Machine Learning Project:

Yaseen Haffejee 1827555

June 17, 2021

Abstract

This Project serves to apply multiple supervised learning algorithms to a dataset, such as Logistic Regression, Naive Bayes, Neural Networks, Decision Trees and Support Vector Machines. The dataset is a combination of features of a car that correlate with whether or not the price of a car is justifiable.

Contents

1 Dataset Description **4**

1.1 General Overview 4

1.2 Features/Attributes 4

2 Data Pre-Processing **6**

2.1 Feature Encoding 6

2.2 Data Split 7

3 Preliminary Data Analysis **8**

3.1 Feature vs Category 8

3.2 Histogram 9

4 Classification Algorithms **10**

4.1 Logistic Regression. 10

4.2 Naive Bayes. 17

4.3 Neural Network 21

4.4 Decision Tree 30

4.5 Support Vector Machine 31

5 Test Results **32**

5.1 Logistic Regression. 32

5.2 Naive Bayes. 34

5.3 Neural Network 36

5.4 Decision Tree 38

5.5 Support Vector Machine 40

6 Comparison of models **42**

6.1 Per class 44

6.2 General Overview 45

Citations **49**

Code **50**

1 Description of the dataset

1.1 General Overview:

The dataset we intend on using to perform our machine learning algorithms on is from the UCI Machine Learning Repository. Click here to go to the dataset. The dataset was originally derived from a hierarchical decision model in the 1998. The dataset has 1728 data-points. Each data-point has 6 features/attributes and can belong to one of four target classes. The purpose of the dataset is to take the pivotal features of a car and determine whether or not the purchase price is justified. Based on the goal of the dataset, it appealed to the group since we feel that given the exorbitant prices manufacturers associate with their cars, the features of the car need to appropriately justify it.

1.2 Features/Attributes:

There a 6 features that we look at:

1) The purchasing price.
2) The maintenance cost.
3) The number of doors the car has.
4) The number of people that can be seated in the car.
5) The size of the boot.
6) The safety of the car.

Each feature has it owns scale which is as follows:

1) The purchasing price : Very High, High, Medium, Low.
2) The maintenance cost : Very High, High, Medium, Low.
3) The number of doors the car has : 2, 3, 4, 5 or more.
4) The number of people that can be seated in the car : 2, 4, more than 4.
5) The size of the boot : Small , Medium, Big.
6) The safety of the car : Low, Medium, High.

These features are what your average individual purchasing a car would look at. The price bracket of the car will directly influence whether a consumer can purchase the car or not. The maintenance of the car is a key feature to consider when purchasing a car since it is a persistent expense we have to incur. The number of doors indicate how big the car is and will directly affect the number of people that can be seated in the car which is an important feature for families. The size of the boot is also an important attribute since most cars are used for travelling and running daily errands such as shopping which requires larger boots. The final feature is the safety of the car which is a paramount feature which can ultimately be the difference between you living and dying in the instance of an accident or in terms of general reliability.

Based on these features, we try to classify the purchasing price in one of four categories:

1) Unacceptable
2) Acceptable
3) Good
4) Very Good

From these 4 target classes, if a cars price-tag is not justified by the six pivotal features, it will then be classified as unacceptable. However, if it is justifiable, the car can be classified as acceptable but if the price-tag is not too high and the features are justified, it can be classified as good and where the price is low and the features are justified it can be classified as very good.

Four sample data-points would be:

Purchasing Price	Maintenance cost	Number of doors	Number of people can be seated	Boot Size	Safety	Target
Very High	High	4	More than 4	Big	High	Unacceptable
High	High	4	4	Big	Medium	Acceptable
Low	Medium	2	4	Small	High	Good
Low	High	5 or more	More than 4	Medium	High	Very Good

Looking at all the data-points, the distribution of classes is as follows:

CLASS	NUMBER OF POINTS BELONGING TO THE CLASS	PERCENTAGE
Unacceptable	1210	70%
Acceptable	384	22.22%
Good	69	3.99%
Very Good	65	3.76%

2 Data Preprocessing

2.1 Encoding:

Considering our data is categorical, we decided to encode the categories using integers. By encoding the data, it simplifies the learning process as well as simplifies the data. Also the numeric representation is pivotal for many of the algorithms we use.

The encoding is represented below.

Target Encoding:

For the targets the encoding will be as follows:

- Unacceptable would be denoted by 0
- Acceptable would be denoted by 1
- Good would be denoted by 2
- Very Good would be denoted by 3

Feature Encoding:

For the Purchasing price and Maintenance cost the encoding will be as follows:

- Low = 0
- Medium = 1
- High = 2
- Very High = 3

For the size of the boot and safety of the car the encoding is as follows:

- Small/Low = 0
- Medium = 1
- Big/High = 2

For the number of doors the encoding is as follows:

- 2 doors = 0
- 3 doors = 1

- 4 doors = 2
- 5 or more doors = 3

For the number of people that can be seated the encoding is as follows:

- 2 people = 0
- 4 people = 1
- More than 4 = 2

2.2 Data Split:

The dataset had 1728 data-points. We decided to split the data into three sets, namely the training data, validation data and testing data.
The training data would have 60% of the data-points and the validation and testing data would both have 20% of the data-points.
Our reasoning behind doing so is to have as large as possible dataset for each in order to optimise our models. By allowing the training dataset to be the largest, we ensure that the model is trained on a greater variety of data-points. By having a relatively large validation dataset , we ensure that we are able to use the dataset to optimise the hyperparameters in our models in order to enhance it's performance. The testing dataset serves as barometer for us to ascertain how accurate our model is in classifying the data. Hence all of these datasets are important in the training and measuring of success of our model.

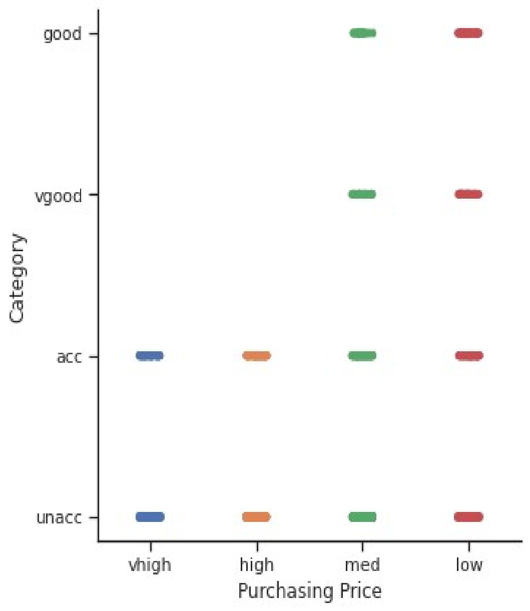
Thus the structure is as follows:

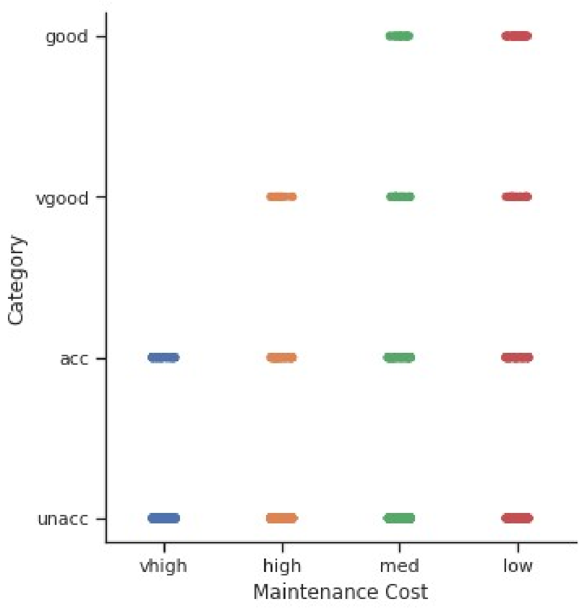
Dataset	Number of data-points
Training Data	1728 x 60% = 1037
Validation Data	1728 x 20% = 345.6 = 346
Test Data	1728 x 20% = 345.6 = 345

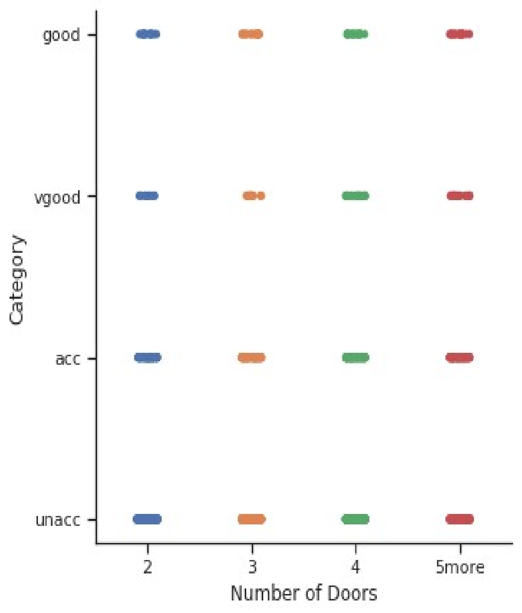
Note that we have given the validation data an extra data-point in comparison to the test data.

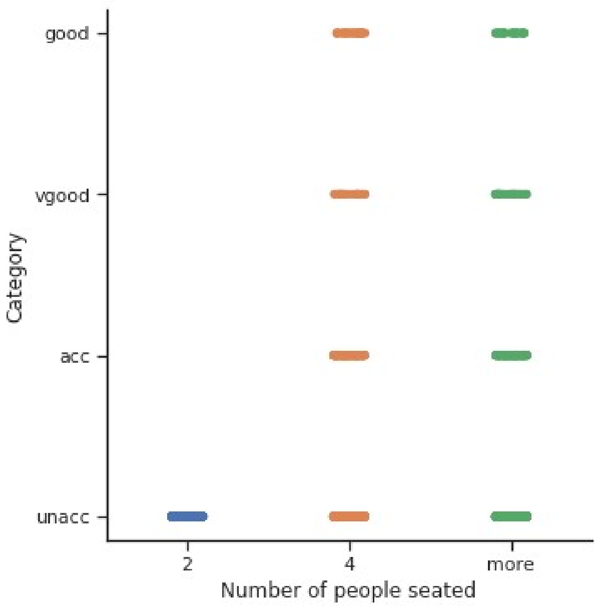
3 Preliminary Data Analysis

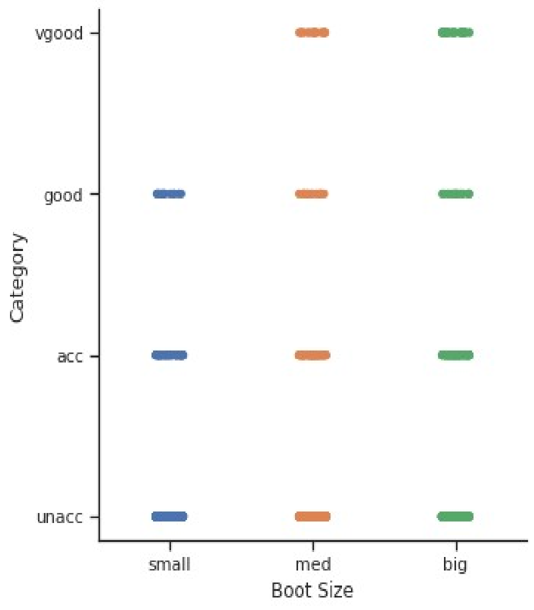
3.1 Feature vs Category

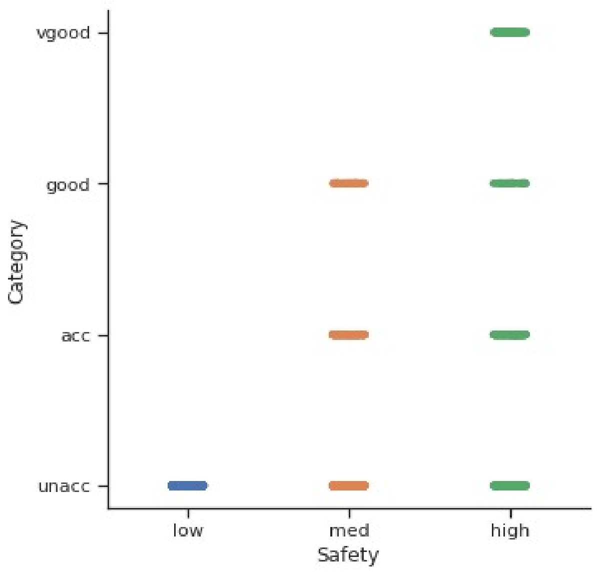












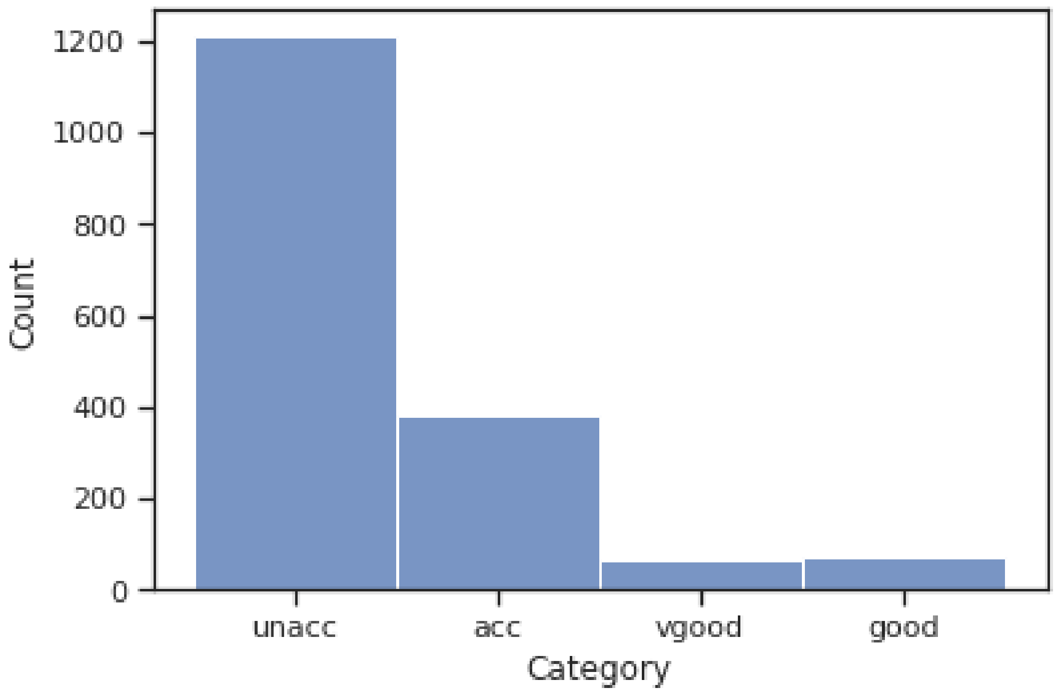
The above graphs give us insight into how each feature and it's corresponding values affect the category in which the car will be classified within.

For example from the above we can decipher the following:

If a car can only seat 2 people, it will most likely be classified as unacceptable. Similarly, if a car has low safety, it will most likely be classified as unacceptable.

3.2 Histograms

The diagram below shows us the distribution of the data-points according to the category they belong to.



4 Classification Algorithms

4.1 Logistic Regression

Logistic Regression is a discriminative model which can be used to predict the probability of a data-point belonging to a certain class. The class with which the highest probability is associated with , is the class the data-point will be assigned to. For our dataset, multi-class logistic regression is the model we employed to learn the dataset. Multi-class logistic regression works in the same manner as binary logistic regression, in the sense that we adopt a one verse all approach in the algorithm. In essence, it is equivalent to studying multiple binary logistic regression models. Since we had four possible outcomes/targets, we consequently had four logistic regression models being learnt simultaneously.

The process of learning using Logistic Regression is as follows:

1. Firstly create the design matrix for the Training Data. The first column of the design matrix will always be the bias and is a column containing the number one at every entry.
2. Since we are learning four different models simultaneously, we randomly assign a vector of linear weights to each model. The size of each of these vectors corresponds to the number of features, in our case it is six and an additional column which corresponds to the bias, giving us a vector with seven randomly assigned weights. Thus we have four vectors of randomly assigned weights corresponding to each feature in the dataset as well as the bias.
3. We also need a heuristic function which will be used to predict the probability. For this we used the sigmoid function, which is given by the following formula.
   $$h_\Theta(z)=\sigma(z)=\frac{1}{1+e^{-z}}$$
   Formula 1: Sigmoid Formula
4. Every machine learning model has a cost function associated with it. The cost function for Logistic Regression is give by:
   $$E(\Theta)=-\sum_{i=1}^{m} y^{(i)}\log(h_\Theta(x^{(i)})+(1-y^{(i)})\log(1-h_\Theta(x^{(i)})))$$
   Formula 2: Error Function

5. In order to actually learn the ideal weights in order to predict correctly, we employ Gradient Descent. Gradient Descent is a method of finding a minima of a function. In this case the function we are minimising is the cost function given above. We perform Gradient Descent until the norm between consecutive updated weights, are less than a threshold which is denoted by ε. Gradient Descent also has another hyper-parameter called the learning rate, denoted by α. The learning rate represents the size of the step we take in the downhill direction (negative gradient). Let Θ be representative of a vector of weights, $x^{(i)}$ represents the ith data-point and $y^{(i)}$ represents the ith target value. The weight update is as follows: $\Theta_j \leftarrow \Theta_j - \alpha(h_\Theta(x^{(i)}) - y^{(i)})$

Formula 3: Weight Update

whereby, $h_\Theta(x^{(i)}) = \frac{1}{1+e^{-\Theta^T x^{(i)}}}$.

6. We perform the above weight update iteratively until convergence. However, we have four weight updates for each weight vector at each iteration for each respective model.

7. We now run the algorithm and capture the accuracy and confusion matrix prior to applying any weight updates. We ran the algorithm four times. We capture the results below.

8. Once the model is trained we can look at the Error within our Training Data. We ran the Algorithm four times and the Training Error is captured below.

9. Since the algorithm involves hyper-parameters, we need to learn the optimal hyper-parameters. For this we use our validation data. We ran the algorithm four times. The corresponding accuracies and confusion matrices with their respective hyper-parameters can be summarised in the table below:

Iteration 1:

Accuracy	Confusion Matrix	Learning Rate(α)	Threshold(ε)
Prior to training: 23.24%	$\begin{pmatrix} 50 & 0 & 0 & 0 \\ 521 & 183 & 35 & 32 \\ 21 & 9 & 0 & 0 \\ 130 & 42 & 6 & 8 \end{pmatrix}$		
After training: 64,32%	$\begin{pmatrix} 648 & 222 & 18 & 25 \\ 44 & 4 & 5 & 2 \\ 14 & 1 & 5 & 3 \\ 16 & 7 & 13 & 10 \end{pmatrix}$		
Validation: 65,03%	$\begin{pmatrix} 221 & 69 & 10 & 9 \\ 15 & 0 & 1 & 0 \\ 4 & 0 & 1 & 1 \\ 8 & 3 & 1 & 3 \end{pmatrix}$	0,01	0.000000005
61,12%	$\begin{pmatrix} 202 & 64 & 6 & 6 \\ 32 & 5 & 4 & 2 \\ 6 & 0 & 1 & 1 \\ 8 & 3 & 2 & 4 \end{pmatrix}$	0.001	0.00000000005
20,23%	$\begin{pmatrix} 15 & 0 & 0 & 0 \\ 184 & 53 & 13 & 11 \\ 2 & 4 & 0 & 0 \\ 47 & 15 & 0 & 2 \end{pmatrix}$	0.0000000000000000001	0.005
20,23%	$\begin{pmatrix} 15 & 0 & 0 & 0 \\ 184 & 53 & 13 & 11 \\ 2 & 4 & 0 & 0 \\ 47 & 15 & 0 & 2 \end{pmatrix}$	0.000000001	0.005

Table 1: Iteration 1 Results

Iteration 2:

Accuracy	Confusion Matrix	Learning Rate(α)	Threshold(ε)
Prior to Training: 3,09%	$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 256 & 40 & 0 & 0 \\ 474 & 190 & 45 & 32 \end{pmatrix}$		
After Training: 62,30%	$\begin{pmatrix} 646 & 227 & 45 & 32 \\ 1 & 0 & 0 & 0 \\ 83 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$		
Validation: 60,12%	$\begin{pmatrix} 208 & 77 & 9 & 18 \\ 0 & 0 & 0 & 0 \\ 34 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$	0,01	0.000000005
58,09%	$\begin{pmatrix} 201 & 77 & 9 & 18 \\ 5 & 0 & 0 & 0 \\ 36 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$	0.001	0.00000000005
5,20%	$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 103 & 11 & 0 & 0 \\ 139 & 66 & 9 & 18 \end{pmatrix}$	0.0000000000000000001	0.005
5,20%	$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 103 & 11 & 0 & 0 \\ 139 & 66 & 9 & 18 \end{pmatrix}$	0.000000001	0.005

Table 2: Iteration 2 Results

Iteration 3:

Accuracy	Confusion Matrix	Learning Rate(α)	Threshold(ε)
Prior to Training: 3,57%	$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 \\ 730 & 221 & 37 & 44 \\ 0 & 0 & 0 & 0 \end{pmatrix}$		
After Training: 30,18%	$\begin{pmatrix} 250 & 158 & 37 & 44 \\ 478 & 63 & 0 & 0 \\ 5 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 \end{pmatrix}$		
Validation: 27,17%	$\begin{pmatrix} 84 & 66 & 15 & 12 \\ 157 & 10 & 0 & 0 \\ 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$	0,01	0.000000005
27,46%	$\begin{pmatrix} 85 & 66 & 15 & 12 \\ 156 & 10 & 0 & 0 \\ 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$	0.001	0.00000000005
4,34%	$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 \\ 241 & 76 & 15 & 12 \\ 0 & 0 & 0 & 0 \end{pmatrix}$	0.0000000000000000001	0.005
4,34%	$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 \\ 241 & 76 & 15 & 12 \\ 0 & 0 & 0 & 0 \end{pmatrix}$	0.000000001	0.005

Table 3: Iteration 3 Results

Iteration 4:

Accuracy	Confusion Matrix	Learning Rate(α)	Threshold(ε)
Prior to Training: 51,88%	$\begin{pmatrix} 526 & 172 & 24 & 30 \\ 65 & 5 & 1 & 0 \\ 16 & 0 & 0 & 0 \\ 132 & 42 & 17 & 7 \end{pmatrix}$		
After Training: 69,05%	$\begin{pmatrix} 716 & 219 & 42 & 37 \\ 23 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$		
Validation: 65,32%	$\begin{pmatrix} 226 & 89 & 12 & 17 \\ 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$	0,01	0.000000005
65,61%	$\begin{pmatrix} 227 & 89 & 12 & 17 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$	0.001	0.00000000005
50,58%	$\begin{pmatrix} 171 & 72 & 8 & 14 \\ 16 & 1 & 0 & 0 \\ 8 & 0 & 0 & 0 \\ 33 & 16 & 4 & 3 \end{pmatrix}$	0.0000000000000000001	0.005
50,58%	$\begin{pmatrix} 171 & 72 & 8 & 14 \\ 16 & 1 & 0 & 0 \\ 8 & 0 & 0 & 0 \\ 33 & 16 & 4 & 3 \end{pmatrix}$	0.000000001	0.005

Table 4: Iteration 4 Results

From the above tables, we can definitely conclude that the algorithm does work since the accuracy after training is greatly improved in comparison to the accuracy prior to training.
Since this implies the model is learning we can now discuss irregularities in the results.

In iteration 3, the training accuracy after learning is 30,18%. This is relatively low in comparison to the other iterations. One viable explanation for this is the fact that at the beginning of each iteration, the dataset is randomly shuffled and the split into the 3 relative datasets, namely training, validation and testing. This randomisation of the split could lead to the training data being skewed to a certain target, consequently the algorithm becomes bias to that target in the weights adjustment during gradient descent, hence when we received an even distribution in the validation data, the algorithm performed poorly.

Another fact to point out lies within the results of the validation data. The last two rows, with a learning rate of 0.0000000000000000001 and 0.000000001 respectively and the same threshold of 0.005, always have the same accuracy and confusion matrix as well as relatively poor performance. A possible explanation for this is the fact that in both, the learning rate is extremely small. Thus, during gradient descent, we are taking extremely small steps to the minima and as a result the error between consecutive weight updates does not decrease drastically. This results in the while loop terminating once a 1000 iterations have been performed and at that time , the weights have not been optimally adjusted. Hence , we end up with an underwhelming model.

We carry out the validation testing to find the optimised hyper-parameters. Since our validation data varies from our training and test data, it gives us a good indication of how the model is affected by the current set of hyper-parameters. Since we ran the algorithm 4 times, we will choose the optimised parameters as the model with the best average accuracy. The averages are summarised below:

Threshold and Learning Rate	Time taken to train (seconds)	Accuracy	Average Accuracy
Learning Rate = 0,01 Threshold = 0.000000005	0,19	Iteration 1: 65,03 % Iteration 2: 60,12% Iteration 3: 27,17% Iteration 4: 65,32%	54,51%
Learning Rate = 0.001 Threshold = 0.00000000005	0,3	Iteration 1: 61,12 % Iteration 2: 58,09 % Iteration 3: 27,46% Iteration 4: 65,61%	53,07%
Learning Rate = 0.0000000000000000001 Threshold = 0.005	0,25	Iteration 1: 20,23% Iteration 2: 5,20 % Iteration 3: 4,34 % Iteration 4: 50,58 %	20,09%
Learning Rate = 0.000000001 Threshold = 0.005	0,18	Iteration 1: 20,23% Iteration 2: 5,20 % Iteration 3: 4,34 % Iteration 4: 50,58 %	20,09%

Table 5: Summary of Validation Results

From the above table, we can conclude that the optimised hyper-parameters correspond to the values $\alpha = 0{,}01$ and $\varepsilon = 0.000000005$.
We will use the model corresponding to these values on our testing data.

4.2 Naive Bayes

Naive Bayes is a generative machine learning model which uses conditional probabilities and the assumption of class independence to classify data into these desired classes. It formulates a prediction by calculating the probability of a data-point belonging to a class. Since the model is easy to implement, understand and has a high predictive accuracy, we chose to implement it on our dataset.

The process of learning with Naive Bayes is as follows:

1. Since Naive Bayes does not have any hyper-parameters to optimise, the data split was an 80% Training data and 20% Testing data split. The reason we chose to make the training dataset as large as possible, is because Naive Bayes works with conditional probabilities and can only formulate a conditional probability if it has seen that data. Hence, we maximise the size of the training data in order to ensure the model sees as many variations of data-points as possible. As a result, our Training dataset had 1383 data-points and the test dataset had 345 data-points.

2. The next step is to calculate the prior probabilities of each class. This is the probability of each class occurring in our training dataset. It can be described by the following formula:

$$P(c) = \frac{Number\ of\ occurences\ of\ c}{Total\ number\ of\ datapoints}$$

   Formula 1: Prior Probability

   The Prior Probabilities of our dataset are given by:

```
[0.6999276934201012, 0.22559652928416485, 0.03759942154736081,
0.0368763557483731].
```

3. Thereafter we calculate the likelihood table. The likelihood table is the table which contains the conditional probabilities which we learn from the training data. The formula for conditional probabilities is represented as follows where $x^{(i)}$ represents the ith data-point and C represents a specific class.

$$P(x^{(i)}|C)$$

   Formula 2: Class Conditional Probability

The likelihoods for our dataset are given as follows:

For the first feature, Purchasing Price, it can take on four values given by 0,1,2 or 3 and the data-point can belong to either class 0,1,2 or 3. Hence we have a 4x4 matrix for feature 1.

For feature 2, Maintenance cost, it can take on four values given by 0,1,2 or 3 and the data-point can belong to either class 0,1,2 or 3. Hence we have a 4x4 matrix for feature 2.

For feature 3, Number of Doors, it can take on four values given by 0,1,2 or 3 and the data-point can belong to either class 0,1,2 or 3. Hence we have a 4x4 matrix for feature 3.

For feature 4, the number of people that can be seated, it can take on three values given by 0,1 or 2 and the data-point can belong to either class 0,1,2 or 3. Hence we have a 3x4 matrix.

For feature 5, boot size, it can take on four values given by 0,1 or 2 and the data-point can belong to either class 0,1,2 or 3. Hence we have a 3x4 matrix for feature 5.

For feature 6, safety of the car, it can take on four values given by 0,1 or 2 and the data-point can belong to either class 0,1 2 or 3. Hence we have a 3x4 matrix for feature 6 .

Using feature 1 as an example, I will illustrate how to read the matrix.
Index [0][0] refers to the probability that the feature takes on the the value 0 and belongs to class 0.
Index[1][1] refers to the probability that the feature takes on the value 1 and belongs to class 1.
Index[2][3] refers to the probability that the feature takes on the value 2 and belongs to class 3.
The first index(the row) refers to the value the feature takes on and the second index(the column) refers to the class. At their combined index, we get the conditional probability denoted by P(x|c).
An example of a set of these matrices is as follow:

$$\begin{pmatrix} 0.21084954 & 0.2394822 & 0.60344828 & 0.55555556 \\ 0.22313204 & 0.27831715 & 0.32758621 & 0.38095238 \\ 0.26714432 & 0.28478964 & 0.01724138 & 0.01587302 \\ 0.29682702 & 0.19093851 & 0.01724138 & 0.01587302 \end{pmatrix}$$

Figure 2: Feature 1: Likelihood

$$\begin{pmatrix} 0.2272262 & 0.2394822 & 0.62068966 & 0.38095238 \\ 0.21699079 & 0.29449838 & 0.31034483 & 0.38095238 \\ 0.2569089 & 0.27508091 & 0.01724138 & 0.19047619 \\ 0.29682702 & 0.18446602 & 0.01724138 & 0.01587302 \end{pmatrix}$$

Figure 3: Feature 2: Likelihood

$$\begin{pmatrix} 0.27430911 & 0.20064725 & 0.22413793 & 0.15873016 \\ 0.24667349 & 0.2491909 & 0.29310345 & 0.19047619 \\ 0.2395087 & 0.26860841 & 0.20689655 & 0.31746032 \\ 0.23746162 & 0.27508091 & 0.24137931 & 0.3015873 \end{pmatrix}$$

Figure 4: Feature 3: Likelihood

$$\begin{pmatrix} 0.47287615 & 0.00323625 & 0.01724138 & 0.01587302 \\ 0.24360287 & 0.51132686 & 0.44827586 & 0.46031746 \\ 0.28045036 & 0.47572816 & 0.48275862 & 0.47619048 \end{pmatrix}$$

Figure 5: Feature 4: Likelihood

$$\begin{pmatrix} 0.38382805 & 0.25889968 & 0.31034483 & 0.01587302 \\ 0.32753327 & 0.35275081 & 0.2931034 & 0.36507937 \\ 0.28556807 & 0.37864078 & 0.34482759 & 0.57142857 \end{pmatrix}$$

Figure 6: Feature 5: Likelihood

$$\begin{pmatrix} 0.48208802 & 0.00323625 & 0.01724138 & 0.01587302 \\ 0.28761515 & 0.46925566 & 0.51724138 & 0.01587302 \\ 0.2272262 & 0.51779935 & 0.4137931 & 0.92063492 \end{pmatrix}$$

Figure 7: Feature 6: Likelihood

4. The final step is to apply the all the above into the Naive Bayes model which is given by:

$$P(C|x) = \frac{\prod_{i}^{n} P(x_i|c) * P(C)}{P(x)}$$

Formula 3: Naive Bayes Model

Note that P(x) is just a normalization term and is given by:

$$P(x) = \sum P(x|c) * P(c)$$

5. One we have applied the above, the class with the maximum probability is the class we assign the data-point to. This is referred to as the Maximum a posteriori(MAP) solution.

We ran the algorithm 4 times and the performance metrics are captured below:

Accuracy	Confusion Matrix	Time (seconds)	Precision	Recall
84,67%	$\begin{pmatrix} 949 & 32 & 1 & 0 \\ 95 & 197 & 7 & 0 \\ 1 & 41 & 8 & 1 \\ 0 & 34 & 0 & 17 \end{pmatrix}$	0,09	Class0: 0.9081 Class1: 0.6480 Class2: 0.5 Class3: 0.9444	Class0: 0.9664 Class1: 0.6589 Class2: 0.1569 Class3: 0.3333
85,61%	$\begin{pmatrix} 940 & 33 & 0 & 0 \\ 84 & 216 & 5 & 0 \\ 1 & 40 & 10 & 1 \\ 0 & 35 & 0 & 18 \end{pmatrix}$	0,09	Class0: 0.9171 Class1: 0.6667 Class2: 0.6667 Class3: 0.9474	Class0: 0.9661 Class1: 0.7082 Class2: 0.1923 Class3: 0.3396
85,90%	$\begin{pmatrix} 944 & 28 & 0 & 0 \\ 82 & 216 & 5 & 0 \\ 0 & 43 & 15 & 0 \\ 0 & 36 & 1 & 13 \end{pmatrix}$	0,11	Class0: 0.9200 Class1: 0.6687 Class2: 0.7143 Class3: 1	Class0: 0.9712 Class1: 0.7129 Class2: 0.2586 Class3: 0.2600
85,54%	$\begin{pmatrix} 953 & 24 & 2 & 0 \\ 85 & 203 & 9 & 0 \\ 1 & 43 & 14 & 0 \\ 0 & 35 & 1 & 13 \end{pmatrix}$	0,14	Class0: 0.9172 Class1: 0.6656 Class2: 0.5385 Class3: 1	Class0: 0.9734 Class1: 0.6835 Class2: 0.2414 Class3: 0.2653

Table 6: Naive Bayes Training Data Results

4.3 Neural Network

A neural network is a series of algorithms that embarks on the journey of learning underlying trends and relationships in a dataset. This process is one that tries to mimic the way the human brain works.
In a neural network, we have layers of neurons, in our case these are artificial. Artificial neural networks are comprised of a node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network. When we stack these layers together, the result is a neural network. Neural networks rely on training data to learn and improve their accuracy over time. However, Neural networks also have the ability to adapt to changing input and we also have the flexibility of using a variety of activation functions at each layer of the network. Hence neural networks become extremely effective at learning underlying trends within the dataset and have a strong predictive power.

We implemented the neural network as follows:

1. The first step was to structure our output data correctly. In order to do so, we employed one-hot encoding and implemented in on the targets of our dataset.

2. The next step was to decide on which activation functions to use at each layer. We decided to use a sigmoid function as the activation at the first layer and a softmax function at every other layer in the network. The formulas are as follows:

$$\sigma(z)=\frac{1}{\left(1+e^{(-z)}\right)}$$

Formula 4: Sigmoid Activation

$$f(X,\Theta)=\frac{e^{\left(\Theta^{(T)}X\right)}}{\sum e^{\left(\Theta^{(T)}X\right)}}$$

Formula 5: Softmax Activation

3. We then created functions which will initialise a neural network with the respective layers, nodes and weights between these layers. Since we randomly assigned weights, these assignments can heavily impact how rapidly learning occurs depending on how great the adjustments that occur to these weights are.

4. Thereafter we implemented a function which would handled the transfer of input data from the beginning of the network, until the output. This process is

referred to as forward propagation. The output of this function are referred to as the activations, denoted by a.

5. Once we have propagated through the network, we need to see how great our error was and use this information to adjust the weights between nodes so the network can learn.
   The first step in doing this is calculating the error at the final/output layer. This is given by the following formula:

   $$Final\,Error = Prediction - Actual$$

   Formula 6: Error at the final layer

6. Once we have calculated the error at the final layer, we can move backwards through the network and assign blame to certain weights which contributed the most to this error at the final layer. This is referred to as Backpropagation. Tis process involves us calculation the derivative of the activation functions with respect to the weights at a given layer. The formula is given by:

   $$\frac{df}{\Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$$

   Formula 7: Backpropagation Formula

   This represents the update per weight. However we can vectorise the calculation.

7. Thereafter we calculate the gradient of the loss function with respect to the weights. This is given by the following formula:

   $$\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$$

   Formula 8: Gradient Formula for a single weight

   We can also vectorise the step and the formula is as follows:

   $$\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$$

   Formula 9: Vectorised Gradient Formula

8. Once we have all of this information we can actually perform an informative weight update. The proportion that the weight is updated is in accordance to how much the weight contributed/influenced the error. The adjustment enables us to tweak the weights in order to minimise the error. This is accomplished since the adjustment is dependent on the minimisation of our error/loss function. The formula for the weight update without using a regularised average gradient is as follows:

$$\Theta_{ij}^{(l)} = \Theta_{ij}^{(l)} - \alpha \Delta_{ij}$$

Formula 10: Weight Update

and can be vectorised as follows:

$$\Theta^{(l)} = \Theta^{(l)} - \alpha \Delta^{(l)}$$

Formula 11: Vectorised weight update

Note that α represents the learning rate.

9. We perform the above steps , 4-8, iteratively for every single data-point and for a specified amount of epochs. We set the number of epochs as a standard 1000.

10. Since there are hyper-parameters to learn, we perform the learning four times on multiple models. We also stuck with one hidden layer and varied the amount of nodes within that hidden layer. The results are captured in the tables below.

Iteration 1:

Accuracy	Confusion Matrix	Nodes in the hidden Layer	Time Taken to Train	Learning Rate(α)	Threshold(ε)
Prior to Training: 21,79%	$\begin{pmatrix} 0 & 724 & 0 & 0 \\ 0 & 226 & 0 & 0 \\ 0 & 42 & 0 & 0 \\ 0 & 45 & 0 & 0 \end{pmatrix}$				
After Training: 70,20%	$\begin{pmatrix} 711 & 12 & 1 & 0 \\ 209 & 17 & 0 & 0 \\ 20 & 22 & 0 & 0 \\ 8 & 36 & 1 & 0 \end{pmatrix}$				
Validation: 73,70%	$\begin{pmatrix} 241 & 2 & 1 & 0 \\ 66 & 14 & 0 & 0 \\ 8 & 8 & 0 & 0 \\ 2 & 4 & 0 & 0 \end{pmatrix}$	8	0,085	0,1	0.05
23,12%	$\begin{pmatrix} 0 & 244 & 0 & 0 \\ 0 & 80 & 0 & 0 \\ 0 & 16 & 0 & 0 \\ 0 & 6 & 0 & 0 \end{pmatrix}$	8	0,20	0.001	0.05
92,77%	$\begin{pmatrix} 235 & 9 & 0 & 0 \\ 3 & 70 & 7 & 0 \\ 0 & 0 & 16 & 0 \\ 0 & 0 & 6 & 0 \end{pmatrix}$	8	100,28	0.1	0.000000005
92,77%	$\begin{pmatrix} 234 & 10 & 0 & 0 \\ 4 & 68 & 2 & 6 \\ 0 & 0 & 13 & 3 \\ 0 & 0 & 6 & 6 \end{pmatrix}$	10	37,17	0.1	0.000000005
91,33%	$\begin{pmatrix} 238 & 6 & 0 & 0 \\ 10 & 62 & 8 & 0 \\ 0 & 0 & 16 & 0 \\ 0 & 0 & 6 & 0 \end{pmatrix}$	9	45,95	0.1	0.000000005

Table 6: Iteration 1 Results

Iteration 2:

Accuracy	Confusion Matrix	Nodes in the hidden Layer	Time taken to Train(seconds)	Learning Rate(α)	Threshold(ε)
Prior to Training: 3,86%	$\begin{pmatrix} 0 & 0 & 0 & 740 \\ 0 & 0 & 0 & 213 \\ 0 & 0 & 0 & 44 \\ 0 & 0 & 0 & 40 \end{pmatrix}$				
After Training: 71,36%	$\begin{pmatrix} 740 & 0 & 0 & 0 \\ 213 & 0 & 0 & 0 \\ 44 & 0 & 0 & 0 \\ 40 & 0 & 0 & 0 \end{pmatrix}$				
Validation: 69,94%	$\begin{pmatrix} 242 & 0 & 0 & 0 \\ 80 & 0 & 0 & 0 \\ 8 & 0 & 0 & 0 \\ 16 & 0 & 0 & 0 \end{pmatrix}$	8	0,093	0,1	0.05
4,62%	$\begin{pmatrix} 0 & 0 & 0 & 242 \\ 0 & 0 & 0 & 80 \\ 0 & 0 & 0 & 8 \\ 0 & 0 & 0 & 16 \end{pmatrix}$	8	0,11	0.001	0.05
94,51%	$\begin{pmatrix} 233 & 8 & 1 & 0 \\ 7 & 70 & 0 & 3 \\ 0 & 0 & 8 & 0 \\ 0 & 0 & 0 & 16 \end{pmatrix}$	8	95,82	0.1	0.000000005
92,49%	$\begin{pmatrix} 235 & 6 & 1 & 0 \\ 7 & 69 & 1 & 3 \\ 0 & 1 & 7 & 0 \\ 0 & 0 & 1 & 15 \end{pmatrix}$	10	29,82	0.1	0.000000005
94,22%	$\begin{pmatrix} 235 & 6 & 1 & 0 \\ 7 & 69 & 1 & 3 \\ 0 & 1 & 7 & 0 \\ 0 & 0 & 1 & 15 \end{pmatrix}$	9	50,51	0.1	0.000000005

Table 7: Iteration 2 Results

Iteration 3:

Accuracy	Confusion Matrix	Nodes in the hidden Layer	Time taken to Train(seconds)	Learning Rate(α)	Threshold(ε)
Prior to Training: 3,57%	$\begin{pmatrix} 0 & 0 & 0 & 724 \\ 0 & 0 & 0 & 228 \\ 0 & 0 & 0 & 48 \\ 0 & 0 & 0 & 37 \end{pmatrix}$				
After Training: 69,82%	$\begin{pmatrix} 724 & 0 & 0 & 0 \\ 228 & 0 & 0 & 0 \\ 48 & 0 & 0 & 0 \\ 37 & 0 & 0 & 0 \end{pmatrix}$				
Validation: 71,97%	$\begin{pmatrix} 249 & 0 & 0 & 0 \\ 76 & 0 & 0 & 0 \\ 9 & 0 & 0 & 0 \\ 12 & 0 & 0 & 0 \end{pmatrix}$	8	0,12	0,1	0.05
4,05%	$\begin{pmatrix} 2 & 0 & 0 & 247 \\ 1 & 0 & 0 & 75 \\ 1 & 0 & 0 & 8 \\ 0 & 0 & 0 & 12 \end{pmatrix}$	8	0,098	0.001	0.05
92,49%	$\begin{pmatrix} 243 & 5 & 1 & 0 \\ 7 & 62 & 3 & 4 \\ 0 & 3 & 3 & 3 \\ 0 & 0 & 0 & 12 \end{pmatrix}$	8	92,24	0.1	0.000000005
95,95%	$\begin{pmatrix} 248 & 0 & 1 & 0 \\ 5 & 66 & 0 & 5 \\ 0 & 0 & 6 & 3 \\ 0 & 0 & 0 & 12 \end{pmatrix}$	10	95,15	0.1	0.000000005
97,98%	$\begin{pmatrix} 246 & 3 & 0 & 0 \\ 2 & 72 & 2 & 0 \\ 0 & 0 & 9 & 0 \\ 0 & 0 & 0 & 12 \end{pmatrix}$	9	95,16	0.1	0.000000005

Table 8: Iteration 3 Results

Iteration 4:

Accuracy	Confusion Matrix	Nodes in the hidden Layer	Time taken to Train(seconds)	Learning Rate(α)	Threshold(ε)
Prior to Training: 4,05%	$\begin{pmatrix} 0 & 0 & 723 & 0 \\ 0 & 0 & 231 & 0 \\ 0 & 0 & 42 & 0 \\ 0 & 0 & 41 & 0 \end{pmatrix}$				
After Training: 83,12%	$\begin{pmatrix} 694 & 26 & 2 & 1 \\ 82 & 145 & 3 & 1 \\ 16 & 3 & 18 & 5 \\ 3 & 12 & 21 & 5 \end{pmatrix}$				
Validation: 86,42%	$\begin{pmatrix} 233 & 6 & 0 & 0 \\ 24 & 57 & 0 & 0 \\ 4 & 1 & 8 & 0 \\ 0 & 4 & 8 & 1 \end{pmatrix}$	8	0,45	0,1	0.05
4,61%	$\begin{pmatrix} 3 & 0 & 236 & 0 \\ 0 & 0 & 81 & 0 \\ 0 & 0 & 13 & 0 \\ 0 & 0 & 13 & 0 \end{pmatrix}$	8	0,101	0.001	0.05
93,64%	$\begin{pmatrix} 230 & 8 & 1 & 0 \\ 7 & 73 & 1 & 0 \\ 0 & 0 & 10 & 3 \\ 0 & 2 & 0 & 11 \end{pmatrix}$	8	92,78	0.1	0.000000005
96,82%	$\begin{pmatrix} 248 & 0 & 1 & 0 \\ 5 & 66 & 0 & 5 \\ 0 & 0 & 6 & 3 \\ 0 & 0 & 0 & 12 \end{pmatrix}$	10	95,65	0.1	0.000000005
96,53%	$\begin{pmatrix} 234 & 4 & 1 & 0 \\ 4 & 77 & 0 & 0 \\ 0 & 0 & 12 & 1 \\ 0 & 1 & 1 & 11 \end{pmatrix}$	9	95,48	0.1	0.000000005

Table 9: Iteration 4 Results

From the above tables, we can undoubtedly conclude that the algorithm is learning since the accuracy prior to training is significantly lower than the error post training. Since we are certain the algorithm is learning correctly, we can discuss the results of the validation data.

The first notable outlier, is the model with the learning rate(α) of 0.001 which consistently performed poorly. The poor performance is a direct result of the learning rate being too small and as a result, the gradient descent algorithm converges at an extremely slow speed. Consequently, the weight updates at each step are relatively small and insignificant, which ensures that the threshold(ε) is never reached. Instead we reach the maximum epochs and the algorithm terminates. Hence no significant learning occurs, resulting in the accuracy being extremely low. We can now exclude this model from the discussion.

If we observe the first model, the threshold is significantly larger than the other 3 remaining models. This has a direct effect on the performance of the model. In every iteration , the first model only predicts that the outcome would be 0. Since our dataset is skewed, with majority of the targets being 0, the model achieves a relatively good accuracy. However, if we look at the recall score of the model, on average it is 76.8%, 25.84%, 50% and 25% for classes 0,1,2 and 3 respectively. This clearly indicates how poorly the model correctly predicts the classes other than 0. This error can be due to the fact that since the threshold is so small , the algorithm converges quite rapidly and reaches this threshold prematurely, consequently leading to the model not learning holistically. Thus we end up with an misleadingly high accuracy because of the skewness of our dataset.

If we look at the 3 remaining models, they all have the same learning rate and threshold. However, the amount of nodes in the hidden layer vary. On average we conclude that the model with 9 neurons within the hidden layer performs the best. This value lies between the 8 and 10 neurons of the other models. A possible explanation for this is that when we increase the neurons within a layer, we enable the model to learn greater intricacies within the dataset. Hence we have improved performance. You may argue that with that logic, the model with 10 neurons in the hidden layer should perform better. However that is not necessarily the case since the factor of overfitting may come into play. As a result, when the model is asked to classify data it was not trained on, the accuracy decreases slightly due to the overfitting that occurs.

The details of the models on the validation data are summarised in the table below.

Threshold and Learning Rate	Time taken to train(seconds)	Accuracy	Average Accuracy
Learning Rate = 0,1 Threshold = 0.05 Nodes in hidden layer = 8	0,19	Iteration 1: 73,70 % Iteration 2: 69,94% Iteration 3: 71,97% Iteration 4: 86,42%	75,50%
Learning Rate = 0.001 Threshold = 0.05 Nodes in hidden layer = 8	0,13	Iteration 1: 23,12 % Iteration 2: 4,62 % Iteration 3: 4,05% Iteration 4: 4,61%	9,1%
Learning Rate = 0.1 Threshold = 0.000000005 Nodes in hidden layer = 8	95,28	Iteration 1: 92,77% Iteration 2: 94,51% Iteration 3: 92,49% Iteration 4: 93,64 %	93,35%
Learning Rate = 0.1 Threshold = 0.000000005 Nodes in hidden layer = 10	64,45	Iteration 1: 92,77% Iteration 2: 92,49 % Iteration 3: 95,95 % Iteration 4: 96,82%	94,51%
Learning Rate = 0.1 Threshold = 0.000000005 Nodes in hidden layer = 9	71,78	Iteration 1: 91,33% Iteration 2: 94,22% Iteration 3: 97,98 % Iteration 4: 96,53%	95,02%

Table 10: Summary of Validation Results

From the above table, we can conclude that the optimised hyper-parameters correspond to the values α = 0,1 , ε = 0.000000005 and 9 neurons within the hidden layer.
We will use the model corresponding to these values on our testing data.

Additional Algorithms:

4.4 Decision Trees:

We used the sklearn built in model to implement decision trees. We ran the algorithm four times and the training results can be captured as follows:

Accuracy	Time to train(seconds)	Confusion Matrix
100%	0,03	$\begin{pmatrix} 966 & 0 & 0 & 0 \\ 0 & 309 & 0 & 0 \\ 0 & 0 & 55 & 0 \\ 0 & 0 & 0 & 53 \end{pmatrix}$
100%	0,04	$\begin{pmatrix} 966 & 0 & 0 & 0 \\ 0 & 309 & 0 & 0 \\ 0 & 0 & 55 & 0 \\ 0 & 0 & 0 & 53 \end{pmatrix}$
100%	0,04	$\begin{pmatrix} 966 & 0 & 0 & 0 \\ 0 & 309 & 0 & 0 \\ 0 & 0 & 55 & 0 \\ 0 & 0 & 0 & 53 \end{pmatrix}$
100%	0,03	$\begin{pmatrix} 966 & 0 & 0 & 0 \\ 0 & 309 & 0 & 0 \\ 0 & 0 & 55 & 0 \\ 0 & 0 & 0 & 53 \end{pmatrix}$

Table 11: Decision Trees Training Results

As we can see, the model achieves a 100% accuracy every single time on the training data. This is impressive, however, we will test of overfitting has occurred when we observe the results on the test data in the next section.

4.5 Support Vector Machines:

We used the sklearn built in model to implement a support vector machine. The model focuses on separating data-points using a hyperplane. The best hyperplane would be the one that ensures the largest separation between classes.
We ran the algorithm four times and the training results can be captured as follows:

Accuracy	Time taken to train(seconds)	Confusion Matrix
86,70%	0,04	$\begin{pmatrix} 913 & 99 & 15 & 1 \\ 44 & 206 & 3 & 7 \\ 7 & 3 & 37 & 2 \\ 2 & 1 & 0 & 43 \end{pmatrix}$
85,39%	0,04	$\begin{pmatrix} 910 & 107 & 18 & 2 \\ 45 & 203 & 5 & 10 \\ 6 & 4 & 37 & 3 \\ 2 & 0 & 0 & 31 \end{pmatrix}$
85,62%	0,05	$\begin{pmatrix} 917 & 114 & 18 & 0 \\ 42 & 187 & 1 & 11 \\ 6 & 1 & 40 & 3 \\ 2 & 1 & 0 & 40 \end{pmatrix}$
85,98%	0,05	$\begin{pmatrix} 915 & 105 & 18 & 3 \\ 44 & 193 & 4 & 7 \\ 6 & 3 & 35 & 1 \\ 2 & 1 & 0 & 46 \end{pmatrix}$

Table 12: Support Vector Machines Training Results

As we can observe that the model achieves relatively the same accuracy on every iteration. The model has great stability. The slight variation is due to the variation in training data at each instance. We will test it’s performance on the test data and report the results in the next chapter.

5 Test Results

5.1 Logistic Regression

In the previous discussion we found that the model with the greatest average accuracy had/hyper-parameters of learning rate(α) = 0.01 and threshold(ε) = 0.000000005.

We used this model to ascertain the performance on the test data over four iterations. The results were as follows:

Accuracy	Precision	Recall	Confusion Matrix
66,09%	Class 0: 97,85% Class 1: 0 % Class 2: 0 % Class 3: 0 %	Class 0: 67,06% Class 1: 0 % Class 2: 0 % Class 3: 0 %	$\begin{pmatrix} 228 & 84 & 10 & 18 \\ 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$
73,04%	Class 0: 99,21% Class 1: 2,74 % Class 2: 0 % Class 3: 0 %	Class 0: 73,75% Class 1: 33,33 % Class 2: 0 % Class 3: 0 %	$\begin{pmatrix} 250 & 71 & 7 & 11 \\ 2 & 2 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$
70,43%	Class 0: 92,99% Class 1: 5,26% Class 2: 0 % Class 3: 0 %	Class 0:74,45% Class 1: 17,39 % Class 2: 0 % Class 3: 0 %	$\begin{pmatrix} 239 & 72 & 4 & 6 \\ 17 & 4 & 0 & 2 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$
67,25%	Class 0: 86,01% Class 1: 31,51% Class 2: 0 % Class 3: 0 %	Class 0: 74,11% Class 1: 36,51 % Class 2: 0 % Class 3: 0 %	$\begin{pmatrix} 209 & 50 & 10 & 13 \\ 34 & 23 & 2 & 4 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$

Table 13: Logistic Regression Test Results

As we can explicitly see from the above table, the model achieves a fair accuracy. On average , the accuracy is 69,20%. However, this accuracy is misleading. We can clearly observe from the confusion matrix as well as recall and precision that the model only predicts one of two classes, either Class 0 or Class 1. Since our dataset is skewed towards class 0, even though the algorithm classifies most points as belonging to class 0, the accuracy will remain relatively high. This is when we look to our other measures of performance, namely recall and precision.

If we look at the recall in each instance for class 2 and 3 , we immediately notice that it is always 0%. We also notice that for class 1, on average the precision is 21,81%. This implies that on average, we only correctly classify 21,81% of data-points that belong to class 1 and we never classify any data-points correctly to either class 2 or class 3.

The graph below is a graphical representation of the above observations.

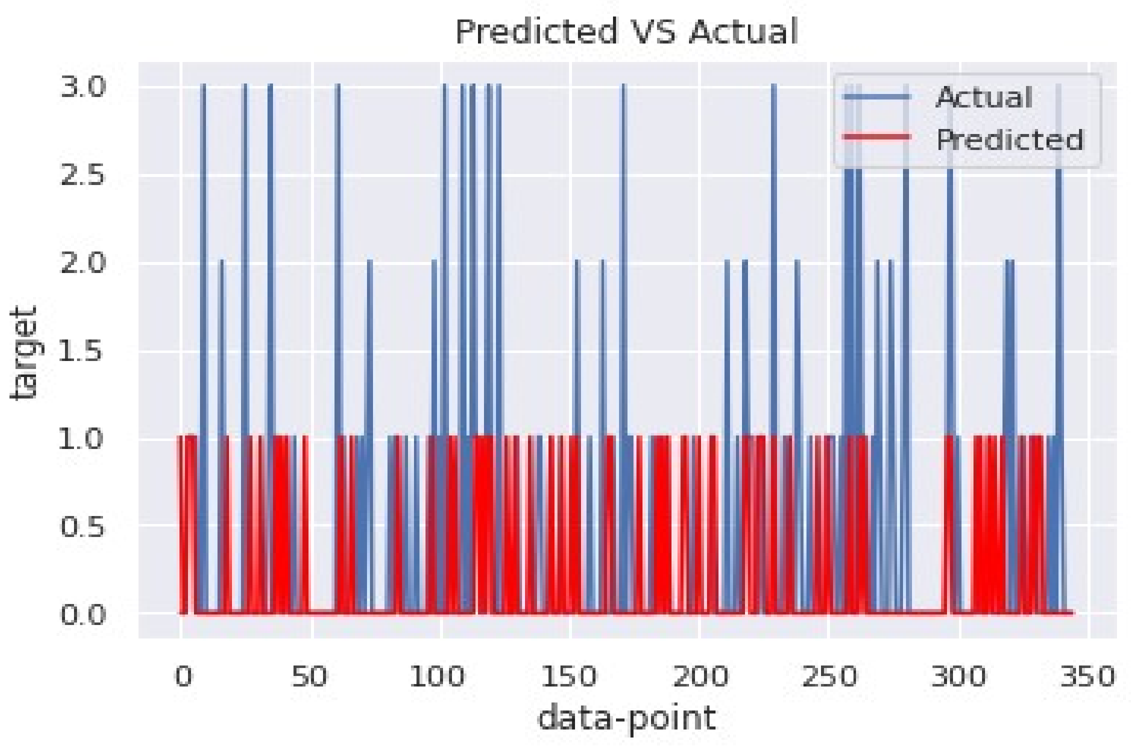


Figure 8: Logistic Regression Actual vs Predicted Graph

As we can see from the graph, the predicted(red) graph never takes on the class values 2 or 3. We can also gauge how often these incorrect predictions occur from the graph.

A viable explanation for the behaviour of the model could be the fact that since our dataset is so biased towards a certain class, while the model is training, it is not given a data-point which belongs to these other classes and as a result the model tends to “forget” about them and as a result only predicts the two common classes.
However, we are certain that the one verse all representation has been implemented correctly, since in the discussion about the validation data, the model was predicting other classes as well.
A possible solution to this problem could be to generate more data-points that belong to the minority classes in order to even out the distribution. This will provide the model with adequate data from each class to find the underlying trends within the data.

In our implementation , we stopped the algorithm after a maximum of a 1000 iterations were reached, or if the norm between consecutive weight updates was less than the threshold. If we enable the algorithm to run for a longer period of time, the better it will learn the data and consequently perform better as well.

In conclusion, the model performed relatively well given the dataset. It enabled us to understand the importance of having a diverse dataset.

5.2 Naive Bayes

In the previous section we discussed the performance of the naive bayes model on the training data. Since naive bayes has no hyper-parameters, we merged the training and validation data to give us a larger training set.
We then ran the algorithm four times on different sets of test data and the results were as follows:

Accuracy	Precision	Recall	Confusion Matrix
84,93%	Class 0: 91,76%	Class 0: 95,12%	$\begin{pmatrix} 234 & 12 & 0 & 0 \\ 21 & 52 & 0 & 0 \\ 0 & 13 & 2 & 0 \\ 0 & 6 & 0 & 5 \end{pmatrix}$
	Class 1: 62,65 %	Class 1: 71,23 %	
	Class 2: 100 %	Class 2: 13,33 %	
	Class 3: 100 %	Class 3: 45,46 %	
85,51%	Class 0: 88,84%	Class 0: 96,54%	$\begin{pmatrix} 223 & 8 & 0 & 0 \\ 28 & 62 & 2 & 0 \\ 0 & 6 & 4 & 0 \\ 0 & 6 & 0 & 6 \end{pmatrix}$
	Class 1: 75,61 %	Class 1: 67,39%	
	Class 2: 66,67 %	Class 2: 40,00 %	
	Class 3: 100 %	Class 3: 50,00 %	
85,80%	Class 0: 89,97%	Class 0: 97,08%	$\begin{pmatrix} 233 & 7 & 0 & 0 \\ 26 & 59 & 0 & 0 \\ 0 & 9 & 1 & 0 \\ 0 & 7 & 0 & 3 \end{pmatrix}$
	Class 1:71,95%	Class 1: 69,41 %	
	Class 2: 100 %	Class 2: 10 %	
	Class 3: 100 %	Class 3: 30 %	
82,61%	Class 0: 90,84%	Class 0: 97,02%	$\begin{pmatrix} 228 & 7 & 0 & 0 \\ 23 & 55 & 2 & 0 \\ 0 & 15 & 1 & 0 \\ 0 & 13 & 0 & 1 \end{pmatrix}$
	Class 1: 61,11%	Class 1: 68,75 %	
	Class 2: 33,33 %	Class 2: 6,25 %	
	Class 3: 100 %	Class 3: 7,14 %	

Table 14: Naive Bayes Test Results

The above table gives us a clear indication of the performance of the model. The model achieves a steady accuracy throughout and has an average accuracy of 84,71%.
The model also achieves an average precision score of 90.35%, 67.83% , 75% and 100% respectively for each class. Thus we have a strong percentage of values in each class that our model classifies as positive and we thought it was positive.

However we also look at the average recall which was 94.44%, 69.20%, 17.40% and 33.15% respectively for each class. This gives us the percentage of positives we classified correctly. Immediately we notice that class 2 and 3 have a significantly lower average recall in comparison to class 0 and class 1, this is a direct result of the skewness and incongruence of our dataset.
The bias of our dataset to Class 0 and Class 1 ensures that the majority of the training examples belong to either of those classes. As a result, their conditional probabilities are much larger in the likelihood table which ends up manipulating the posterior probabilities in the favour of these two classes and as a result, we tend to classify these classes more correctly than the other two classes.
This will especially be the result in data-points that are considered to be outliers. Since they do not have the usual features of a certain class, their probabilities will be skewed as a result of the likelihood table, leading to misclassification of the data-point.

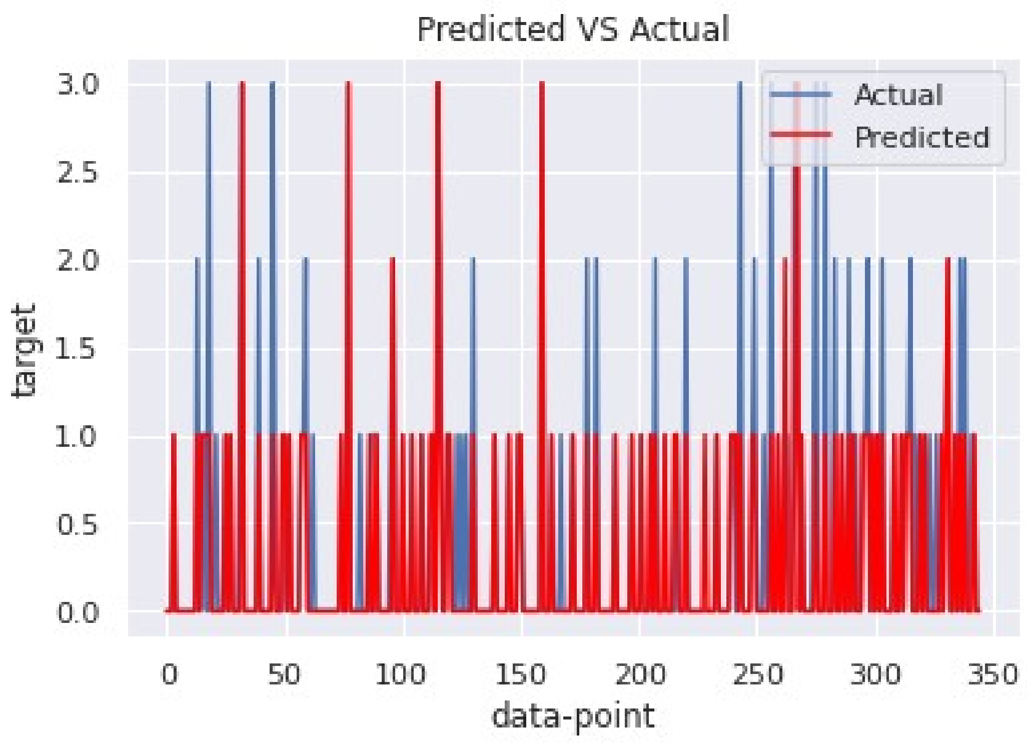


Figure 9: Naive Bayes: Actual vs Predicted Graph

From the graph we can deduce how certain data-points were misclassified.

A possible solution to this problem could be to introduce more data-points that belong to the other classes, or to remove data-points that belong to the majority classes, class 0 and 1, ultimately ensuring a Gaussian distribution of the data which will lead to even more accurate predictions.

Another factor we had to take into account was the zero-frequency problem. We accounted for it using Laplacian smoothing. However this still does not counter the strength that the other likelihoods had and was insignificant in the greater scheme.

Finally a limitation of the model is the independence assumption. This is incorrect since in almost all real life situations, it is nearly impossible to attain a set of predictors that are completely independent.

In conclusion, the simplicity of Naive Bayes ensures it has a relatively high and stable predictive power which could be immensely increased if we had a gaussian spread of the data.

5.3 Neural Network

From the previous discussion regarding the validation data, we came to the conclusion that the model with the optimised hyper-parameters had the following values: learning rate(α) = 0.1 ,threshold(ε) = 0.000000005 and 9 neurons within the hidden layer.
We will use this model to evaluate the performance of the neural network on the test data.

Accuracy	Precision	Recall	Confusion Matrix
88,70%	Class 0: 88,04% Class 1: 93,18 % Class 2: 100 % Class 3: 82,35 %	Class 0: 98,78% Class 1: 55,41% Class 2: 88,89 % Class 3: 87,50 %	$\begin{pmatrix} 243 & 1 & 0 & 2 \\ 33 & 41 & 0 & 0 \\ 0 & 0 & 8 & 1 \\ 0 & 2 & 0 & 4 \end{pmatrix}$
91,88%	Class 0: 94,14% Class 1: 85,53 % Class 2: 82,35 % Class 3: 100 %	Class 0: 97,40% Class 1: 81,25% Class 2: 77,78% Class 3: 81,25%	$\begin{pmatrix} 225 & 5 & 1 & 0 \\ 14 & 65 & 1 & 0 \\ 0 & 4 & 14 & 0 \\ 0 & 2 & 1 & 13 \end{pmatrix}$
94,49%	Class 0: 96,23% Class 1: 89,47% Class 2: 92,86% Class 3: 93,75 %	Class 0: 97,87% Class 1: 86,08 % Class 2: 86,67 % Class 3: 93,75 %	$\begin{pmatrix} 230 & 5 & 0 & 0 \\ 9 & 68 & 1 & 1 \\ 0 & 2 & 13 & 0 \\ 0 & 1 & 0 & 15 \end{pmatrix}$
94,78%	Class 0: 98,33% Class 1: 83,14% Class 2: 100 % Class 3: 100 %	Class 0: 95,14% Class 1: 94,52 % Class 2: 100 % Class 3: 84,62 %	$\begin{pmatrix} 235 & 12 & 0 & 0 \\ 4 & 69 & 0 & 0 \\ 0 & 0 & 12 & 0 \\ 0 & 2 & 0 & 11 \end{pmatrix}$

Table 15: Neural Network Test Results

From the table we can deduce that the neural network has an extremely high accuracy rate. On average, the Neural Network achieved an accuracy of 92,46%. This is indicative of the predictive power Neural Networks possess.

If we study the other performance metrics, namely the precision and recall we obtain the following: On average, the precision per class was 94.19%, 87.83%, 93.80% and 94.03% respectively. This implies that the on average our percentage classification of each class was extremely high. However, in order to determine the percentage of positives we classified correctly, we look at the recall. On average, the recall per class was 97.30%, 79.32%, 88.34% and 86.78% respectively. From the recall we can deduce that most of the time the Neural Network classifies a data-point correctly. The high precision tell us that the model always returns more relevant results than irrelevant results and the high recall tells us that the model returns mostly relevant results. In our case, relative results are denoted as correctly classified data-points.

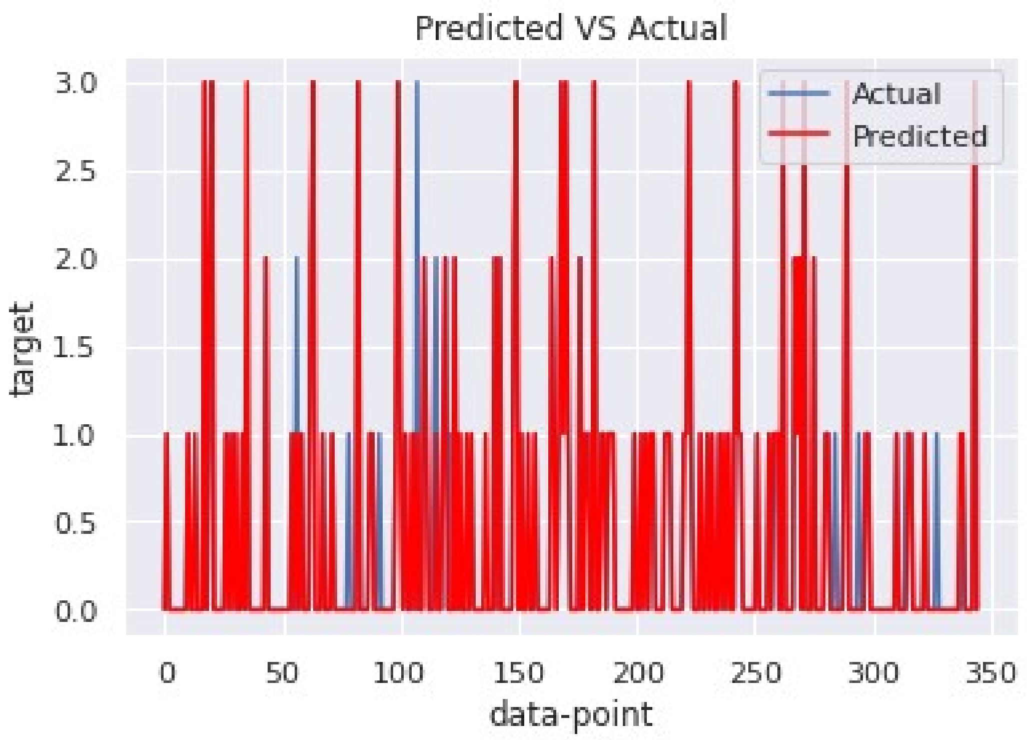


Figure 10: Neural Network: Actual vs Predicted

If we look at the figure above, we can see that the predicted curve(red) is almost a perfect fit for the blue(actual) curve. This shows us how accurate the neural network is in performing its classification task.
Even though our dataset is skewed, the neural network was able to capture the different classes.

A reason behind the accuracy could lie in the epochs. If we allowed the algorithm to run even longer, or made the threshold(ε) even smaller, the model would have learnt even better and would have reported an increased accuracy.
One outlier in the table above is the first accuracy, we reported an accuracy of 88.7% which is by no means low, however in comparison to the others, it is low. An explanation for this could be the data that the network was fed during training. Since we randomly shuffled the data, it could have received an greatly incoherently distributed dataset which affected the respective weight updates. As a result, the accuracy is not as high as the other iterations.

In conclusion, the Neural Network performed extremely well and was able to classify majority of all the classes correctly.

5.4 Decision Tree

The decision tree experience a 100% accuracy in every iteration with the training data. In order to test if overfitting has occurred, we will now measure the performance of the model on the testing data.
The results are captured below:

Accuracy	Precision	Recall	Confusion Matrix
98,55%	Class 0: 99,61% Class 1: 95,59 % Class 2: 90,91% Class 3: 100 %	Class 0: 99,60% Class 1: 97,01% Class 2: 90,91 % Class 3: 92,86 %	$\begin{pmatrix} 252 & 1 & 0 & 0 \\ 1 & 65 & 1 & 0 \\ 0 & 1 & 10 & 0 \\ 0 & 1 & 0 & 13 \end{pmatrix}$
96,23%	Class 0: 99,59% Class 1: 91,43 % Class 2: 78,57 % Class 3: 80,00 %	Class 0: 97,61% Class 1: 96,97% Class 2: 73,33% Class 3: 92,31%	$\begin{pmatrix} 245 & 4 & 2 & 0 \\ 1 & 64 & 1 & 0 \\ 0 & 1 & 11 & 3 \\ 0 & 1 & 0 & 12 \end{pmatrix}$
97,68%	Class 0: 99,60% Class 1: 92,11% Class 2: 100% Class 3: 88,89 %	Class 0: 98,82% Class 1: 97,22 % Class 2: 75,00 % Class 3:100%	$\begin{pmatrix} 250 & 3 & 0 & 0 \\ 1 & 70 & 0 & 1 \\ 0 & 3 & 9 & 0 \\ 0 & 0 & 0 & 8 \end{pmatrix}$
97,97%	Class 0: 98,32% Class 1: 97,56% Class 2: 88,89 % Class 3: 100 %	Class 0: 100% Class 1: 94,12 % Class 2: 80,00% Class 3: 100,00 %	$\begin{pmatrix} 234 & 0 & 0 & 0 \\ 4 & 80 & 1 & 0 \\ 0 & 2 & 8 & 0 \\ 0 & 0 & 0 & 16 \end{pmatrix}$

Table 16: Decision Tree Test Results

As we can deduce from the above, the decision tree model performs extremely well on the test data as well. The model has an average test accuracy of 97,61%. However we cannot look at the accuracy in isolation, we consequently look at the other two reported performance metrics as well, namely precision and recall.

On average, the precision score per class is 99.28%, 94.17%, 89.59% and 99.22% respectively. The recall score per class is 99.00%, 96.33%, 79.91% and 96.30% respectively. From these two metrics we can deduce the following, the true number of positives in comparison to the positives that were predicted by the model are extremely close. Thus our average precision scores per class are justifiably high. Considering the recall scores, we can deduce that most of the time, the model classified a positive correctly. The only recall score that is incoherent with the other results, is the recall for Class 2, which is 79.91% on average. A possible explanation for this as well as the incorrect predictions in general stem from the fact that some data-points could be outliers. This meaning that the values the features take on are in conflict with the usual values other members of those classes take on. However we accept this, since achieving a 100% accuracy is an unrealistic expectation.

Below is a graph to visualise the performance of the model.

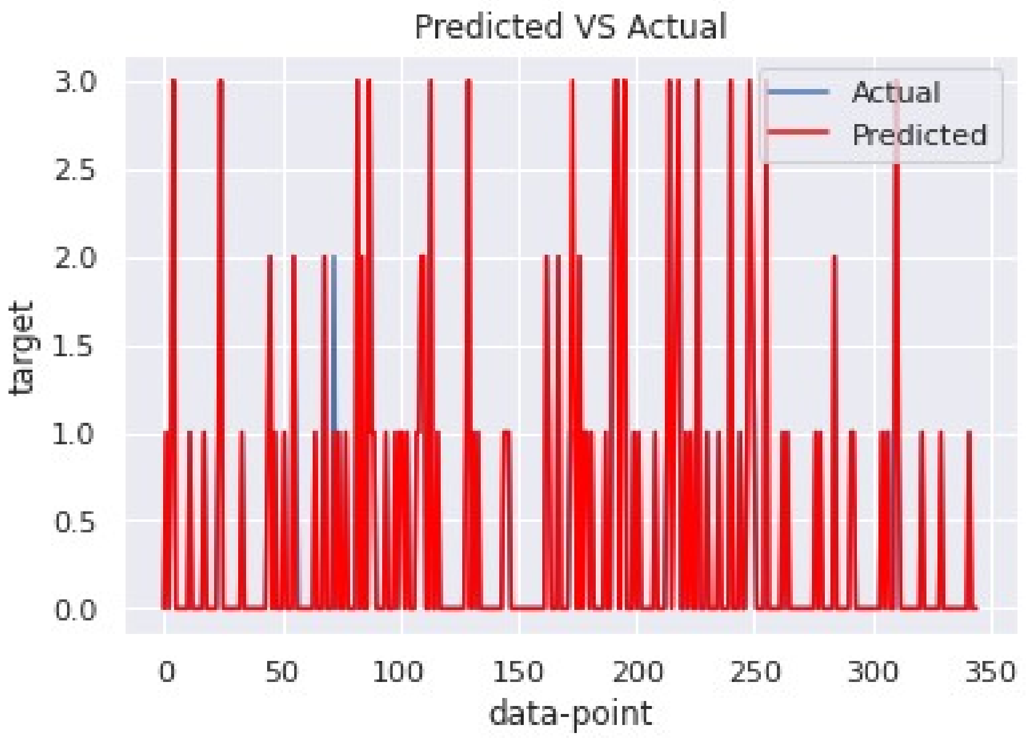


Figure 11: Decision Tree: Predicted vs Actual Graph

From this figure, we can visually confirm that the model predicts almost every data-point correctly since the predicted(red) graph almost totally envelops the actual(blue) graph. The instances where the blue graph is visible are the instances whereby a misclassification occurred.

In conclusion, the decision tree model achieved an astounding accuracy and is capable of capturing the trends within the dataset effectively.

5.5 Support Vector Machine (SVM)

In the previous section, we saw that the SVM achieved a stable accuracy on the training data. However, we are going to test the reliability of the model by presenting it the test data and delving into the performance metrics of the model.

The results are as follows:

Accuracy	Precision	Recall	Confusion Matrix
82,32%	Class 0: 89,75%	Class 0: 87,60%	$\begin{pmatrix} 219 & 28 & 3 & 0 \\ 18 & 47 & 0 & 1 \\ 5 & 2 & 8 & 1 \\ 2 & 1 & 0 & 10 \end{pmatrix}$
	Class 1: 60,26%	Class 1: 71,21%	
	Class 2: 72,73%	Class 2: 50,00 %	
	Class 3: 83,33%	Class 3: 76,92%	
86,09%	Class 0: 95,32%	Class 0: 88,53%	$\begin{pmatrix} 224 & 22 & 7 & 0 \\ 11 & 52 & 2 & 3 \\ 0 & 0 & 9 & 3 \\ 0 & 0 & 0 & 12 \end{pmatrix}$
	Class 1: 70,27 %	Class 1: 76,47%	
	Class 2: 50,00 %	Class 2: 75,00%	
	Class 3: 66,67%	Class 3: 100%	
83,77%	Class 0: 94,07%	Class 0: 86,05%	$\begin{pmatrix} 222 & 28 & 8 & 0 \\ 12 & 48 & 0 & 4 \\ 1 & 1 & 9 & 1 \\ 1 & 0 & 0 & 10 \end{pmatrix}$
	Class 1: 62,34%	Class 1: 75,00 %	
	Class 2: 52,94%	Class 2: 75,00 %	
	Class 3: 66,67 %	Class 3: 90,91 %	
86,67%	Class 0: 96,58%	Class 0: 86,92%	$\begin{pmatrix} 226 & 30 & 4 & 0 \\ 8 & 52 & 2 & 0 \\ 0 & 0 & 10 & 1 \\ 0 & 1 & 0 & 11 \end{pmatrix}$
	Class 1: 62,65%	Class 1: 83,87 %	
	Class 2: 62,5 %	Class 2: 90,91 %	
	Class 3: 91,67%	Class 3: 91,67 %	

Table 17: Support Vector Machine Test Results

From the above table we can conclude that the model achieves the relatively same accuracy that it did with the training data. On average the model achieved 84.71%. This is an acceptable accuracy. However, since our dataset is skewed, we have to look at the other reported metrics, namely precision and recall.

On average, the precision score per class can be reported as follows, 93.93%, 63.88%, 59.43% and 77.09% respectively. On average, the recall score per class can be reported as follows, 87.28%, 76.64%, 72.73% and 89.88% respectively. From this we can delve into the actual performance of the actual model. If we look at the average precision, we can see that the model achieves a high precision for the dominant class , which is class 0. However for the three remaining classes it achieves a relatively low precision. However , if we look at the recall scores per class, classes 1 and 3 achieve the highest recalls which implies that those classes have the highest percentage of correctly classified data-points, whereas the recall scores for Class 1 and 2 are not extremely low, but are lower than the other classes which implies that the model is less accurate in predicting these classes than the other two.

A graphical representation of these results are represented graphically below.

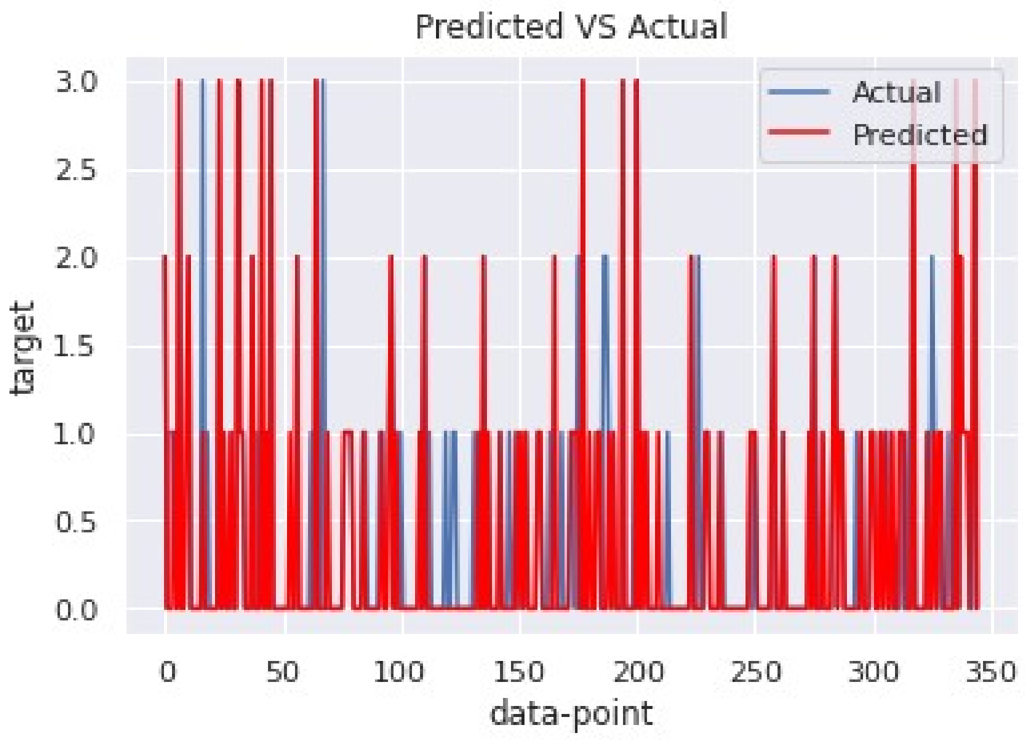


Figure 12: SVM Predicted vs Actual Graph

Graphically, we can deduce that the model predicts the outputs relatively well. The incorrect predictions can be seen at the points where the actual(blue) graph is visible and not overlapped by the predicted(red) graph.
There is a greater proportion of incorrect classifications, hence the actual graph is not completely enveloped by the predicted(red) graph.

These inaccuracies can be accounted for. Since SVM's perform poorly when classes overlap, the data-points which contribute to the decrease of the models predictive accuracy are data-points that can be considered outliers/noise. The features of these data-points take on values that are usually not associated with their class, as a result, the hyperplanes position is distorted and we end up with a larger proportion of incorrectly classified points.

In conclusion, SVM’s provide us with a great accuracy, however the model is extremely sensitive to outliers and noise. Thus, if we want to increase the accuracy of the model, we can remove the noise and prevent the distortion of the hyperplane boundaries.

6 Comparison of models

The average of the test metrics per model are captured in the table below.

Model	Accuracy	Precision	Recall
Logistic Regression	69,20%	Class 0: 94,02%	Class 0: 72,34%
		Class 1: 21,81%	Class 1: 21,81%
		Class 2: 0%	Class 2: 0%
		Class 3: 0%	Class 3: 0%
Naive Bayes	84,71%	Class 0: 90,35%	Class 0: 94,44%
		Class 1: 67,83%	Class 1: 69,20%
		Class 2: 75,00%	Class 2: 17,40%
		Class 3: 100%	Class 3: 33,15%
Neural Network	92,46%	Class 0: 94,19%	Class 0: 97,30%
		Class 1: 87,83%	Class 1: 79,32%
		Class 2: 93,80%	Class 2: 88,34%
		Class 3: 94,03%	Class 3: 86,78%
Decision Tree	97,61%	Class 0: 99,28%	Class 0: 99,00%
		Class 1: 94,17%	Class 1: 96,33%
		Class 2: 89,59%	Class 2: 79,91%
		Class 3: 99,22%	Class 3: 96,30%
Support Vector Machine	84,71%	Class 0: 93,93%	Class 0: 87,28%
		Class 1: 63,88%	Class 1: 76,64%
		Class 2: 59,43%	Class 2: 72,73%
		Class 3: 77,09%	Class 3: 89,88%

Table 18: Summary of Test Results per Model

We will use the above table for a comparative discussion between the models.

First we delve into the accuracy of each model. From the table we blatantly note that the Decision Tree and Neural Network achieve the highest accuracies whereas Naive Bayes and the Support Vector Machine achieve a moderate accuracy and finally the most underwhelming performance is achieved by the Logistic Regression model. These results are captured graphically below.

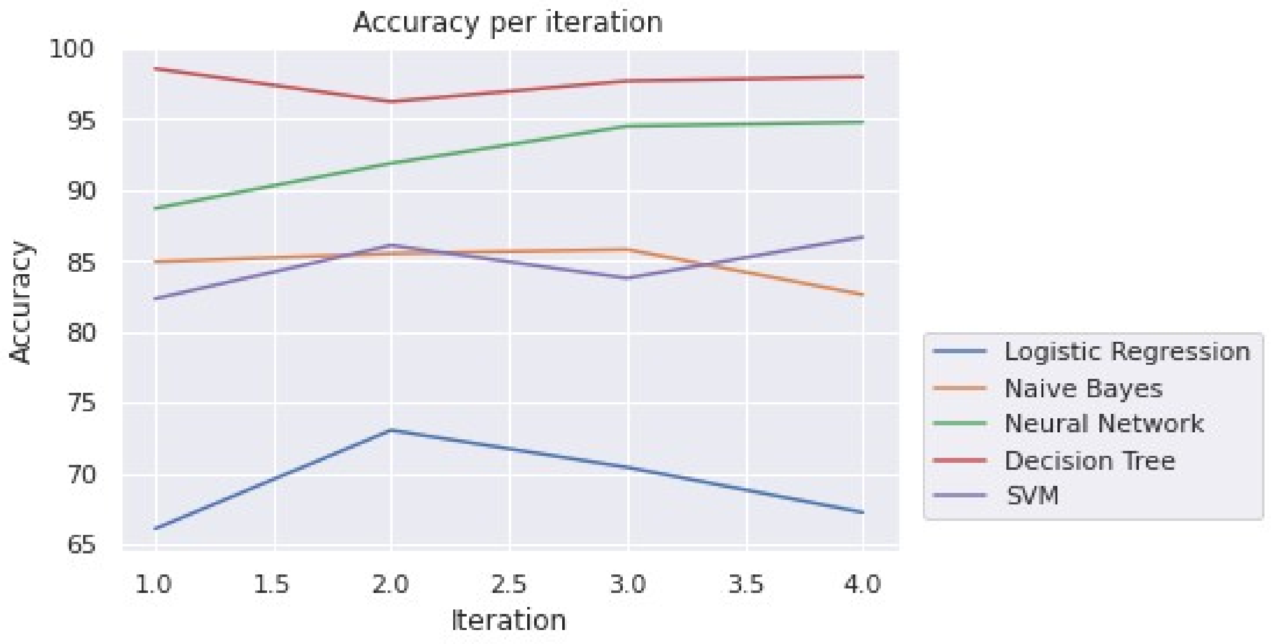


Figure 13: Average Accuracy Per Model

Since the accuracy of the model is dependent on how many data-points are classified correctly, we also need to take a look at that the precision and recall of each model.

Firstly we will look at the precision and recall of each model comparatively. Below are graphical representations which we will refer to in the discussion of the metrics.

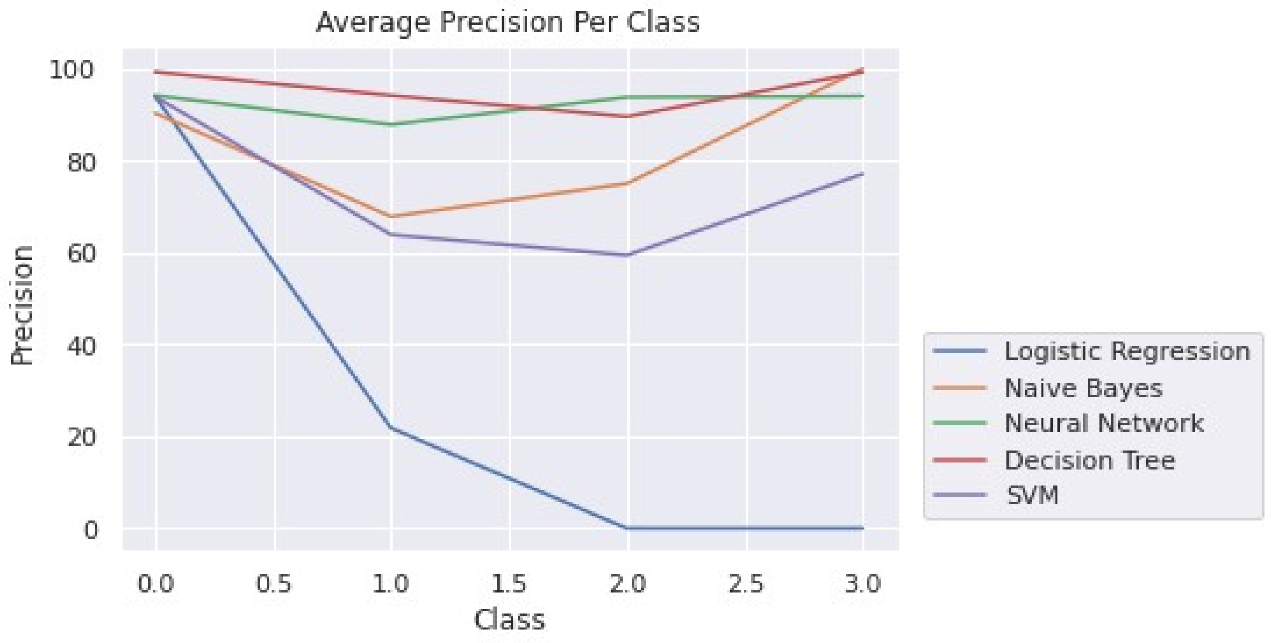


Figure 14: Average Precision Per Class

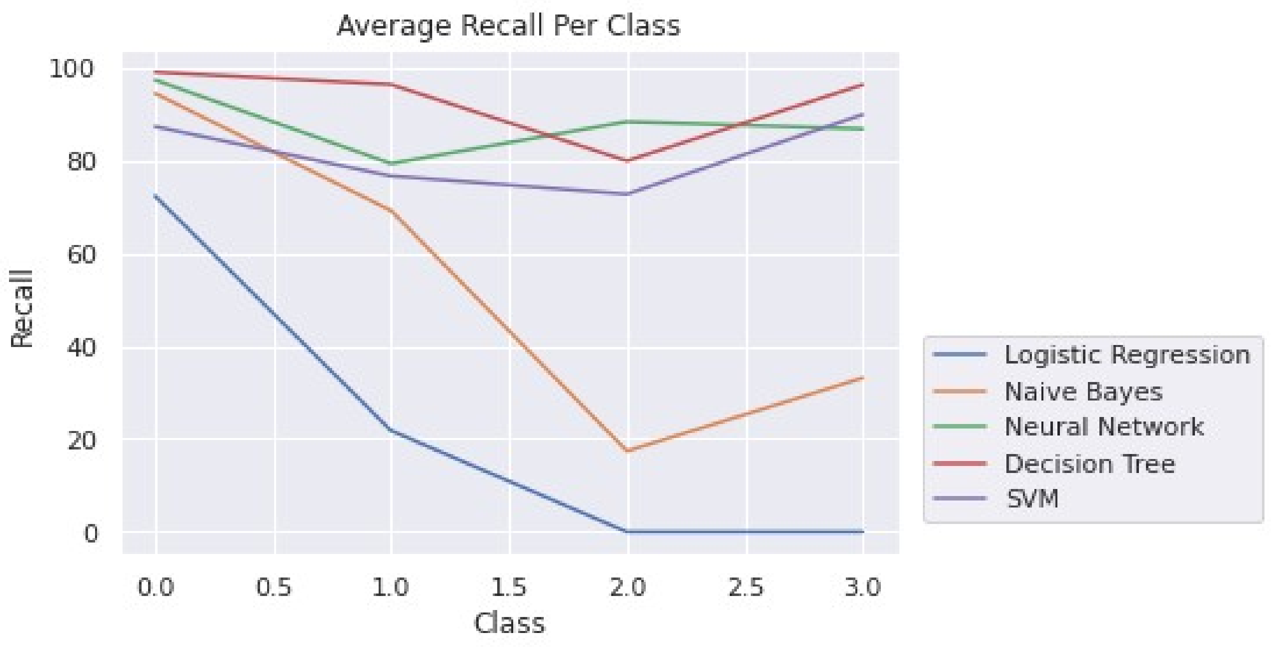


Figure 15: Average Recall Per Class

6.1 Per class

Class 0:

Beginning with Class 0, each model has a relatively high precision score for this class. The average precision score per model for Class 0 is above 90% as can be seen from the graph above. If we look at the recall per class, every model besides the Logistic Regression model has a recall of 85% and greater. This implies that for Class 0, each model has a strong accuracy in predicting the class correctly. However, considering that our dataset was skewed towards class 0, whereby class 0 accounted for 70% of the dataset, a model could solely predict class 0 and achieve a relatively acceptable accuracy and precision score for the class. This is the case for the Logistic Regression model. Even though for Class 0 it has a precision score of 94,02%, the recall for the class is 72,34% which immediately tell us that the model is predicting positively for class 0 majority of the time, however only 72.34% of those classifications are actually correct. However if we look at the recall scores for the remaining models, they are all relatively similar to their precision scores. The implication is that even though the model predicts class 0 majority of the time, a higher percentage of these predictions are actually correct. Thus they have a higher overall accuracy.

Class 1:

If we move onto class 1 and look at the precision of each model, we notice that the models with a higher average accuracy, namely the Decision Tree and Neural Network have a high precision and recall for this class as well, whereas the moderately performing models have a moderate precision and recall for class 1 and the Logistic Regression model has an average precision and recall of 21.81% for class 1. The inability of the Logistic Regression model to classify this correctly majority of the time is one of the reasons for it's overall poor performance. Since Class 1 accounted for 22.22% of the dataset, a model that is capable of capturing the trends and correctly predicting this class will automatically achieve a higher accuracy than a model that is unable to do so. If we look comparatively at the recall for the Neural Network and Decision Tree, they are 79,32% and 96,33% respectively. We immediately note that the Decision Tree is much more accurate in it's prediction of the class compared to the Neural Network, consequently it has an overall higher accuracy.

If we look at the SVM and Naive Bayes Model, they were able to achieve a moderate recall and accuracy for this class. However, their recalls are much lower than that of the Decision Tree and Neural Network, but much larger than that of the Logistic Regression model. Hence, their accuracy tends to lie within the moderate range.

<u>Class 2 and 3:</u>

If we look at Classes 2 and 3, these are the classes that accounted for 3,99% and 3,76% of the dataset respectively. The ability of a model to correctly predict these two minority classes is what separated the models with moderate accuracy from those with a high accuracy. Looking at the Decision Tree and Neural Network, even though the Neural Network had a considerably higher recall for class 2, 88,34%, whereas the Decision Tree had a recall score of 79,91%, the Decision Tree still achieved a higher overall accuracy since class 2 accounted for a small percentage of the dataset.

The SVM achieved a relatively moderate recall for both classes and consequently maintained a steady average accuracy. However, if we look at the Naive Bayes model for these classes, it achieved an average recall of 17,4% and 33.15% respectively. As a result, the accuracy of the model took a dip, even though these two classes are a minority. Lastly, if we look at the Logistic Regression model, we note that it achieved a recall and precision score of 0% for both classes. This is simply due to the fact that the model failed to even predict these classes. The model solely predicted that a data-point belonged to either one of the majority classes, that being Class 0 or 1.

<u>6.2 General Discussion:</u>

If we look at the average results, we noticed that the Logistic Regression model performed the worst.

One of the possible reasons for this is the structure of the dataset. Since we had a multi-class output, we had to undertake the one verse all approach to implement the model. However, considering that the classes 2 and 3 accounted for an extremely small percentage of the dataset, and considering the fact that the data is randomly shuffled before being split, the likelihood of having enough data-points from these two minority classes for the model to capture their underlying trends is extremely low. Consequently, the model does not capture their trends and fails to predict them.

A similar problem is encountered by the Naive Bayes model. The model predicted the majority classes extremely well since their was a larger proportion of data to learn from in comparison to these two minority classes. The SVM managed to maintain a decent accuracy in predicting these classes, whereas the Decision Tree and Neural Network did an astounding job in predicting these classes. There a variety of reasons for this. In the case of the Neural Network, we believe that it's success is due to epochs. This enabled the model to learn for longer periods of time. In the case of the Decision Tree, one possible explanation for it's brilliant performance is the structure of the dataset. The dataset follows almost a decision making process whereby each factor greatly influences the outcome. This is also another downfall for the Naive Bayes model which assumes independence amongst features even though some features within the dataset, such as the number of doors a car has and the number of people that can be seated, had a degree of dependence.

A consistent theme throughout this document is the skewness of the dataset. If the dataset is to be used in the future, one recommendation would be to definitely either generate data for the minority classes or remove data from the majority classes to ensure a more even distribution of the data. This will remove the bias from a single class and give the models a fair chance to attempt to learn each possible outcome.

From our quest we also deduce that even though models like multi-class Logistic Regression exist, we feel that we should rather tend towards a Neural Network for multi-class datasets. This is simply due to the fact that even though both models were presented the same data, the Neural Network outperformed Logistic Regression in every aspect. One reason for this is that Logistic Regression aims to fit linear decision boundaries even when the data is not linear, whereas Neural Networks are capable of finding much more complicated boundaries for multi-class datasets.

SVM's can be used as well, however SVM's tries to fit hyperplanes as decision boundaries and tries to maximise the distance between these hyperplanes. However, given their sensitivity to outliers/noise, the models performance was sub-par. Again the Neural Network performed better since the outliers did not have a major effect on the weights during gradient descent.

If complexity is a problem, then Naive Bayes and Decision Trees are viable classification models. Decision Trees performed extremely well on our dataset since the dataset was extremely categorical. Naive Bayes performed relatively well however since it assumes independence of features, it may not capture all the trends and consequently have a decrease in it’s accuracy.

Lastly we look at the time taken to train these models. The average times are captured below.

Model	Average Training Time(seconds)
Logistic Regression	0,23
Naive Bayes	0,11
Neural Network	71,78
Decision Tree	0,04
Support Vector Machine	0,05

Table 19: Average Training Time

From the table we immediately notice that the Neural Network requires the most amount of time to train. This is simply due to its complexity. The Decision Tree and SVM are trained extremely rapidly, however note that built-in libraries trained these models and as a result of the internal optimization, such speed is possible. The Naive Bayes model also trains considerably fast but this is simply because it does not require convergence to a minima and is only calculating probabilities. The opposite is said for Logistic Regression which aims to minimise the cost function during gradient descent and consequently takes a little longer to train. Another factor that influences the models that implement gradient descent is the threshold(ε). However the reported times are from the models that were trained using the optimised hyper-parameters. The reported times are as expected since Neural Networks always take the longest to train.

A sample data-point that every algorithm classified incorrectly is as follows:

Feature	Value
Purchasing Price	2(high)
Maintenance Cost	2(high)
Number of doors	0(two doors)
Number of people that can be seated	0(two people)
Boot Size	0(small)
Safety	2(high)
Outcome	2(good)

It belongs to a minority class, Class 2, as a result the Logistic Regression model completely fails to classify this class.

The only reason the other algorithms failed to classify a point like this is because it is considered noise. Naive Bayes would have not seen a training instance like this and consequently the probabilities of the likelihoods will sway its prediction to a class whereby the features take on these values. In SVM's points like these distort the positions of the hyperplanes and are also incorrectly classified since they are outliers. In Neural Networks the prediction is dependent on the weights associated with neurons, these weights in general predict an outcome based on the values of the features. Since these features are in conflict with the values of the features that trained the model, the point is classified incorrectly. In the Decision Trees, we follow the path of features and end up with an incorrect classification since this combination of features is never encountered outside this point.

In conclusion each model has it's advantages and disadvantages. The model you choose should be the best model for your respective dataset. However the structure of the dataset also plays a pivotal role in the success of any model. In our case, the best models were a Neural Network and Decision Tree whereas the worst performing model was Logistic Regression.

Citations

M. Bohanec, B.Zupan. Car Evaluation Data Set. Available at:
http://archive.ics.uci.edu/ml/datasets/Car+Evaluation
(Accessed: 12 May 2021)

D. Varghese. Comparative Study on Classic Machine learning Algorithms. Available at: **https://towardsdatascience.com/comparative-study-on-classic-machine-learning-algorithms-24f9ff6ab222** (Accessed: 13 June 2021)

CODE