CITS5504 Data Warehousing

Project Report

Varun Jain 21963986

Akhil Naseem 22237476

Semester 1 2021

Project 2

Pattern Discovery and Building Predictive Models

(Teamwork split 50% each)

Introduction

Mobile Price Classification Dataset, provided by the UWA CITS5508 Team, is set to be analysed for Pattern Discovery and Predictive Modelling. The aim of this project is to apply several different machine learning algorithms to predict whether the price of a mobile phone is high or low. The mobile_price.csv file will undergo vigorous data pre-processing and cleaning, and the original dataset may be split into separate data marts dependent on the machine learning technique applied.

Data Pre-processing

Overview of Data

In the mobile price dataset, there are 2000 records with 21 individual attributes (excluding the identifier column - id). Each attribute represents the presence/absence of a feature or the specification of a feature. For instance, the following attributes "blue" , "wifi, "ram" , "sc_h" and "sc_w" indicate whether the phone has Bluetooth, Wi-Fi, RAM in megabytes, and the height and width of the mobile phone in centimetres, respectively.

Attribute Types

The type of each attribute in the dataset is defined in the table below:

Attribute	Type	Attribute	Type	Attribute	Type	Attribute	Type
Id	Nominal	**four_g**	Binary	**px_height**	Numeric	**three_g**	Binary
Battery_power	Numeric	**int_memory**	Numeric	**px_width**	Numeric	**touch_screen**	Binary
blue	Binary	**m_dep**	Numeric	**ram**	Numeric	**wifi**	Binary
clock_speed	Numeric	**mobile_wt**	Numeric	**sc_h**	Numeric	**price_category**	Binary
dual_sim	Binary	**n_cores**	Ordinal	**sc_w**	Numeric		
fc	Numeric	**pc**	Numeric	**talk_time**	Numeric		

Data Cleaning

Mobile Price Classification Dataset is a relatively clean dataset. From our analysis, the data contains no missing values, in other words, each attribute contains 2000 non-null values.

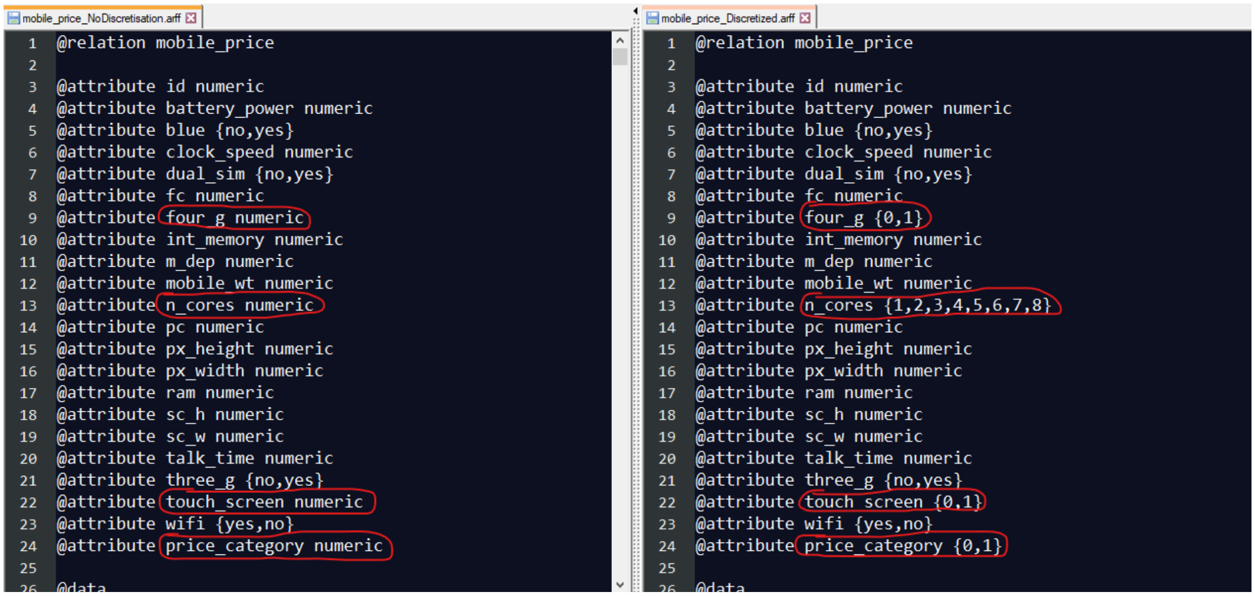
Naming conventions

The attributes **blue**, **dual_sim**, **three_g**, and **wifi** map to binary outputs, yes or no. Within these specific attributes, we noticed inconsistent input conventions. For instance, **blue** was encoded with the following values: ['Yes', 'YES', 'has','Has','yes'] for the binary output yes, and ['NO', 'not', 'Not','No','no'] for the binary output no. To fix this issue, we replaced all the values related to 'yes' to 1, and 'no' to 0. This method was applied to all the attribute values in the columns mentioned above. This can be seen in the code below.

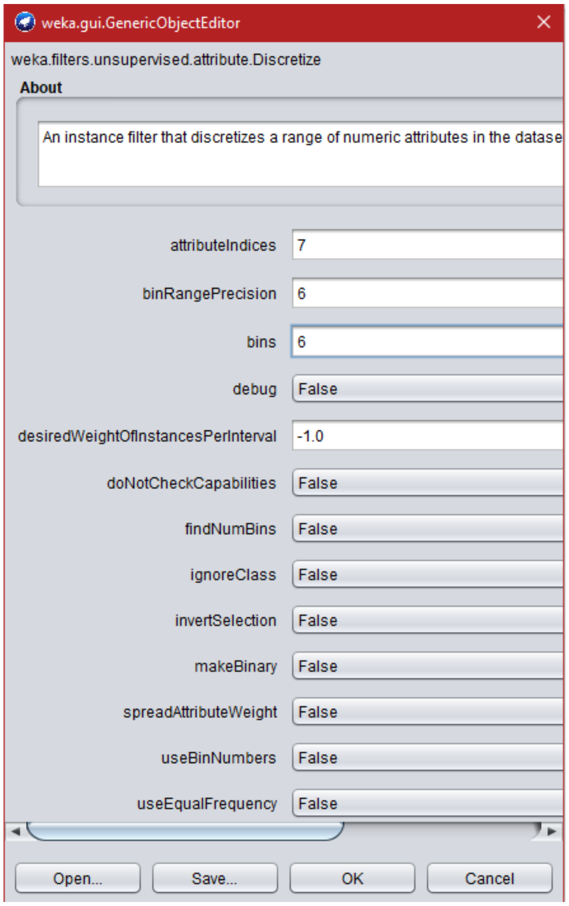
```
def rename_binary_attributes(data, attribute):
    for index in attribute:
        data[attribute] = data[attribute].replace(['Yes', 'YES', 'has','Has','yes'],'yes')
        data[attribute] = data[attribute].replace(['NO', 'not', 'Not','No','no'],'no')
    return data
mobile_data = rename_binary_attributes(mobile_data, ['blue','dual_sim','three_g','wifi'])
```

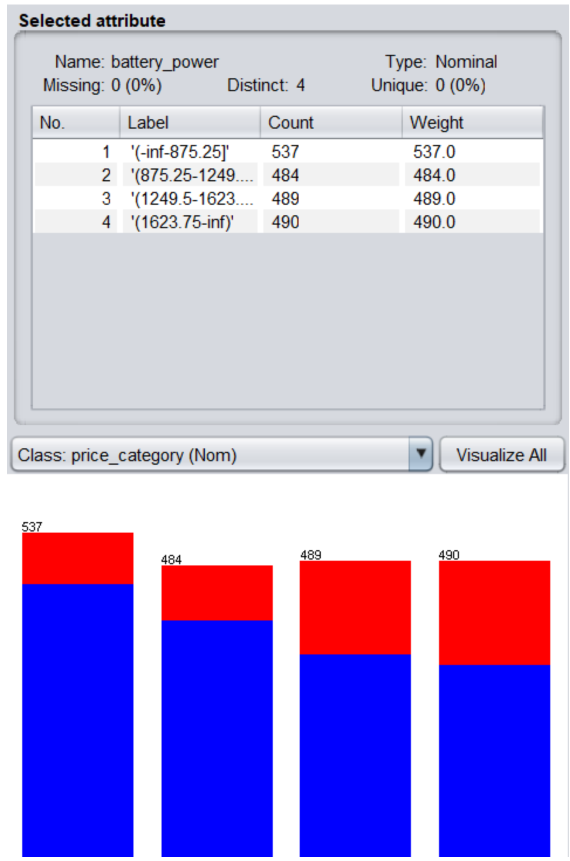
Discretization

The attributes **four_g**, **touch_screen, n_cores** and **price_category** are all categorical labels, but Weka reads them as numerical. Association mining can only be performed on categorical data. So to allow Weka to differentiate between labels, it is more convenient if the categorical data are labelled. Otherwise, upon analysis on Weka, it will return the min, max, mean, standard deviation, instead of the frequency. So, to discretize these attributes, we removed the 'numeric' keyword from the saved arff file and added lists of correct labels (see image below).

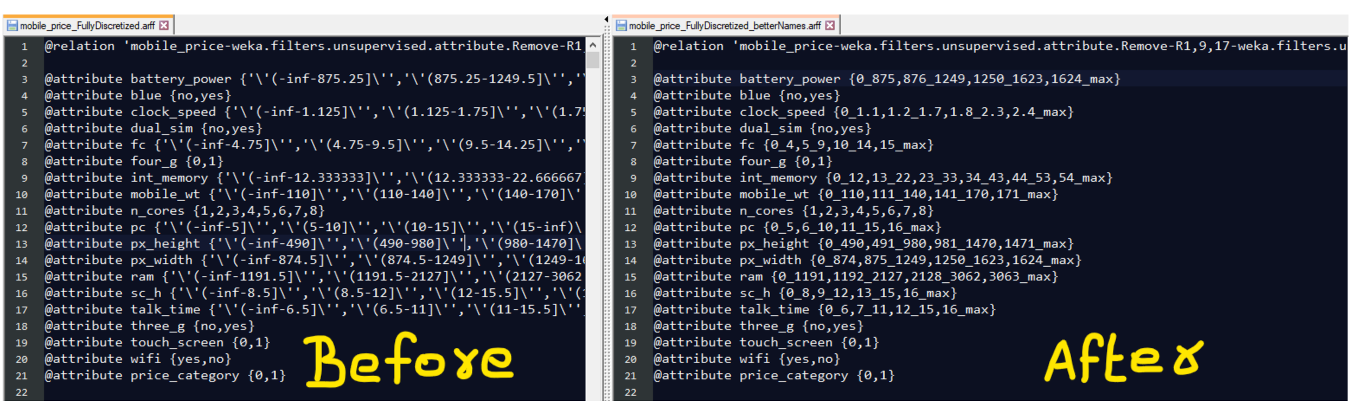


As for the numeric data, we need to discretize them by classifying them into value ranges. This is done with the built-in Weka filter for discretization (see image below). The filter is applied on the **battery_power**, **clock_speed**, **fc**, **mobile_wt**, **pc**, **px_height**, **px_width**, **ram**, **sc_h**, and **talk_time** attributes to convert them from numeric to 4-bin categorical. We also convert **int_memory** attribute to a 6-bin categorical attribute. These bins have more or less an equal-width distribution. The output of the filter on **battery_power** is shown below.





Weka's naming convention for discretized attributes is hard to interpret. Therefore, we have manually done a find/replace for each of these attributes with easier to read attribute values.



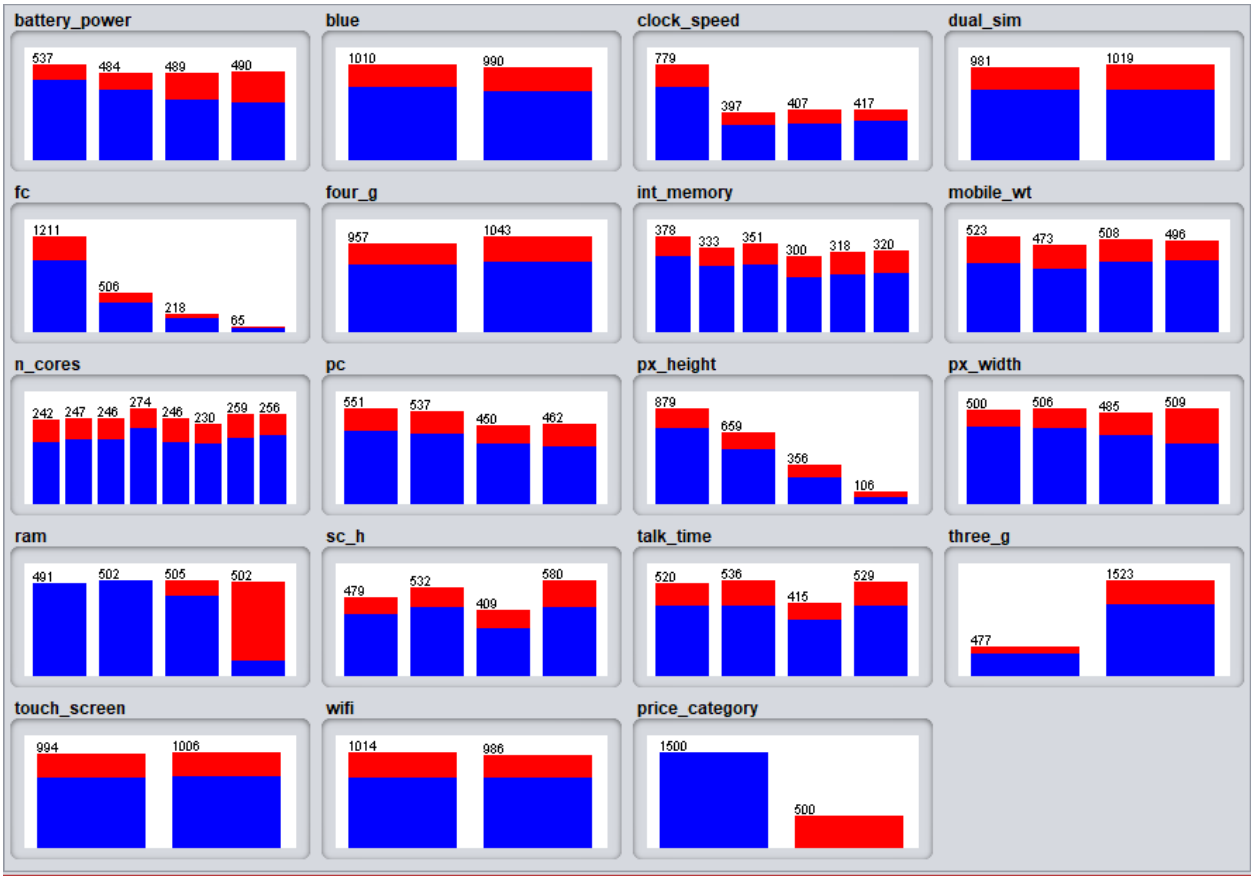
Removing unnecessary attributes

From observing the values in the dataset, the following attributes will be removed.

- **id**: unique number assigned to each instance in the dataset. However, there is no correlation between the id attribute and the target attribute, price_category. So, when passing this attribute over to machine learning algorithms, it will not contribute to the output.
- **m_dep**: this attribute has non-sensible values. It seems unreasonable that phones are less than 0.5cm thick. The meaning of this attribute is unclear and therefore will be removed.
- **sc_w**: this attribute too has non-sensible values because we cannot have a screen width of 0cm, which there are a lot of records of. Therefore this attribute will be removed because its meaning is not clear.

Final attributes

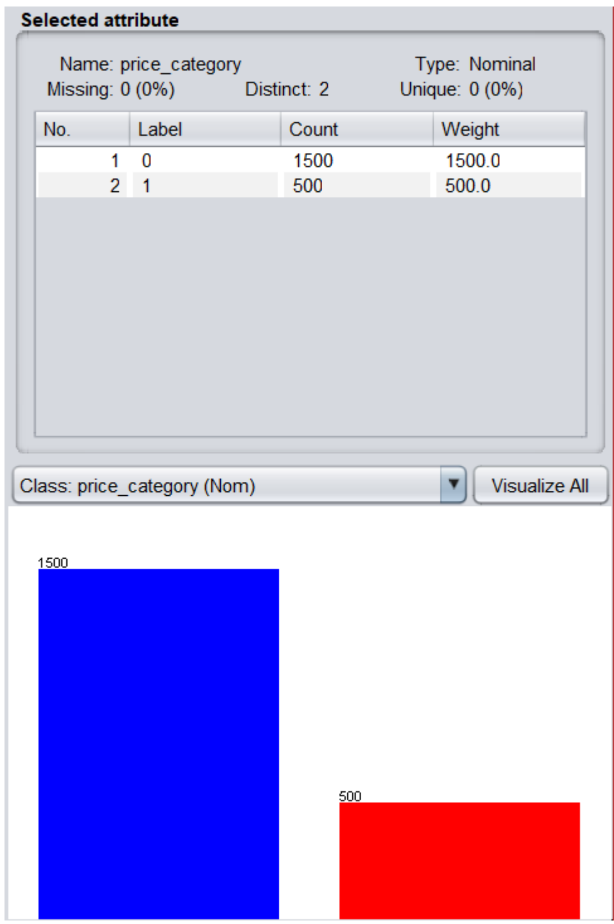
The image below visualises all the final attributes, their distributions, and the representation of the target class (price_category) with the red and blue colours.The red defining the high price category, and the blue being the low class category.



battery_power
537
484
489
490
blue
1010
990
clock_speed
779
397
407
417
dual_sim
981
1019
fc
1211
506
218
65
four_g
957
1043
int_memory
378
333
351
300
318
320
mobile_wt
523
473
508
496
n_cores
242
247
246
274
246
230
259
256
pc
551
537
450
462
px_height
879
659
356
106
px_width
500
506
485
509
ram
491
502
505
502
sc_h
479
532
409
580
talk_time
520
536
415
529
three_g
477
1523
touch_screen
994
1006
wifi
1014
986
price_category
1500
500

Association Rule Mining

Association Rule Mining is a category of unsupervised learning, allowing patterns in data to be found without a target variable. It is associated with finding frequent item sets. Association rules are used to find interesting if-then patterns using metrics like Support, Confidence and Lift. In the context of the mobile price dataset, we want to find attribute sets that have high correlation to the price category. It is important to note the data imbalance in the target attribute **price_category**. There are 3 times more low priced mobile phones than high priced ones.

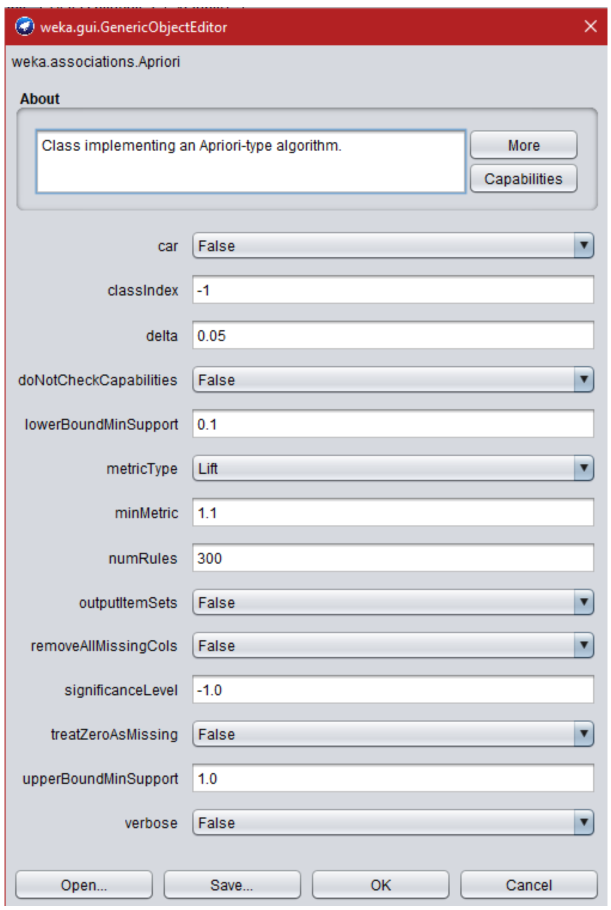


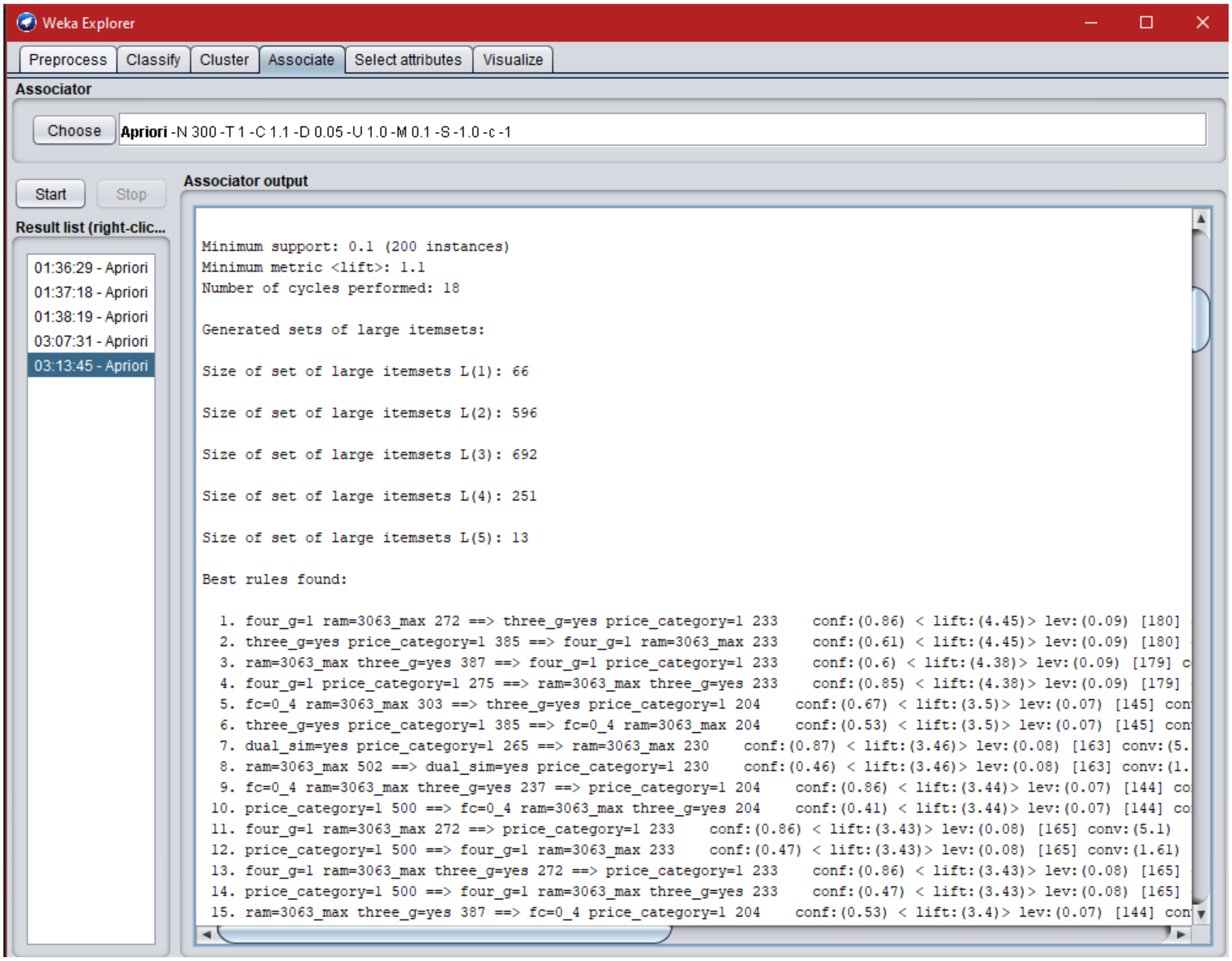
Apriori Algorithm

We used Weka's built-in Apriori algorithm miner for this exercise. Initially, we used confidence and support as the primary metrics to evaluate the quality of the rules generated by the apriori model. We found that confidence was not a good measure for model evaluation as the majority of the associations belonged to either the price_category=low or three_g=yes, as observed in the figure below. This was recurrent for all values of minMetric. Ordering by confidence has the downside that not all rules with high confidence are interesting or helpful. This is because the confidence is calculated off solely based on frequency of the right hand side. And since there is a significantly high sample frequency in **price_category** and **three_g**, we see most top rules having a confidence of 1. The figure below only displays 30 rules; however, it was tested for different sample sizes and resulted similarly.

```
Best rules found:

 1. four_g=1 1043 ==> three_g=yes 1043    <conf:(1)> lift:(1.31) lev:(0.12) [248] conv:(248.76)
 2. four_g=1 price_category=0 768 ==> three_g=yes 768    <conf:(1)> lift:(1.31) lev:(0.09) [183] conv:(183.17)
 3. fc=0_4 four_g=1 634 ==> three_g=yes 634    <conf:(1)> lift:(1.31) lev:(0.08) [151] conv:(151.21)
 4. pc=0_5 551 ==> fc=0_4 551    <conf:(1)> lift:(1.65) lev:(0.11) [217] conv:(217.37)
 5. dual_sim=yes four_g=1 533 ==> three_g=yes 533    <conf:(1)> lift:(1.31) lev:(0.06) [127] conv:(127.12)
 6. four_g=1 touch_screen=1 533 ==> three_g=yes 533    <conf:(1)> lift:(1.31) lev:(0.06) [127] conv:(127.12)
 7. blue=yes four_g=1 523 ==> three_g=yes 523    <conf:(1)> lift:(1.31) lev:(0.06) [124] conv:(124.74)
 8. four_g=1 wifi=no 523 ==> three_g=yes 523    <conf:(1)> lift:(1.31) lev:(0.06) [124] conv:(124.74)
 9. blue=no four_g=1 520 ==> three_g=yes 520    <conf:(1)> lift:(1.31) lev:(0.06) [124] conv:(124.02)
10. four_g=1 wifi=yes 520 ==> three_g=yes 520    <conf:(1)> lift:(1.31) lev:(0.06) [124] conv:(124.02)
11. dual_sim=no four_g=1 510 ==> three_g=yes 510    <conf:(1)> lift:(1.31) lev:(0.06) [121] conv:(121.64)
12. four_g=1 touch_screen=0 510 ==> three_g=yes 510    <conf:(1)> lift:(1.31) lev:(0.06) [121] conv:(121.64)
13. ram=1192_2127 502 ==> price_category=0 502    <conf:(1)> lift:(1.33) lev:(0.06) [125] conv:(125.5)
14. ram=0_1191 491 ==> price_category=0 491    <conf:(1)> lift:(1.33) lev:(0.06) [122] conv:(122.75)
15. three_g=no 477 ==> four_g=0 477    <conf:(1)> lift:(2.09) lev:(0.12) [248] conv:(248.76)
16. four_g=1 px_height=0_490 468 ==> three_g=yes 468    <conf:(1)> lift:(1.31) lev:(0.06) [111] conv:(111.62)
17. fc=0_4 four_g=1 price_category=0 461 ==> three_g=yes 461    <conf:(1)> lift:(1.31) lev:(0.05) [109] conv:(109.9
18. clock_speed=0_1.1 four_g=1 427 ==> three_g=yes 427    <conf:(1)> lift:(1.31) lev:(0.05) [101] conv:(101.84)
19. pc=0_5 three_g=yes 424 ==> fc=0_4 424    <conf:(1)> lift:(1.65) lev:(0.08) [167] conv:(167.27)
20. pc=0_5 price_category=0 419 ==> fc=0_4 419    <conf:(1)> lift:(1.65) lev:(0.08) [165] conv:(165.3)
21. four_g=1 wifi=no price_category=0 394 ==> three_g=yes 394    <conf:(1)> lift:(1.31) lev:(0.05) [93] conv:(93.97
22. blue=no four_g=1 price_category=0 393 ==> three_g=yes 393    <conf:(1)> lift:(1.31) lev:(0.05) [93] conv:(93.73
23. dual_sim=yes four_g=1 price_category=0 389 ==> three_g=yes 389    <conf:(1)> lift:(1.31) lev:(0.05) [92] conv:(
24. four_g=1 touch_screen=1 price_category=0 388 ==> three_g=yes 388    <conf:(1)> lift:(1.31) lev:(0.05) [92] conv
25. ram=1192_2127 three_g=yes 383 ==> price_category=0 383    <conf:(1)> lift:(1.33) lev:(0.05) [95] conv:(95.75)
26. four_g=1 touch_screen=0 price_category=0 380 ==> three_g=yes 380    <conf:(1)> lift:(1.31) lev:(0.05) [90] conv
27. dual_sim=no four_g=1 price_category=0 379 ==> three_g=yes 379    <conf:(1)> lift:(1.31) lev:(0.05) [90] conv:(9
28. blue=yes four_g=1 price_category=0 375 ==> three_g=yes 375    <conf:(1)> lift:(1.31) lev:(0.04) [89] conv:(89.4
29. four_g=1 wifi=yes price_category=0 374 ==> three_g=yes 374    <conf:(1)> lift:(1.31) lev:(0.04) [89] conv:(89.2
30. ram=0_1191 three_g=yes 366 ==> price_category=0 366    <conf:(1)> lift:(1.33) lev:(0.05) [91] conv:(91.5)
31. four_g=1 px_height=0_490 price_category=0 363 ==> three_g=yes 363    <conf:(1)> lift:(1.31) lev:(0.04) [86] con
32. three_g=no price_category=0 362 ==> four_g=0 362    <conf:(1)> lift:(2.09) lev:(0.09) [188] conv:(188.78)
```

Therefore, instead of using confidence, we use Lift and Support as our primary metrics (see below for selected hyperparameters). Lift measures the importance of a rule. A lift value greater than 1 indicates that occurrence of item A has a positive association with item B, lift value less than 1 indicates that occurrence of item A has a negative association with item B and a lift value of 0, means there is no association between the two attributes. Therefore lift ranks according to the dependence of the left side and the right side. This in turn ends up giving us a more accurate depiction if one variable is truly dependent on the other or if it is just due to frequency of that rule in the dataset. The images below show the settings used for the Apriori algorithm using the Lift metric type and the output.




Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Associator
Choose
Apriori -N 300 -T 1 -C 1.1 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1
Start
Stop
Result list (right-clic...
01:36:29 - Apriori
01:37:18 - Apriori
01:38:19 - Apriori
03:07:31 - Apriori
03:13:45 - Apriori
Associator output
Minimum support: 0.1 (200 instances)
Minimum metric <lift>: 1.1
Number of cycles performed: 18
Generated sets of large itemsets:
Size of set of large itemsets L(1): 66
Size of set of large itemsets L(2): 596
Size of set of large itemsets L(3): 692
Size of set of large itemsets L(4): 251
Size of set of large itemsets L(5): 13
Best rules found:
1. four_g=1 ram=3063_max 272 ==> three_g=yes price_category=1 233 conf:(0.86) < lift:(4.45)> lev:(0.09) [180]
2. three_g=yes price_category=1 385 ==> four_g=1 ram=3063_max 233 conf:(0.61) < lift:(4.45)> lev:(0.09) [180]
3. ram=3063_max three_g=yes 387 ==> four_g=1 price_category=1 233 conf:(0.6) < lift:(4.38)> lev:(0.09) [179] c
4. four_g=1 price_category=1 275 ==> ram=3063_max three_g=yes 233 conf:(0.85) < lift:(4.38)> lev:(0.09) [179]
5. fc=0_4 ram=3063_max 303 ==> three_g=yes price_category=1 204 conf:(0.67) < lift:(3.5)> lev:(0.07) [145] con
6. three_g=yes price_category=1 385 ==> fc=0_4 ram=3063_max 204 conf:(0.53) < lift:(3.5)> lev:(0.07) [145] con
7. dual_sim=yes price_category=1 265 ==> ram=3063_max 230 conf:(0.87) < lift:(3.46)> lev:(0.08) [163] conv:(5.
8. ram=3063_max 502 ==> dual_sim=yes price_category=1 230 conf:(0.46) < lift:(3.46)> lev:(0.08) [163] conv:(1.
9. fc=0_4 ram=3063_max three_g=yes 237 ==> price_category=1 204 conf:(0.86) < lift:(3.44)> lev:(0.07) [144] co
10. price_category=1 500 ==> fc=0_4 ram=3063_max three_g=yes 204 conf:(0.41) < lift:(3.44)> lev:(0.07) [144] co
11. four_g=1 ram=3063_max 272 ==> price_category=1 233 conf:(0.86) < lift:(3.43)> lev:(0.08) [165] conv:(5.1)
12. price_category=1 500 ==> four_g=1 ram=3063_max 233 conf:(0.47) < lift:(3.43)> lev:(0.08) [165] conv:(1.61)
13. four_g=1 ram=3063_max three_g=yes 272 ==> price_category=1 233 conf:(0.86) < lift:(3.43)> lev:(0.08) [165]
14. price_category=1 500 ==> four_g=1 ram=3063_max three_g=yes 233 conf:(0.47) < lift:(3.43)> lev:(0.08) [165]
15. ram=3063_max three_g=yes 387 ==> fc=0_4 price_category=1 204 conf:(0.53) < lift:(3.4)> lev:(0.07) [144] con

Top 10 association rules

	Rule	conf	lift
1	**fc**=0_4 **ram**=3063_max **three_g**=yes 237 ==> **price_category**=1 204	0.86	3.44
2	**four_g**=1 **ram**=3063_max 272 ==> **price_category**=1 233	0.86	3.43
3	**four_g**=1 **ram**=3063_max **three_g**=yes 272 ==> **price_category**=1 233	0.86	3.43
4	**fc**=0_4 **ram**=3063_max 303 ==> **price_category**=1 257	0.85	3.39
5	**ram**=3063_max **touch_screen**=1 243 ==> **price_category**=1 206	0.85	3.39
6	**ram**=3063_max **three_g**=yes 387 ==> **price_category**=1 327	0.84	3.38
7	**ram**=3063_max **wifi**=yes 257 ==> **price_category**=1 216	0.84	3.36
8	**blue**=yes **ram**=3063_max 260 ==> **price_category**=1 218	0.84	3.35
9	**ram**=3063_max 502 ==> **price_category**=1 418	0.83	3.33
10	**blue**=no **ram**=3063_max 242 ==> **price_category**=1 200	0.83	3.31

Top 10 association rules comprehension

1) If the mobile phone has a front camera between 0 and 4 pixels, with ram greater than 3063MB and has three_g, then it is 3.39 times more likely for the mobile phone to be high priced.

2) If the mobile phone has four_g and has ram greater than 3063MB, then it is 3.43 times more likely for the mobile to be high priced.

3) If the mobile phone has four_g, three_g and ram greater than 3063MB, then it is 3.43 times more likely for the mobile to be high priced.

4) If the mobile phone has a front camera pixel between 0 and 4 and ram greater than 3063MB, then it is 3.39 times more likely for the mobile to be high priced.

5) If the mobile phone has ram greater than 3063 MB and is touch_screen, then it is 3.39 times more likely for the mobile to be high priced.

6) If the mobile phone has ram greater than 3063 MB and has three_g, then it is 3.38 times more likely for the mobile to be high priced.

7) If the mobile phone has ram greater than 3063 MB and has wifi, then it is 3.36 times more likely for the mobile to be high priced.

8) If the mobile phone has bluetooth and ram greater than 3063 MB, then it is 3.35 times more likely for the mobile to be high priced.

9) If the ram is greater than 3063 MB, then it is 3.33 times more likely for the mobile to be high priced.

10) If the phone does not have bluetooth but has a ram greater than 3063 MB, then it is 3.31 times more likely for the mobile to be high priced.

Recommended features for a high price mobile phone (To Do)

- Noticeable component observed from the rules above is RAM. RAM has a significant influence over the price of the mobile phone as all 10 of the rules presented above, contains high memory RAM with more than 3063MB. Regardless of whether the phone is equipped with bluetooth, we recommend that to build a high priced mobile phone, it is important to have a relatively larger amount of RAM.

- For high priced mobile phone, the device should be equipped with the following features:
  - It should be touchscreen
  - Must have front camera pixels
  - Must be compatible with both three_g and four_g

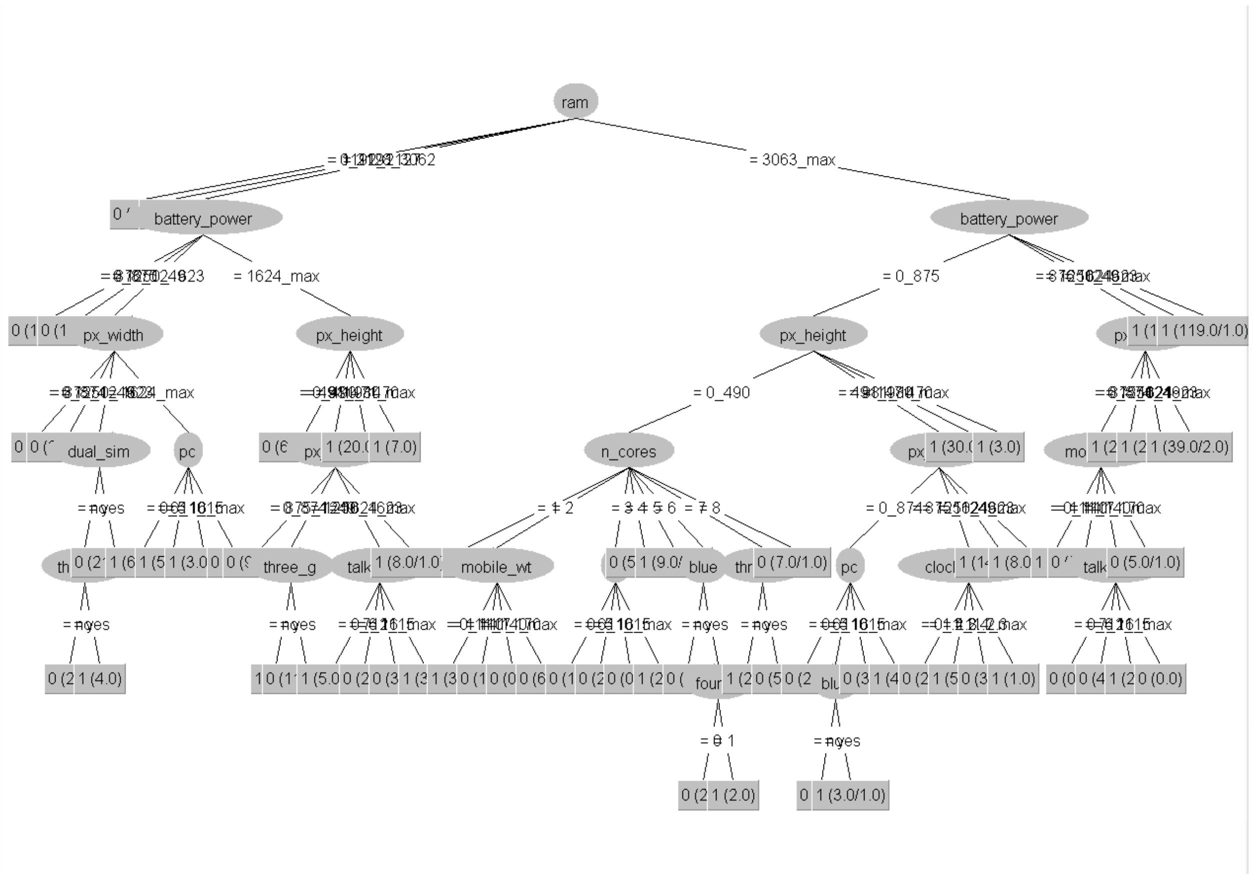
Classification

Decision Trees and Support Vector Classifier (SVM) are two supervised models that will be used for data analysis on the price mobile dataset. Decision Tree models output a tree-like structure with different branches and nodes, representing the statistical probability of a decision or an outcome. These models are versatile as they can perform analysis on both a regression and classification dataset, to predict continuous or categorical data, respectively. Support Vector Machines finds the hyperplane (a.k.a decision boundary) that separates the two features. In this model, the SVM will predict which category the mobile phone price will belong to. The best decision boundary is one that maximises the margins from both features.

Decision Tree

The decision tree was created using the 10 fold cross-validation method on the J48 model.

```
Number of Leaves  :     66

Size of the tree :      91


Time taken to build model: 0.08 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        1846               92.3    %
Incorrectly Classified Instances       154                7.7    %
Kappa statistic                          0.7927
Mean absolute error                      0.0937
Root mean squared error                  0.2476
Relative absolute error                 24.9774 %
Root relative squared error             57.1743 %
Total Number of Instances             2000

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.953    0.168    0.945      0.953    0.949      0.793    0.955     0.981     0
                 0.832    0.047    0.856      0.832    0.844      0.793    0.955     0.836     1
Weighted Avg.    0.923    0.138    0.922      0.923    0.923      0.793    0.955     0.945

=== Confusion Matrix ===

    a    b   <-- classified as
 1430   70 |    a = 0
   84  416 |    b = 1
```



Using our fully discretized data, we get a very big decision tree with 66 leaves and 91 nodes. The overall accuracy of correctly classified instances is 92.3% . The precision and recall are both greater than 94.5% for low priced mobile phones, which is very good and much better than random guess (50%). However, classifying high priced mobile phones scored much lower at 85.6% and 83.2% in precision and recall. This is probably due to the class imbalance in the dataset. Visualising the tree does show that it is very cluttered and could use some data reduction to see if we can get similar results (or better) with lower complexity. More on this after the data reduction section below.

Support Vector Machine

The SVM was created using the 10 fold cross-validation method on the SMO model.

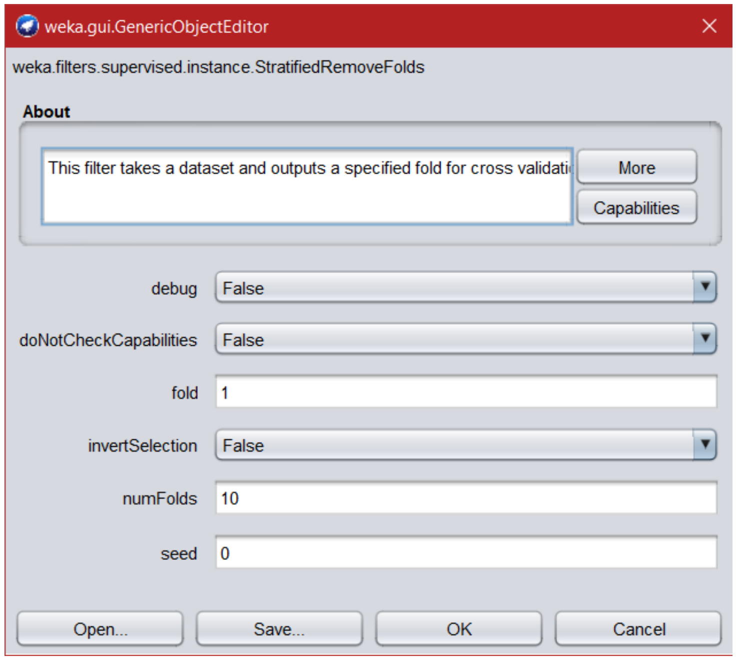
```
Number of kernel evaluations: 349881 (82.676% cached)



Time taken to build model: 0.36 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        1963               98.15   %
Incorrectly Classified Instances        37                1.85   %
Kappa statistic                          0.951
Mean absolute error                      0.0185
Root mean squared error                  0.136
Relative absolute error                  4.9315 %
Root relative squared error             31.4112 %
Total Number of Instances             2000

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.985    0.028    0.991      0.985    0.988      0.951    0.978     0.987     0
                 0.972    0.015    0.955      0.972    0.963      0.951    0.978     0.935     1
Weighted Avg.    0.982    0.025    0.982      0.982    0.982      0.951    0.978     0.974

=== Confusion Matrix ===

    a    b   <-- classified as
 1477   23 |    a = 0
   14  486 |    b = 1
```

Using the fully discretized data, our SVM model gives us a very high accuracy for correctly labelled classes at 98.15% . The precision and recall were very good at over 99.1% and 98.5% for classifying low priced mobile phones, and 95.5% and 97.2% for high priced mobile phones.

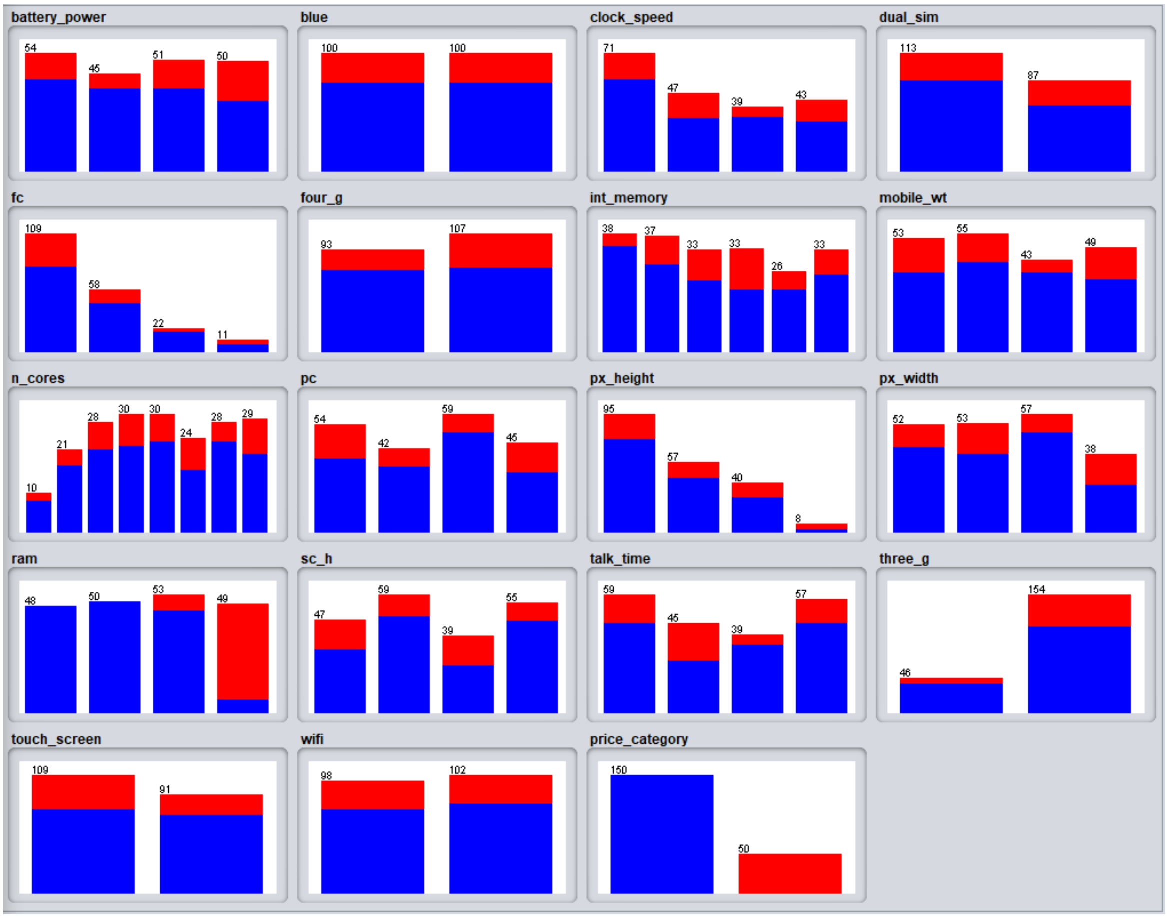
Data Reduction

Numerosity Reduction

Numerosity reduction is a technique which replaces original data volume by smaller forms of data representation. It is useful to increase performance in the analysis of a data set and also provide an easier and quicker way to test attributes of data. A specific form of numerosity reduction is sampling, in which a large data set is represented by a smaller random data sample. In Weka, different forms of sampling exist. We will use the filter StratifiedRemoveFolds using 10 folds. We keep the seed option to default value 0 as we do not want the order of the instances to be randomized. A picture of the parameters for this filter is shown below:



Using this filter we are able to sample the dataset to a much smaller subset as shown below. All attributes now have randomized subsets of the original data.



Feature Reduction

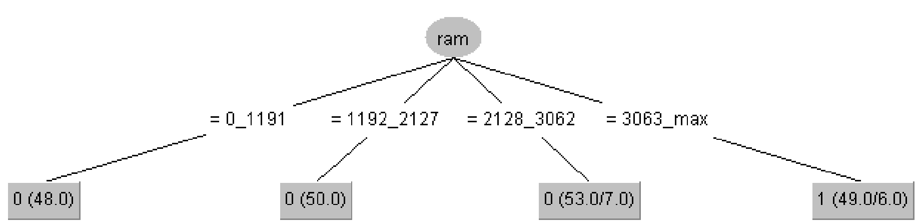
In addition to this we apply dimensionality reduction (or feature reduction) to the data set in order to reduce the number of features and improve the performance even further. If two attributes with different numbers of possible values (categories) have the same Entropy, Information Gain cannot differentiate them (decision tree algorithm will select one of them randomly). In the same situation Gain Ratio, will favor attributes with less categories. Hence, the Gain Ratio strategy leads to better generalization (less overfitting) of decision tree models. We applied Gain Ratio evaluation using Weka's built-in filter GainRatioAttributeEval and then reduced the feature space to the top 10 attributes (about half the number of attributes in the original dataset).

```
=== Attribute Selection on all input data ===

Search Method:
	Attribute ranking.

Attribute Evaluator (supervised, Class (nominal): 19 price_category):
	Gain Ratio feature evaluator

Ranked attributes:
 0.2654568  13 ram
 0.0190314  15 talk_time
 0.0151302  14 sc_h
 0.0139529  11 px_height
 0.0134748  12 px_width
 0.0124999   7 int_memory
 0.0114976  10 pc
 0.010083    1 battery_power
 0.009055   16 three_g
 0.0085773   8 mobile_wt
 0.0075666   3 clock_speed
 0.0070732   6 four_g
 0.0068575   5 fc
 0.0055369  17 touch_screen
 0.0034805   9 n_cores
 0.0019973   4 dual_sim
 0.0000962  18 wifi
-0           2 blue

Selected attributes: 13,15,14,11,12,7,10,1,16,8,3,6,5,17,9,4,18,2 : 18
```

Using Reduced Data in Decision Tree

We will now use the reduced data to train a decision tree to see if it improved the output and/or made the model less complex. For the test set, we will use the data without numerosity reduction (because the fully reduced data is small, and therefore the results will suffer from overfitting, even with cross-validation; plus, why waste available data!).

```
Number of Leaves  :     4

Size of the tree :      5


Time taken to build model: 0.03 seconds

=== Evaluation on test set ===

Time taken to test model on supplied test set: 0.14 seconds

=== Summary ===

Correctly Classified Instances        1834               91.7    %
Incorrectly Classified Instances       166                8.3    %
Kappa statistic                          0.779
Mean absolute error                      0.126
Root mean squared error                  0.2647
Relative absolute error                 33.4809 %
Root relative squared error             61.1228 %
Total Number of Instances             2000

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.944    0.164    0.945      0.944    0.945      0.779    0.944     0.971     0
                 0.836    0.056    0.833      0.836    0.834      0.779    0.944     0.778     1
Weighted Avg.    0.917    0.137    0.917      0.917    0.917      0.779    0.944     0.922

=== Confusion Matrix ===

    a    b   <-- classified as
 1416   84 |    a = 0
   82  418 |    b = 1
```



The reduced feature space still worked well for the decision tree. It performed similarly as the previous tree with comparable values for precision and recall for low priced mobile phones. Even though it did not do as well in precision for the high price mobile phones are the previous tree, the difference is not only 2% . Furthermore, the tree only has 5 nodes (i.e. 1 attribute). This is interesting, but not surprising, as we saw a similar case in association mining where every rule had **ram** in its itemset. Therefore, we were able to achieve similar results with much less complexity.

Using Reduced Data in SVM

We will now use the reduced data to train an SVM to see if it improved the output and/or made the model less complex. Again, for the test set, we will use the data without numerosity reduction.

```
Number of kernel evaluations: 11493 (90.207% cached)



Time taken to build model: 0.1 seconds

=== Evaluation on test set ===

Time taken to test model on supplied test set: 0.06 seconds

=== Summary ===

Correctly Classified Instances        1860               93      %
Incorrectly Classified Instances       140                7      %
Kappa statistic                          0.8126
Mean absolute error                      0.07
Root mean squared error                  0.2646
Relative absolute error                 18.6053 %
Root relative squared error             61.1    %
Total Number of Instances             2000

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.955    0.146    0.952      0.955    0.953      0.813    0.905     0.943     0
                 0.854    0.045    0.864      0.854    0.859      0.813    0.905     0.775     1
Weighted Avg.    0.930    0.121    0.930      0.930    0.930      0.813    0.905     0.901

=== Confusion Matrix ===

    a    b   <-- classified as
 1433   67 |    a = 0
   73  427 |    b = 1
```

Unlike the decision tree, the reduced data did not fare as well as without reduction. We can see that accuracy dropped down to 93% from the 98% from before. Precision and recall fell by more than 10% . This is due to the high false positive rate for high priced mobile phones. Therefore, in this context, the SVM would like the full original data as input for better results.

Model Evaluations

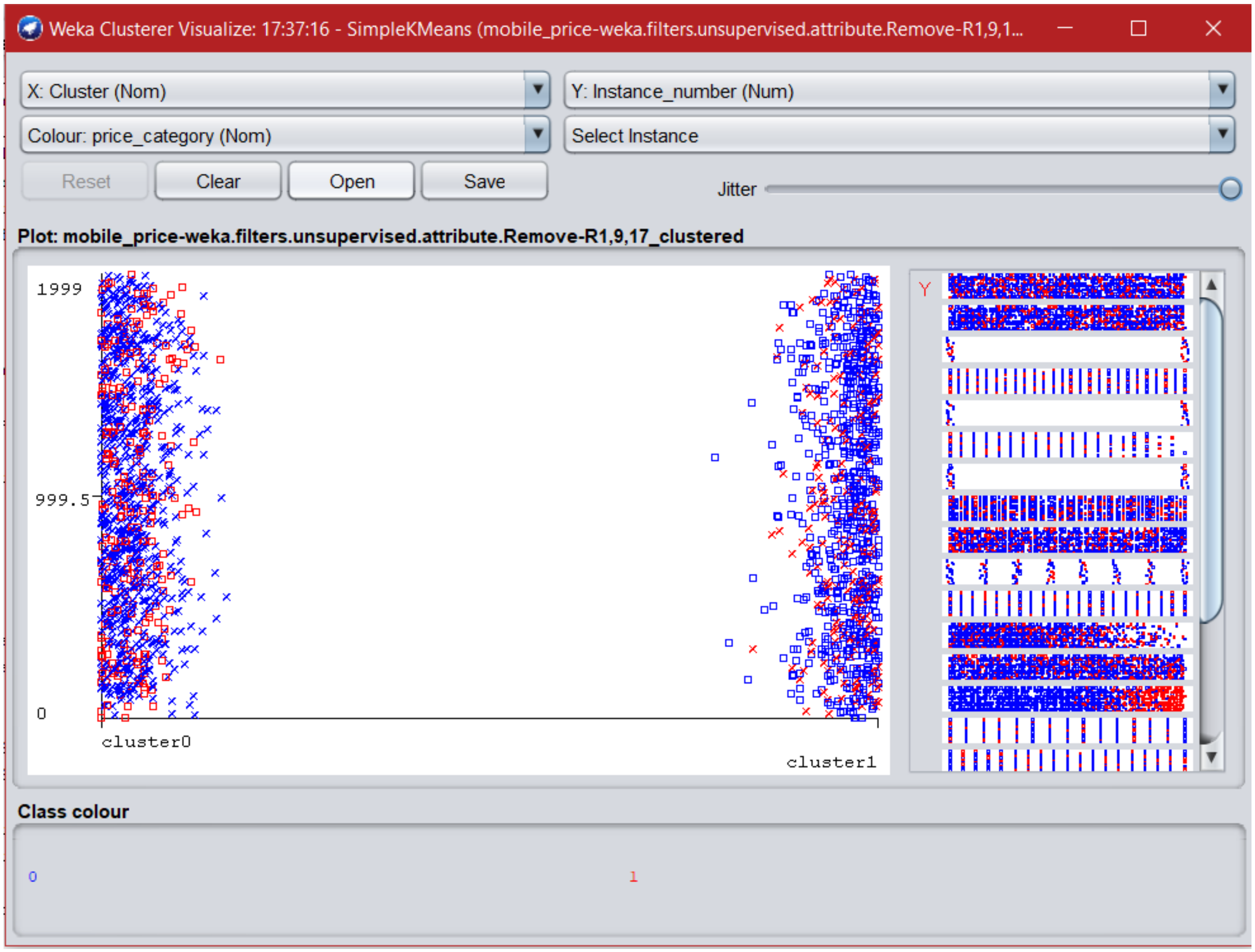
Model	Trained with	Tested with	Accuracy	Precision		Recall	
				Low price	High price	Low price	High price
Decision Tree	Original data	10 cross-val	92.3%	94.5%	85.6%	95.3%	83.2%
Decision Tree	Reduced data	Original data	91.7%	94.5%	83.3%	94.4%	83.6%
SVM	Original data	10 cross-val	98.2%	99.1%	95.5%	98.5%	97.2%
SVM	Reduced data	Original data	93%	95.2%	86.4%	85.4%	93.0%

The above table summarises the model performances of decision trees and SVMs against original and reduced training datasets. For the decision trees, even though the performance decreased across all metrics, it is not too dissimilar (RMSE increased from 24% to 26%). The data reduction did help to simplify the tree model with just a single attribute. Therefore, in our opinion, the quality of the tree did improve because it is simpler whilst still providing similar results. As for the SVM, the reduction in data did not improve performance. Precision for high price mobile phones fell by 9% and recall by 4% . Overall accuracy also fell by 5% . The RMSE increased from 13.6% to 26.4% (meaning that the line of best fit for the predicted values has a lower concentration of actual values around it). Therefore, we can say that data reduction did not improve the quality of the model.

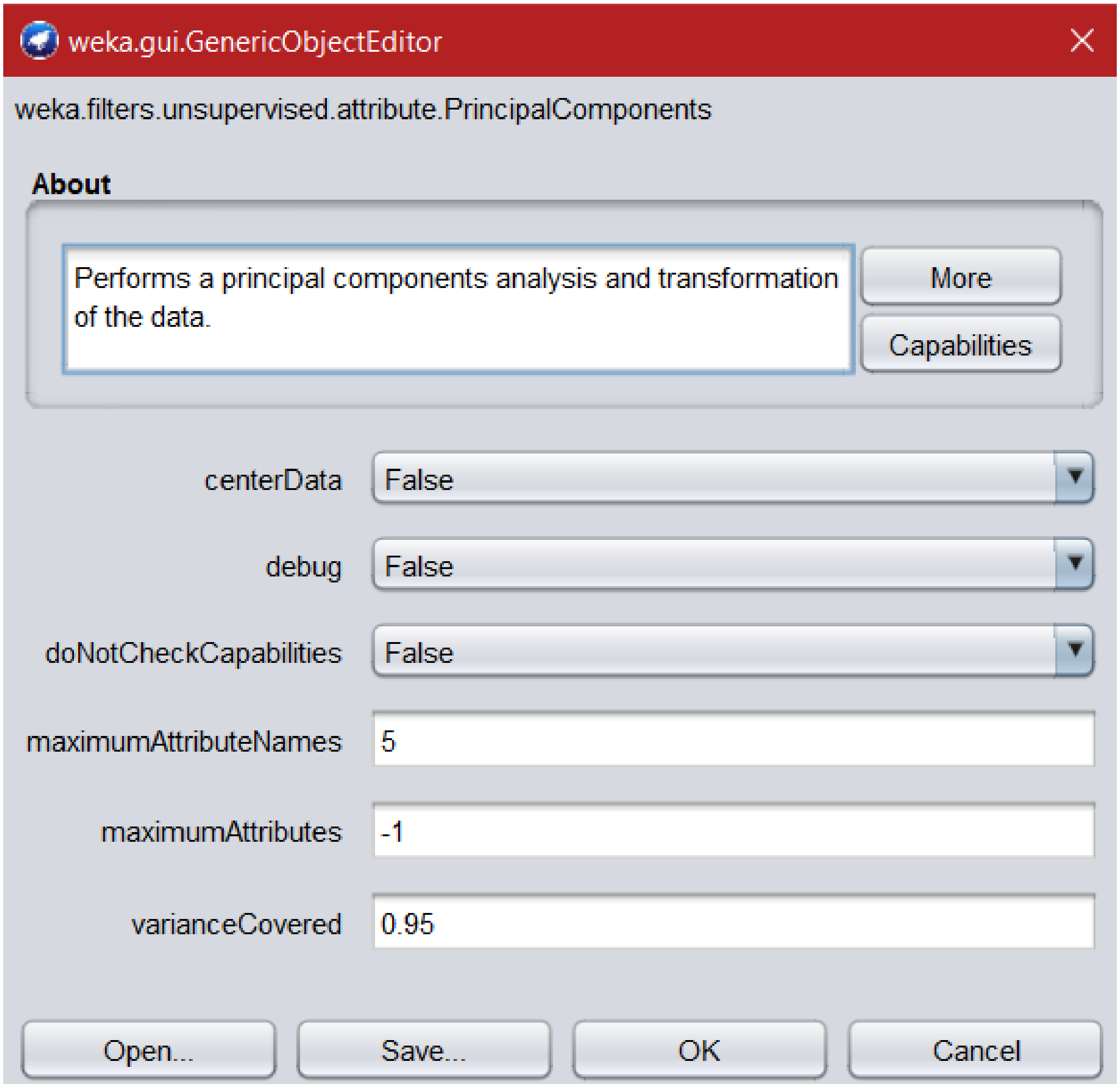
Clustering

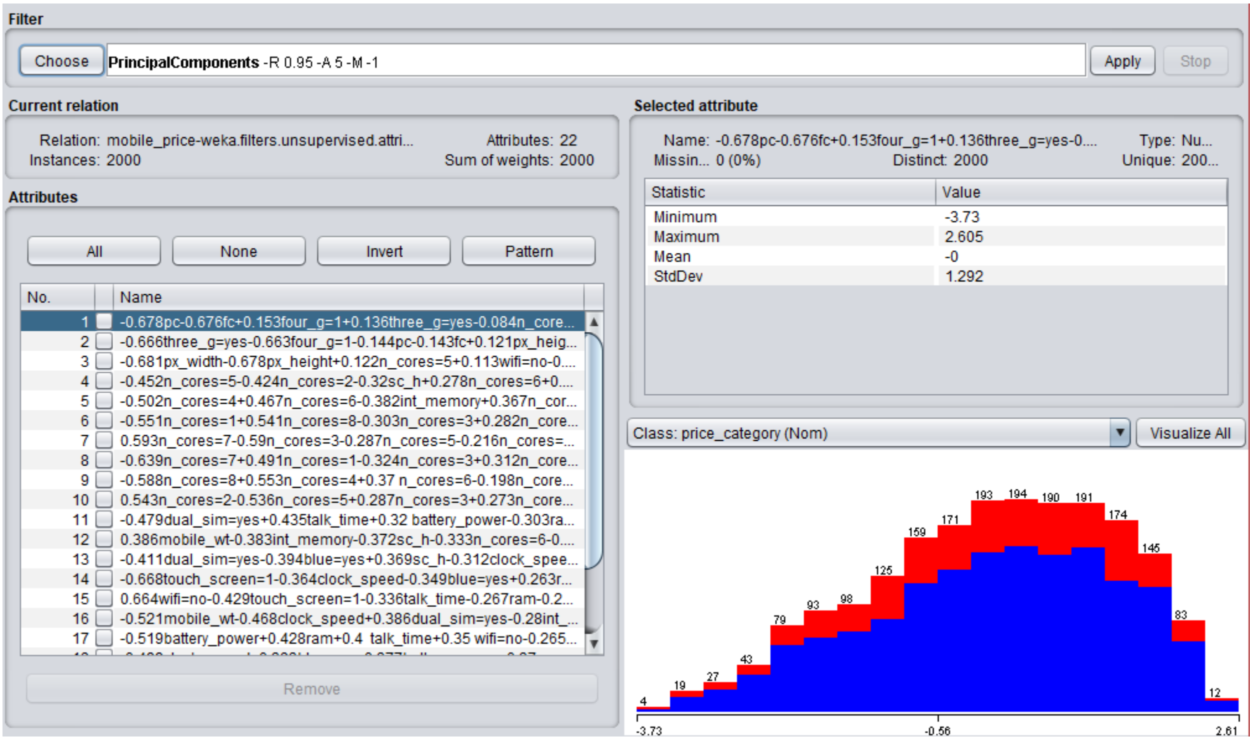
For clustering, we use the k-means algorithm. We set k as 2, since we will be using the classes to cluster evaluation and therefore would want two centroids. For the distance measure, we use the default Euclidean distance. We first run the algorithm on the original data set with some pre-processing (discretization and one-hot encoding). The results are presented below.

```
Time taken to build model (full training data) : 0.02 seconds

=== Model and evaluation on training set ===

Clustered Instances

0      1022 ( 51%)
1       978 ( 49%)


Class attribute: price_category
Classes to Clusters:

   0   1  <-- assigned to cluster
 762 738 | 0
 260 240 | 1

Cluster 0 <-- 0
Cluster 1 <-- 1

Incorrectly clustered instances :       998.0    49.9    %
```



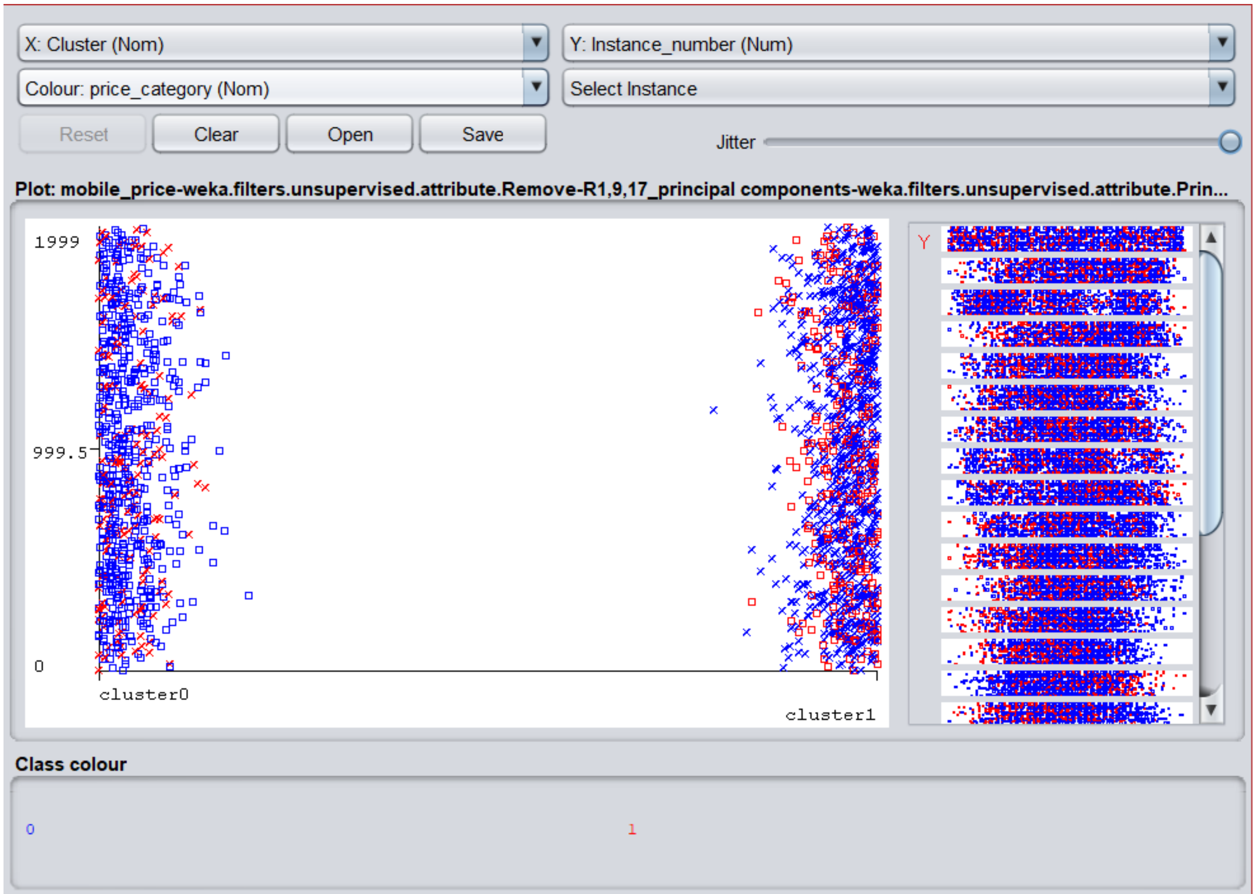
The clustering algorithm did not provide any benefit as it classified almost 50% of the mobile prices in the wrong category, which is only as good as random guessing. To improve the algorithm, we decided to reduce the dataset with Principal Components Analysis that explained 95% of the variance.


weka.gui.GenericObjectEditor
weka.filters.unsupervised.attribute.PrincipalComponents
About
Performs a principal components analysis and transformation of the data.
More
Capabilities
centerData
False
debug
False
doNotCheckCapabilities
False
maximumAttributeNames
5
maximumAttributes
-1
varianceCovered
0.95
Open...
Save...
OK
Cancel



By combining attributes into a regression of attributes, we have achieved our top ranked PC attribute with a standard deviation of 1.29, and the last PC attribute has a StdDev of 0.938. Using this data for the k-means algorithms, where k=2, gives the below result.

```
Time taken to build model (full training data) : 0.02 seconds

=== Model and evaluation on training set ===

Clustered Instances

0        750 ( 38%)
1       1250 ( 63%)


Class attribute: price_category
Classes to Clusters:

   0   1  <-- assigned to cluster
 577 923 | 0
 173 327 | 1

Cluster 0 <-- 1
Cluster 1 <-- 0

Incorrectly clustered instances :       904.0    45.2    %
```



Though not much, the PCA reduction helped increase the accuracy of the clustered instances into the right classes by 5.7% . In context, the 54.8% of mobile phone records have been correctly clustered into high-/low-price categories. Since the visualization does not present a clear separation, clustering might not be a suitable algorithm to group mobile phone data in price categories.

Learning Relation Context

Association mining rules is an unsupervised machine learning technique that helps us to find patterns, correlation and associations between attributes. In the context of the mobile phone dataset, given a set of transactions where each transaction consists of k-itemsets, we are going to apply the following metrics: support, confidence and lift, to predict the category of the mobile phone prices based on the occurrences of other items in the transactions. Association rule mining gives companies an indicator on which certain components of a phone should be grouped together to design a high price mobile phone. We used an association mining technique known as Apriori Algorithm. Apriori Algorithm identifies frequent itemsets based on minimum support and confidence or lift threshold. It is a more efficient method to prune away infrequent itemsets. The algorithm generates association rules which help companies to identify how well associated or correlated certain components are to the price of the mobile phone. These association rules help companies design high priced mobile phones.

Classification is a supervised predictive modelling technique to predict the class of a given input data. The model is trained using cross-validation which splits the data into 10 sub samples of equal or similar sizes. In each iteration of the splitting, the model utilizes nine of those sets in the training model, and uses one set as the test set. Test set is the unseen data, to which the classification model will predict the class for the given input unseen data. For the mobile phone dataset, we use a binary classification to determine whether a mobile phone is a high priced mobile phone. This binary classification involves two states: 1 and 0. For our dataset, we allocated 1 as "high" and 0 and "low" . The following two machine learning techniques: decision tree and support vector machine, are popular methods for binary classification. The overall aim of classification is to predict whether the mobile phone price is high or low based on the attributes and instances given within the dataset.

As previously stated, the decision tree forms a tree-like hierarchical structure, which bases its methodologies on the if-then rule set. The decision tree recursively splits the data downwards in a divide and conquer manner. In our model, the decision tree calculates the information gain for each attribute, and the attribute with the highest information gain value is selected as the root node, so in our case, the RAM. The decision tree splits the data based on the decision into separate branches, and repeats the same process but only for instances that satisfy the condition. This process will continue until the true nodes are found. For a better explanation, refer to the example below.

Support vector machines find the best decision boundary between high and low classifications. The SVM takes all the instances as input and outputs the best hyperplane that maximises the margins from the nearest/closest element/instance of each class. The decision boundary is a linear line that separates those two classes, where all the instances on one side of the line represent high, and low on the other side. In our case, we used a non-linear kernel Polykern to segregate the classes. So when the algorithm is trained with the unseen data, the model will decide whether the price of the mobile phone is low or high depending on which side of the boundary lies on.

Clustering is an unsupervised learning algorithm that splits the instances in a specified number of groups, known as clusters. Clustering is a technique that groups together all the attributes with similar traits into clusters. In order to produce a good cluster, it must consist of high intra-class similarity and low inter-class similarity. In the context of the mobile price dataset, the aim is to group the features with similar components and its respective size of the component, to differentiate between the high and low class price categories.

The clustering method applied to the dataset is k-means. The k-mean algorithm separates the instances into k clusters where each instance is assigned to the nearest cluster centroid. In the contexts of the mobile price dataset, the number of clusters we chose is two. K-means is being used for classification to determine the price of the mobile phone. So in this case, one cluster is allocated to the high price features, and the second cluster is allocated to the low price features. The algorithm selects two inputs from the dataset, and assigns that input value to the cluster whose distance is minimal with the cluster's centroid. Recompute the centroid, and assign input to the relevant clusters, and repeat the whole process. By the end of the model, we should receive two clusters, so when tested with unseen data, it will predict whether the model is high or low. However, the visualisations show that two clusters had no clear separation for the two price categories and hence we can conclude that clustering may not be a suitable method to straightforwardly get market insights into specifications that make up a higher priced mobile phone.