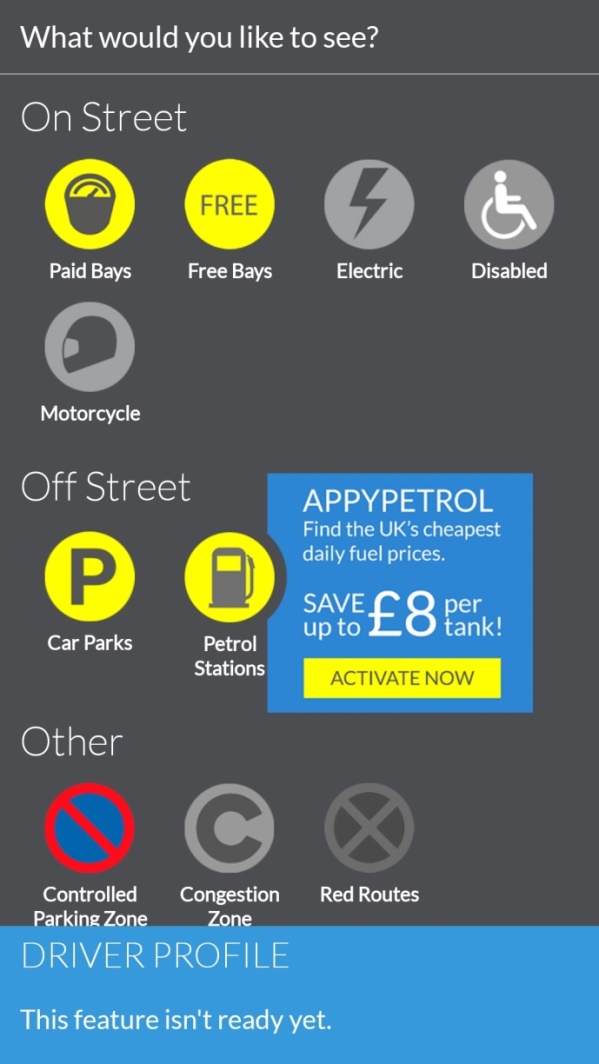
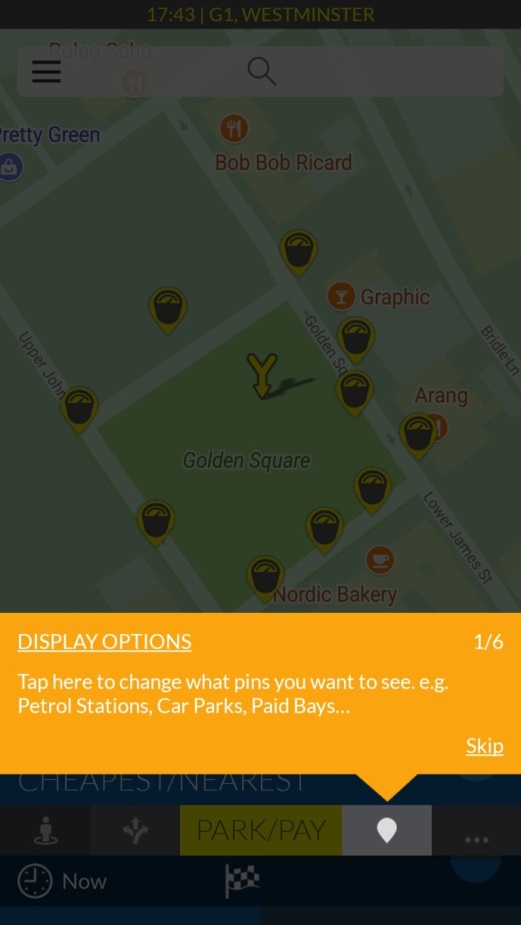
 

Figure 1 (left). Figure 2 (right).

As a result of this, I will be developing the app with a simplistic UI. One has to keep in mind that this app will mostly be used when a driver needs to find a parking bay hence the UI will not have many navigation tabs in the front page and it will need to give a clear and concise outcome within 10 seconds as the driver cannot have their attention diverted from the wheel as it is a potential risk to their lives.



# Design

This section of the report will cover all the design aspect for the project. It will include action diagrams, sequence diagrams, use case diagrams as well as storyboards. Furthermore, it will show the project in an architectural manner in terms of the structure and how the app is built by UML diagrams.

Sequence diagrams –

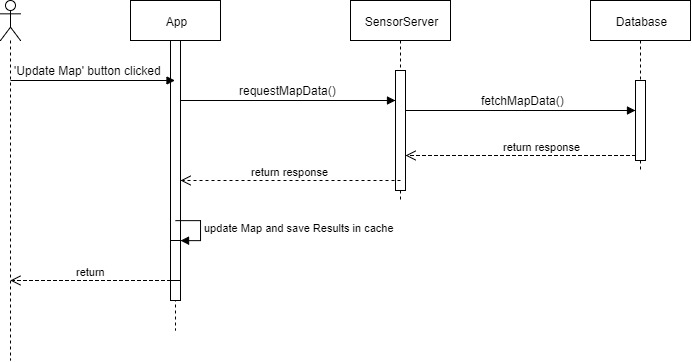


Figure (1) – Above – Shows the interaction with user and app

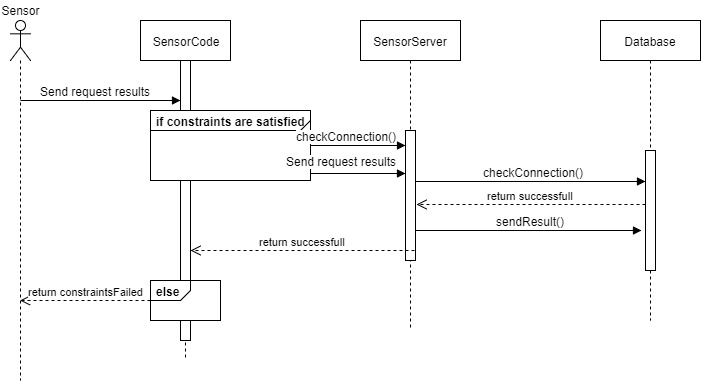


Figure (2) – Above – Shows the sensor interaction with the server and database

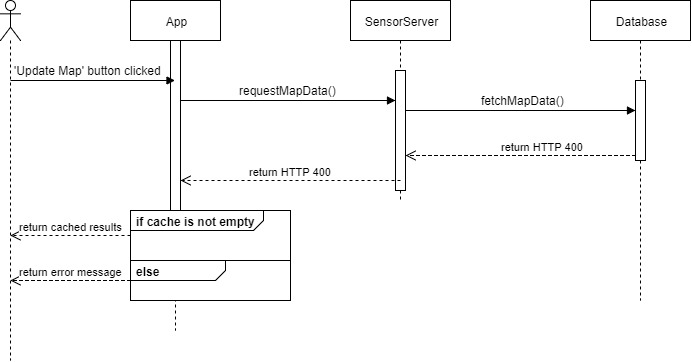


Figure (3) – Above – Shows the interaction of the user if server cannot connect to database

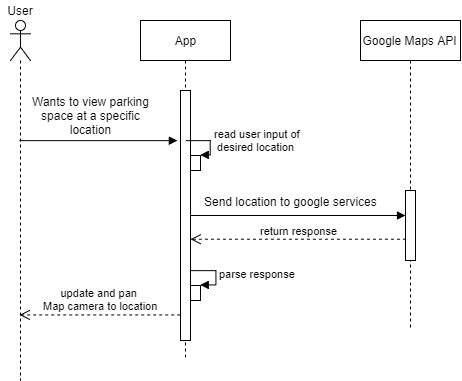
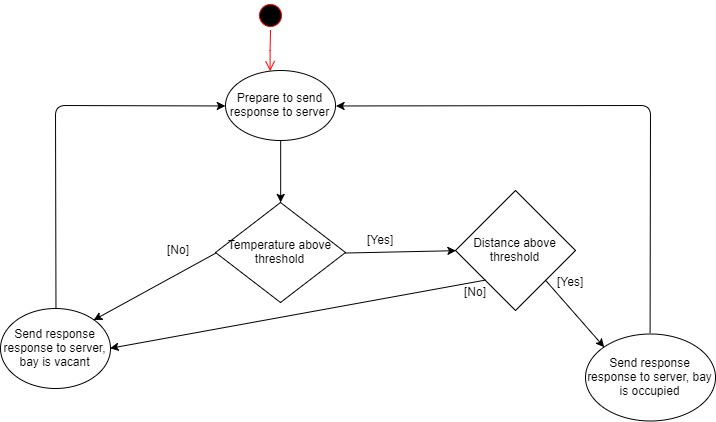


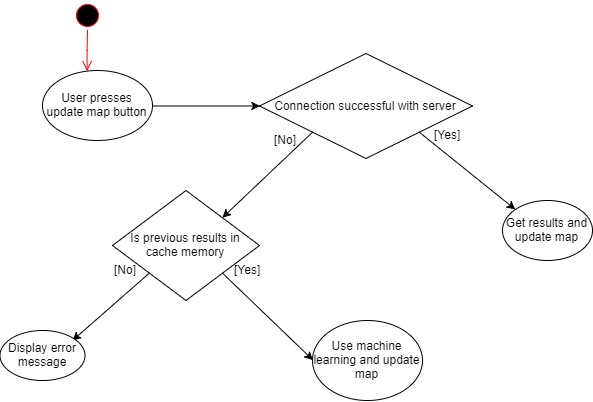
Figure (4) – Above – Shows the interaction of the user when the user wants to check for a parking spot in a certain area.

Action Diagrams –



Figure(1) – Above – Shows the activity diagram of the sensor

Figure (2) – Below – Shows the activity diagram of the user interacting with the aoo.



Use Case Diagrams –

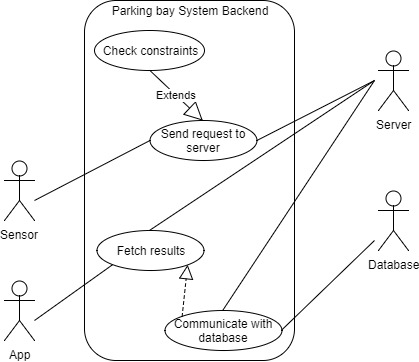
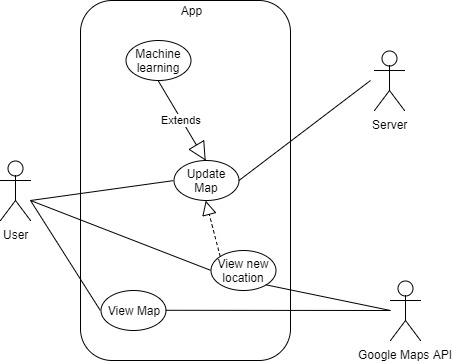


Figure (1) – Above – Shows the uses cases of the backend of the system

Figure (2) – Below – Shows the use cases of the user requesting to a new location to look for a parking bay



(Breadboard circuit diagram?)

# Methodology

In this section I will be talking about the coding methodologies that will be used in the implementation. Even more, I will be talking about general methodologies (such as using Git, working in an Agile manner etc) that will aid me in implementing the code.

Source Control –

Source control is important when it comes to writing code. Not only is it generally a good practise to use but this is also practised in many jobs in the computing industry. There are many varieties of source control out there such as Git, Subversion and Mercurial to name a few. In this project, I have chosen to use git mainly because I am familiar with the concept of it.

The use of git is needed but it needs to be used in a suitable manner to prevent work from being overwritten in an accidental commit, accidentally pushing invalid code to the master branch etc. To prevent this from occurring, the GitFlow methodology will be used. GitFlow is a branching model for Git as it very clear and concise to use. Due to the way the GitFlow model is structured, it’s quite easy for developers to release emergency fixes to patch any serious bugs. Furthermore, it allows developers to work collaboratively due to the nature the branches are laid out. Below is an example of how the GitFlow model will look like:

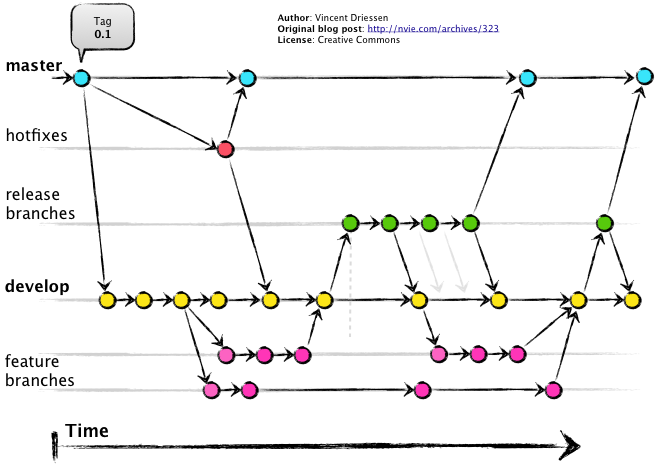


Figure (1) – Above - Author: Vincent Driessen, <http://nvie.com/archives/323>

As you can see from the above diagram, the branches are laid out in a nice and structural manner in which it is easy to revert back to a previous commit if necessary as well as that, it’s easier to track what is going on.

Furthermore, I will be implementing the SCRUM methodology in the way I work. Agile has become popular over the years and a lot of companies is embracing the new style of work and favouring it rather than the waterfall method. SCRUM is a subset of the Agile methodology which is based on iterative development. SCRUMs core concept is the use of development cycles called Sprints which allows one to dynamically adapt to changes whether it be from issues in code or requirements changing. Furthermore, issues/tasks are placed in Sprints and within the duration of the Sprint, one aims to get everything finished. Upon completion of the Sprint, there is a Sprint planning in which issues and tasks are taken from a backlog and placed into the current Sprint. By having this ability, this is what makes SCRUM more dynamic and adaptable to change.

In regards to the software aspect, it is more of logical errors and bugs that will cause a lot of the problems. This will be fixed by following a TDD (test driven development) approach as this is sort of development style is heavily used in the industry as it ensures that your code is in working condition since it shows that the code has been revolved around the unit tests rather than the unit tests being revolved around the code.

# Tech Stack

Maven – Maven is a build automation tool. It is primarily used for Java projects. The advantages of using Maven is that can download the libraries you wish to use in your project, furthermore, it will also download further dependencies the library requires hence it is efficient when it comes to deployment. Furthermore, all the lifecycle is contained in a file called POM.xml in which you can choose what happens in different stages as your project gets built. There are 7 different stages in the lifecycle[[15]](#footnote-15):

validate - validate the project is correct and all necessary information is available

compile - compile the source code of the project

test - test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed

package - take the compiled code and package it in its distributable format, such as a JAR.

verify - run any checks on results of integration tests to ensure quality criteria are met

install - install the package into the local repository, for use as a dependency in other projects locally

deploy - done in the build environment, copies the final package to the remote repository for sharing with other developers and projects.

I have chosen to incorporate Maven in my implementation because I have experience with it, furthermore, I will be using libraries such as Jackson, JSON2POJO so having maven and downloading them from the maven repository will be beneficial.

Spring Boot – Spring Boot is part of the Spring framework, a highly used framework which incorporates a lot of nice features such as dependency injection and web applications (i.e. Spring MVC). Maven is incorporated with Spring Boot which is ideal for me as well as that, it contains an embedded servlet container so it will have the choice of my server on it which is ideal in terms of deployment as everything will be packaged in one jar.

MongoDB – MongoDB is a noSQL database. It uses JSON-like documents with schemas which will be handy for me as I will be handling JSON structures from the server to the database and to the app so it’ll be better to keep one uniform structure throughout the process. Secondly, the reason behind choosing mongoDB is due to the fact that noSQL is known for its speed as it can be horizontally scaled. This is means that the more database servers you have in the server pool, the faster it is to perform operations rather than adding more power (i.e. powerful hardware) to the servers; which is known as vertical scaling. I do plan on taking this project and developing it to a commercial grade hence it is important to think about the overall big picture as there will theoretically be thousands of sensors writing to the database cluster.

Android Studio – Android Studio is the preferred IDE to use when developing apps for Android. This is made by Google themselves.

Java – Java is an object orientated programming language. It is well known throughout the industry and has been used in the early days of programming. Through its many updates, Java has become one of the most popular languages. As of January 2018, the current version of Java is Java 9. I will be developing the majority of my project in Java; specifically the server and the app. The reason for choosing Java is because android apps are primarily developed using Java and is not worth the hassle of trying to develop an android app using another language through other means such as using their NDK (native development kit) which allows developers to code android apps in C, C++. Also, I am proficient in Java.

Arduino – Arduino is a microcontroller. More specifically, it is an open source hardware and software company. I will be using the Arduino microcontroller to build my sensor. The reason for using an Arduino is because of my lack of experience with electronics and microcontrollers and the Arduino is a great place to start for people who have no experience in electronics

C –

Implementation

This section will describe how the major components of the project were implemented as well as challenges faced and strategies used to make unit testing easier. Talk about strucute/class etc.

Model:

The structure to model the parking bays was coded to keep everything simple and elegant therefore the class of the parking bay was created based on the idea of pojo (plain old java objects). Using the definition of a pojo from the Spring community, ‘POJO means Plain Old Java Object. It refers to a Java object (instance of definition) that isn't bogged down by framework extensions.’ [[16]](#footnote-16) In essence, this means that the class shouldn’t extend or inherit anything from any framework class.

The model class, which represents a parking bay, consists of getters and setters. As well as that, the getters and setters conform to the JSON schema so that the json2pojo plugin will be able to create a java object from the JSON response. Furthermore, there is an arraylist which takes ‘Bays’ object. The bays class is used for the KNN algorithm which will be explained below.

(Appendix Code 1)

# Implementation

Machine learning implementation:

One of the fundamental requirements for this app was to implement machine learning. The reason behind this choice was because in the unfortunate event that the server was down, then the app would be rendered useless as there would be no source of result being displayed to the user. Therefore as a precaution, machine learning had to be implemented. Another situation that arose from implementing this was where the machine learning should be implemented. If implementing on the app, then the algorithm used would need to be lightweight in terms of memory size as well as not being CPU intensive by performing complex calculations as this would drain the battery of the phone which is not user friendly. If implementing on the server then there would need to be another server involved just solely for machine learning which would compute the algorithm and send the data to the app but the same scenario would still arise, what if the server went down? Therefore the logical way forward would be to implement it on the app whilst keeping complex calculations to a minimal.

KNN algorithm:

The KNN algorithm is essentially a straight forward algorithm; find the closest variables to your chosen point and take into account the nearest K variables depending on their properties. Whichever types of elements is more prominent, that will be the output. Because this algorithm will be implemented on the app, keeping the calculations as straight forward as possible is critical without introducing any complexities. Therefore, the K = 10 if the size of the data will be bigger than 10 otherwise K will be 1.

Furthermore, the algorithm is implemented in another thread other than the main thread. This is because to prevent the app from freezing as the algorithm computes the data which would degrade the usability of the app. Even more, it’s good programming practise to put extensive computations on a different thread other than the main thread so that the main thread is bottled up computing extensive computations.

(Appendix Code 2)

GMaps route finder:

This feature was implemented to give more functionality to the user. When the use clicks on any of the markers on the map, provided that GPS is enabled, it’ll display the path from the user to the marker. This required using the Google Distance API. Parsing the Distance API was challenging as the response contained complex structures that had to be parsed and converted to different objects for it to be compatible with the GoogleMap object. Fortunately, the template of parsing the Google Distance response is provided online courtesy of the open source community and is included in this project. Furthermore, implementing the parsed data so that it can be used on the GoogleMap object was tricky and hence had to search online and take a few snippets of code from a tutorial which showed how to use the parsed data. [[17]](#footnote-17)

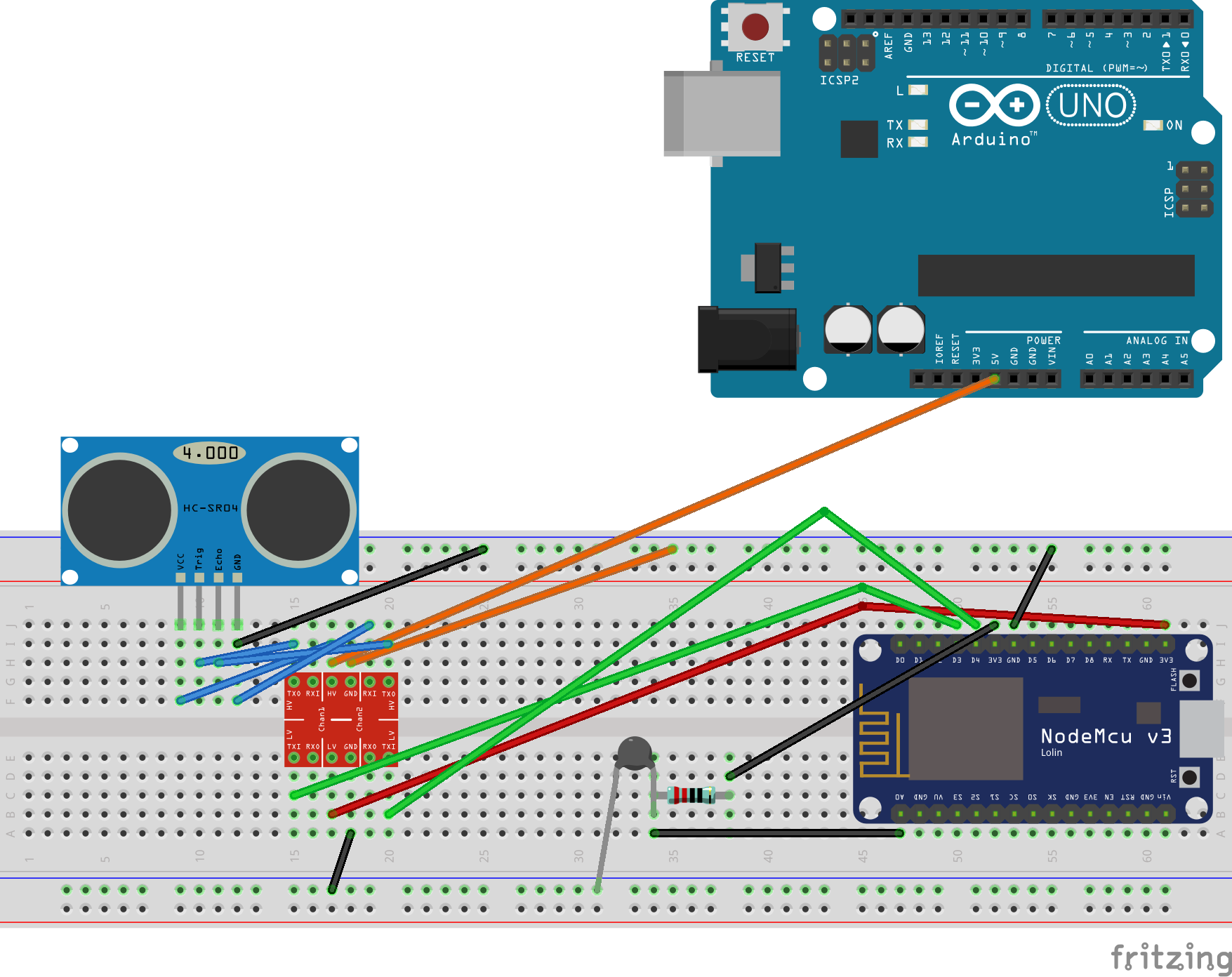
Saving and loading data:

As pointed out in the Android developers guide, there are multiple methods of reading and writing data in order to save and load. Such methods include using SharedPreferences, SQLite and Room; a new DAO framework developed by Google in 2017. Although the ideal choice would using a DOA framework when it comes to storing data, SharedPreferences will be used because it’s simply a key value pair structure as well as me having prior experience in using it. Furthermore, SharedPreferences is more lightweight as there no tables or additional structures are needed to be created unlike in SQLite or Room.

The way this project uses SharedPreferences is by essentially by acquiring a list containing parking bay objects and converts the whole list into a JSON string. This JSON string is then saved as the value of the key-value pair in SharedPreferences. As per the android life cycle, the onCreate() method gets called first and it is in this method that it checks if there is a SharedPrefences available with the key ‘listOfSavedBays’, if yes then it fetches the value and converts the JSON string, containing an array of parking bay objects, to an ArrayList containing the parking bay objects. This is done through the use of the library ‘jackson’ which is one of the widely used libraries in the IT industry catering for JSON to Java conversions.

Arudino – Implementing the sensor; talk about structure of output based on thermistor, ultrasonic etc

The following is the circuit diagram of the sensor:



The sensor circuit comprises of a NodeMCU microcontroller and is powered by an Arduino Uno microcontroller, an ultrasonic sensor, a thermistor and a logic level converter.

The reason for choosing the NodeMCU as the main microcontroller is because it is similar to the Arduino Uno in terms of connections and features but the main difference is that it has built-in WiFi capabilities. The sensor is coded in C using the Arduino IDE. The Arduino IDE is different to many IDEs mainly because it is specifically designed by the Arudino team as well as the feature to change the baud rate and view the serial monitor. In this case, the Arduino Uno is used to power the ultrasonic sensor as that requires 5V which the Arudino Uno can output.

Furthermore, a logic level converter is used to safely step up and step down voltage. This is necessary as the NodeMCU can only output 3.3V but the ultrasonic requires 5V to be used. Therefore, the Arduino Uno is being used to power the module and all the connections from the ultrasonic sensor steps down from 5V to 3.3V using the logic level converter. Once stepped down, all the data from the ultrasonic is fed into the NodeMCU.

Moreover, the thermistor is connected and its data is being fed to the NodeMCU.

The anatomy of an Arduino code has two core methods; setup() and loop(). The setup method is called when the microcontroller is turned on and this is typically the method where you setup up and initialise your variables. After the setup() method has finished executing, the loop() method will constantly run. When the microcontroller finishes executing the last line of code in the loop() method, it’ll execute the loop() method again. This type of structure suits my sensor requirements as there should a sort of loop where it can continuously send data to the server.

The implementation of the code is split into 3 logical components; thermistor, ultrasonic and sending the data over to the server.

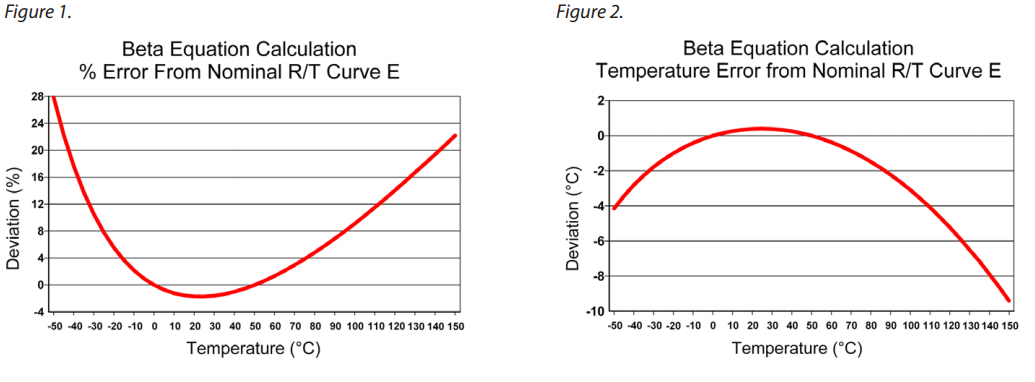
Thermistor:

The thermistor is a widely used component in electronics. Thermistors typically work by lowering its resistance as the temperature rises, and it can also do the opposite. This is all dependent on the material the thermistor is constructed from. When the temperature rises, the lower its resistance becomes hence more current will flow through the thermistor. This is known as Ohms law which can be rearranged to get the following equation, where I is the current, V is voltage and R is the resistance.

There are two main equations that revolve around thermistors; the Stein-Hart equation and the Beta equation. These 2 equations are used to interpolate the resistance vs temperature characteristic of thermistors. The following is the Steinhart equation created by John S Steinhart and Stanley R Hart in 1968 and is released in their paper, ‘Calibration curves for thermistors’:

where T is the temperature in Kelvin, R is the resistance and A, B and C are constants. This equation is typically used to calculate the temperature of the thermistor to a very accurate point. Whilst on the other hand, the Beta equation:

does give near accurate readings. A thorough experiment took place in order to investigate the accuracy of both equations and it was found out that the B equation was the less accurate of the two and the results of the B equation can be found here:

As the graph depicts, using the B equation does give inaccurate results depending on the temperature. From 0 – 60 degrees, the error is minimal as the deviation is less than 1 degree but as the temperature increases, the error becomes apparent.

The implementation of the thermistor in the sensor is using the B equation because firstly, the equation does not have as much variables/constants to work out as opposed to the SteinHart equation. Secondly, despite the B equation not as accurate as the Steinhart equation, the inaccuracy occurs in temperatures that will not be dealt with in this project therefore in this context, one can ignore the inaccuracy.

Furthermore, the beta equation; has most of

Ultrasonic:

The calculation of the ultrasonic module is trivial. The following equation is used, where v is velocity, in this case the speed of sound, s is displacement and t is time. The general idea is to time how long it took for the sound emitted from the trigger port to reach the echo port. By rearranging the above formula, we can get s = vt but this takes into account the time taken for the sound to bounce back from whatever it reflected from hence we’ll need to divide by 2 to cut the time in half hence the formula becomes s=tv/2 which v is 340 if meters is required or 0.034 is cm is required.

Server:

This project mainly revolves around 2 spectrums of communications; machine-to-machine and client-to-machine. Client-to-machine has been with us for a long time. An example of client-to-machine is browsing the internet; the clients, humans, are communicating with a machine; the server. Whilst this method of communication is widely recognized, the other form communication; M2M (machine-to-machine), is getting adopted and is mainly used in areas revolving around IoT. Quoting Carles and Mischa’s definition of M2M, “M2M generally refers to information and communications technologies (ICT) able to measure, deliver, digest, and react upon information in an autonomous fashion, i.e., with no or really minimal human interaction during deployment, configuration, operation, and maintenance phases.” This essentially means machines following a set of protocols and manipulating data. In their book, ‘Machine-to-Machine (M2M) Communications’, they delve into more detail as to what revolves around the concept of machine-to-machine communication. One of the categories that classifies m2m communication is real-time. “…real time allows making optimal and timely decisions based on a large amount of prior collected historical data. The trend is to move away from decision making based on long-term averages to decisions based on real-time or short-term averages, making a real difference to the large amount of nonergodic industrial processes.” Furthermore, the other category that classifies m2m communication is ‘reliability’ of the data as pointed out in their book too. Back in 2014, Google had to abandon its car parking project since relevant stakeholders was not impressed that the data to be used for this feature would be crowdsourced rather than getting fresh data through physical sensors.

This project essentially is split into two methods of communication; machine-to-machine for the communication between the sensor and the server and client-to-machine for the communication between the user/app and the server. The fact that m2m is applied to the sensor-server side further cements that this project revolves around IoT as the data is reliable as there are multiple sensors each contributing to an output of whether or not the parking bay is free or not. And furthermore, the data is fresh as the sensor sends data every few seconds which is near ‘real-time’. And throughout this process, no human contact is needed as it is all automated. There are no set standards, formalities or ‘design pattern’ equivalent in the machine-to-machine realm[[18]](#footnote-18) due to m2m’s infancy but as time progresses there might be a framework that may advance the m2m sphere.

The communication between the server and the sensor is automated therefore nothing much can go wrong in terms of code implementation during runtime/code execution as the sensor’s output will be following a convention/schema with no human interaction and also the server will expect an output from the sensor that will conform to a structure therefore as long as the sensors’ output conforms to the schema, nothing theoretically can go wrong in that concept. The following method is a request method to parse the output from the sensor:

As you can see, the http request looks for 4 request parameters when a request is sent to the ‘/posttodb’ URI. The 4 request parameters relates to the ‘id’, ‘longitude’, ‘latitude’ and ‘status’ of the of the parking bay. After a request is made to this URI, it calls the addGPSEntry and which connects to the mongo database and stores the record there.

Furthermore, from the above code it is evident to see that dependency injection is in use through the use of the ‘@Autowired’ annotation. Dependency injection is mainly used to loosely couple objects from each other so if a major component was to be swapped, it shouldn’t break the code.

Using Spring Boot conventions, a class can be turned into a ‘bean’ by adding the ‘@Component’ annotation above the class file. This can be seen in the DAOImplementation class which implements the ‘DAOInterface’ interface file. Once a class is annotated with ‘Compontent’, Spring knows upon compile time to instantiate this class and places it in the Spring container. Thus by using the @Autowired annotation, Spring knows to look for a bean in the container of the type requested, in this context, ‘DAOInterface’ and because a ‘DAOImplementation’ bean was created of type ‘DAOInterface’, Spring is clever enough to use this bean. By using this approach, it negates the use of using the ‘new’ keyword and thus making code as independent as possible.

Another example of dependency injection is in the DAOImplementation class, the MongoClient object is being injected into the class through the use of autowiring a MongoClient bean.

M2M is about “…big data, notably about (i) real-time, (ii) scalable, (iii) ubiquitous, (iv)reliable, and (v) heterogeneous big data, and thus associated opportunities.”[[19]](#footnote-19) Carles and Mischa breaks down these points and explains them at a finer detail. Essentially, real-time

Testing:

Loosely Coupling Code:

Throughout the project implementation, loosely coupling the code has been stressed as much as possible. By loosely coupling the code, it is easier to unit test as well as making the code cleaner and won’t be prone to breaking if a component would be swapped with another component in the future (e.g. if a new database were to be used, the code should easily be able to integrate with the new database).

In the android implementation, there is a ‘Helper’ class which contains methods that are used throughout the class in the modules. One of the method that is loosely coupled is the ‘getCurrentTime()’ method. This method essentially gets the current time and returns an float representing how many minutes have passed from 12:00am to the current time. Upon initial inspection, one might instantiate a calendar object in the method and then conduct the relevant business logic like thus:



There is nothing wrong with this approach however it would be impossible to correctly test this method because you would not be able to test your logic on a random time therefore not asserting that the business logic works. The way to get around this problem is to make the object independent of the method; essentially loosely coupling the method and the object. As you can see from the following code:



The calendar object is not strictly tied to the method. This is because the reference of the calendar object is created through a ‘TimeHelperImpl’ class which provides a method that returns a Calendar object. The beauty of this is that you can essentially create your own Calendar object and then load it into the Helper class for the getCurrentTime() method to use which is needed to effectively unit test the business logic in the method. As you can see from the following unit test:



A calendar object is created and is assigned a time of 03:00am. Using mockito, a powerful testing framework, a mock object of the TimeHelperImpl class is created and whenever the getTiming() method is called, we return our custom Calendar object which contains a time of 3am. Thus our business logic will be tested on the custom time provided.

# Evaluation

This section of the report will conclude my findings and explain any difficulties I faced whilst implementing the code as well as talking about the sources and references used.

## Logistic Regression:

Initially, logistic regression was implemented on the app. This was because it was the first algorithm that was researched about for this project. The following depicts how the logistic regression was implemented on the app:

As it can be seen from the code, the code is intensive to compute on a phone. To save battery life on the phone and to make sure the phone does not heat up due to performing too many calculations, the epochs had to be lowered because as the data would grow, the time to compute the algorithm would also grow. Furthermore, the training of the weights would add more time and complexity to it. Furthermore, the data structure used to incorporate this algorithm is requires more space/memory than to the KNN algorithm. As it can be seen, the data structure used for the logistic regression by default reserves two Integer arrays of size 1440. This is hefty compared to the dynamic data structure of the KNN algorithm in which it uses an ArrayList to store the relevant objects.

## Timing of algorithms

In order to find out which implementation of the machine learning algorithm was more efficient, it was timed using Java’s static method ‘currentTimeMillis()’ found in the System class. The results are shown below for the KNN algorithm

|  |  |  |
| --- | --- | --- |
| Results for execution time for KNN algorithm | | |
| # of data | K | Time (ms) (avg of 5 exectutions) |
| 1000 | 1 | 0 |
| 2000 | 1 | 6.4 |
| 4000 | 1 | 3.2 |
| 8000 | 1 | 9.4 |
| 16000 | 1 | 9.4 |
| 32000 | 1 | 9.2 |
| 64000 | 1 | 15.8 |
| 1000 | 10 | 3.2 |
| 2000 | 10 | 0 |
| 4000 | 10 | 3.2 |
| 8000 | 10 | 9.6 |
| 16000 | 10 | 6.4 |
| 32000 | 10 | 12.2 |
| 64000 | 10 | 15.6 |
| 1000 | 100 | 6.2 |
| 2000 | 100 | 6.2 |
| 4000 | 100 | 6.4 |
| 8000 | 100 | 6.2 |
| 16000 | 100 | 9.6 |
| 32000 | 100 | 12.2 |
| 64000 | 100 | 12.6 |

|  |  |  |
| --- | --- | --- |
| # of data | Epoch | Time (ms) (avg of 5 exectutions) |
| 1000 | 3 | 6.4 |
| 2000 | 3 | 9.6 |
| 4000 | 3 | 28.0 |
| 8000 | 3 | 22.4 |
| 16000 | 3 | 31.4 |
| 32000 | 3 | 37.8 |
| 64000 | 3 | 78.0 |
| 1000 | 10 | 9.6 |
| 2000 | 10 | 12.8 |
| 4000 | 10 | 21.0 |
| 8000 | 10 | 33 |
| 16000 | 10 | 50.2 |
| 32000 | 10 | 93.8 |
| 64000 | 10 | 193.8 |
| 1000 | 100 | 34 |
| 2000 | 100 | 53 |
| 4000 | 100 | 96.4 |
| 8000 | 100 | 188 |
| 16000 | 100 | 375 |
| 32000 | 100 | 800 |
| 64000 | 100 | 1554 |

As it can be seen for the KNN timings, the execution time does increase relative to the size of the data and the timing do not seem to be dependent on the value of K. KNN is far more efficient compared to the logistic algorithm because the timing for the logistic algorithm depends on how many epochs are made. The reason behind this is that the logistic regression has to train the weights hence the looping and intensive calculations.

Despite logistic regression not being as efficient as KNN, there is potential of using it in the app as one may train the weights once a day for each parking sensor rather than continuously calculate the weights. In other words; caching the weights for a set interval.

However, as seen from the evaluations from Matlab, KNN is far more accurate and faster hence the reason to choose it over logistic regression.

INSERT CODE FOR KNN

INSERT CODE FOR LOGISTIC

## Sensor

Originally, this project was made solely using the Arduino Uno and not the NodeMCU microcontroller but that decision was changed as difficulty arose when trying to program/flash the wifi module that was supposed to be connected to the Arduino. Originally, an ESP8266 chip was used but because this component was the first generation of Wi-Fi modules available for the Arduino Uno, it was difficult to program as it had limited capabilities as well as few GPIO pins. Furthermore, it could not communicate with any Wi-Fi access points that had security enabled which is a major flaw. This was the reason why a NodeMCU microcontroller was used as it had almost the same capabilities as the Arduino Uno but with integrated Wifi. Although the NodeMCU having Wi-Fi integrated inside it, it could not access the university’s Wi-Fi access point. The reason being is that the university wifi access point is an enterprise access point which the NodeMCU is not capable of connecting due to security features.

Even more, the sensor originally had a GPS module connected. This was going to be used to dynamically get the longitude and latitude of the sensor rather than hardcoding the coordinates but this was later dropped as the GPS module was not very efficient in terms of functionality. The GPS module needed to be placed outside and the sky must be clear.

## Future prospects

The future pro

Appendix Code







1. http://www.bbc.co.uk/news/uk-england-35312562 [↑](#footnote-ref-1)
2. https://www.forbes.com/sites/oliviergarret/2017/03/03/10-million-self-driving-cars-will-hit-the-road-by-2020-heres-how-to-profit/#3c617ee37e50 [↑](#footnote-ref-2)
3. RAC Foundation, December 2015, Council parking 'profits' up again in England, Available at: http://www.racfoundation.org/media-centre/council-parking-profits-up-again-in-england-2014-15 [↑](#footnote-ref-3)
4. http://runeberg.org/matetids/1920b/0030.html [↑](#footnote-ref-4)
5. PDF paper [↑](#footnote-ref-5)
6. Again pdf [↑](#footnote-ref-6)
7. Again pdff [↑](#footnote-ref-7)
8. https://www.exastax.com/machine-learning/the-difference-between-ai-machine-learning/ [↑](#footnote-ref-8)
9. http://www.bbc.co.uk/news/technology-42599345 [↑](#footnote-ref-9)
10. http://proquest.safaribooksonline.com.ezproxy.westminster.ac.uk/book/programming/machine-learning/9781784396589/supervised-learning/ch01lvl2sec20\_html#X2ludGVybmFsX0h0bWxWaWV3P3htbGlkPTk3ODE3ODQzOTY1ODklMkZjaDAxczA1X2h0bWwmcXVlcnk9 [↑](#footnote-ref-10)
11. Google Blogs, August 2017, Put it in park with new features in Google Maps, Available at:https://www.blog.google/products/maps/put-it-park-new-features-google-maps/

    [↑](#footnote-ref-11)
12. http://www.appyparking.com/rta.html [↑](#footnote-ref-12)
13. http://www.appyparking.com/rta.html [↑](#footnote-ref-13)
14. http://inrix.com/blog/2017/12/ultrasonic-sensor-parking-availability-technology/ [↑](#footnote-ref-14)
15. https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html [↑](#footnote-ref-15)
16. https://spring.io/understanding/POJO [↑](#footnote-ref-16)
17. http://wptrafficanalyzer.in/blog/drawing-driving-route-directions-between-two-locations-using-google-directions-in-google-map-android-api-v2/ [↑](#footnote-ref-17)
18. Book chapter 1.4 [↑](#footnote-ref-18)
19. http://proquest.safaribooksonline.com.ezproxy.westminster.ac.uk/book/electrical-engineering/communications-engineering/9781782421023/1-introduction-to-machine-to-machine-m2m-communications/s0010\_html [↑](#footnote-ref-19)