CA2: DETECT PATIENTS THAT HAVE ASD
REPORT BY: CONOR GRIFFIN X00111602

x00111602

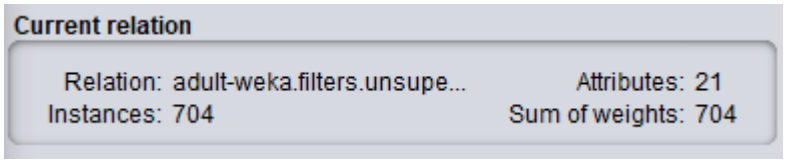
IT TALLAGHT Conor Griffin

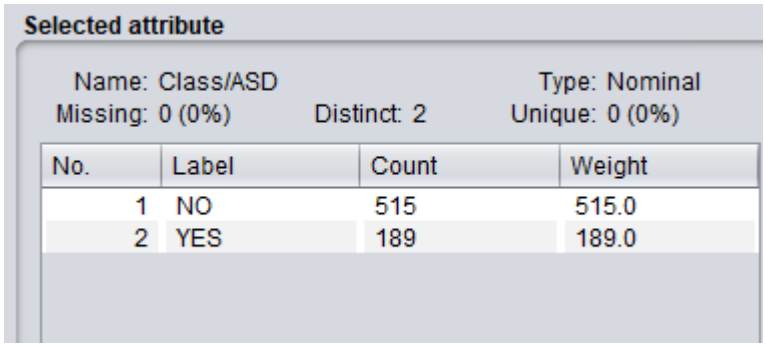
Contents

Section One

1. Analyse the dataset

From looking at the dataset in Weka explorer there are a few things to take notice of. Looking at the current relation section in Weka explorer it tells us that there are 704 rows of data and 21 columns/attributes in the dataset. For the attributes question 1-10 with a scale value of nominal type, each attribute has no missing values and no unique values, also each attribute has 2 distinct values. Another thing to take note of is that there are only 2 numeric attributes in the dataset, this can be a negative aspect in a dataset for reaching conclusions as it is good to have numeric attributes because they give a machine learning algorithm constant and final values to use. For each of the numeric attributes they have almost no missing values, age has just 2 missing values. Also the result numeric attribute has a minimum 0, maximum 10, Mean 4.875, Standard deviation 2501 this gives an idea of the spread of data. Since the result attribute is the sum of the A1_Score to A10_Score and since it is perfectly correlated with those attributes it will have an impact any models produced so I will remove it from the dataset. The attribute relation has a high amount of missing values 95 (13%) this will be interesting to see how algorithms interpret this amount of missing data and will this ratio of missing data mean I will have to remove or impute the values. The class attribute determines if an adult is autistic, “Yes”, or not, “No”, there is a higher count of patients in the No category 515 and 189 Yes. I would say that this is an accurate dataset in the sense you would expect to see a higher count in the no category based off the 10 questions.

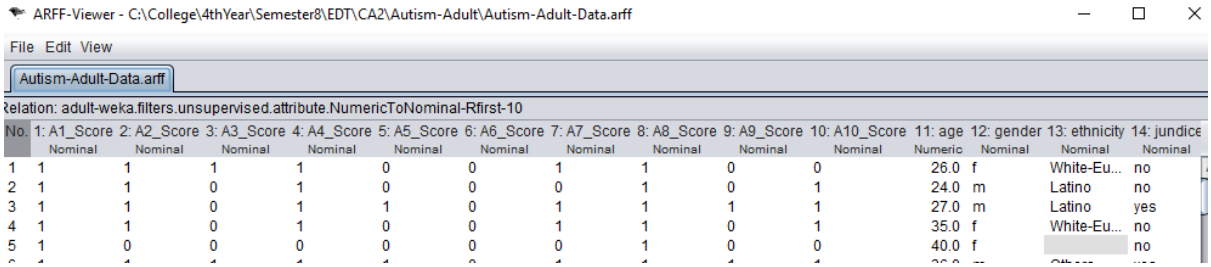




2. Prepare a number of views (formats) of the dataset

Original Dataset

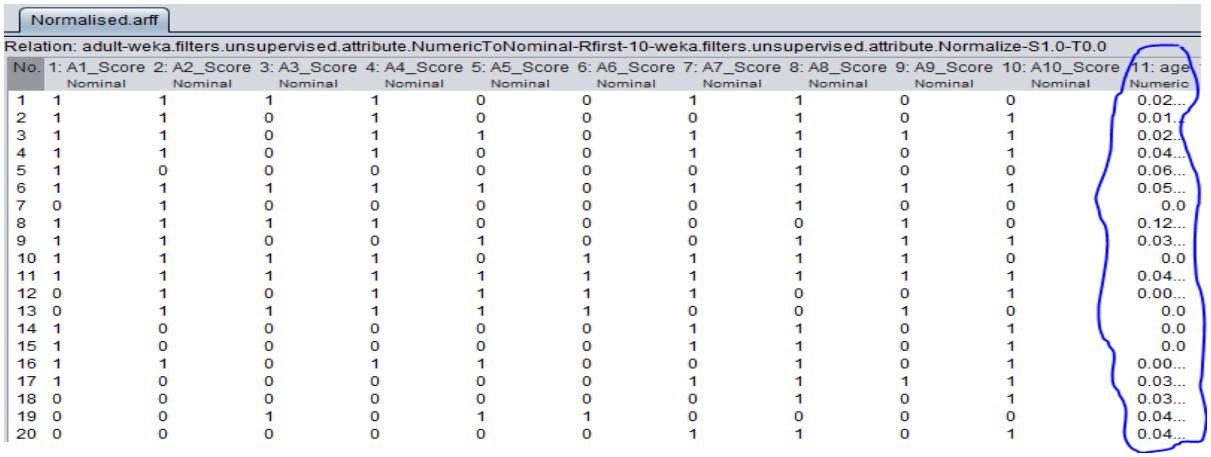
I loaded the dataset into Weka arff viewer.



No.	1: A1_Score	2: A2_Score	3: A3_Score	4: A4_Score	5: A5_Score	6: A6_Score	7: A7_Score	8: A8_Score	9: A9_Score	10: A10_Score	11: age	12: gender	13: ethnicity	14: jundice
	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal	Numeric	Nominal	Nominal	Nominal
1	1	1	1	1	0	0	1	1	0	0	26.0	f	White-Eu...	no
2	1	1	0	1	0	0	0	1	0	1	24.0	m	Latino	no
3	1	1	0	1	1	0	1	1	1	1	27.0	m	Latino	yes
4	1	1	0	1	0	0	1	1	0	1	35.0	f	White-Eu...	no
5	1	0	0	0	0	0	0	1	0	0	40.0	f		no

Normalised View

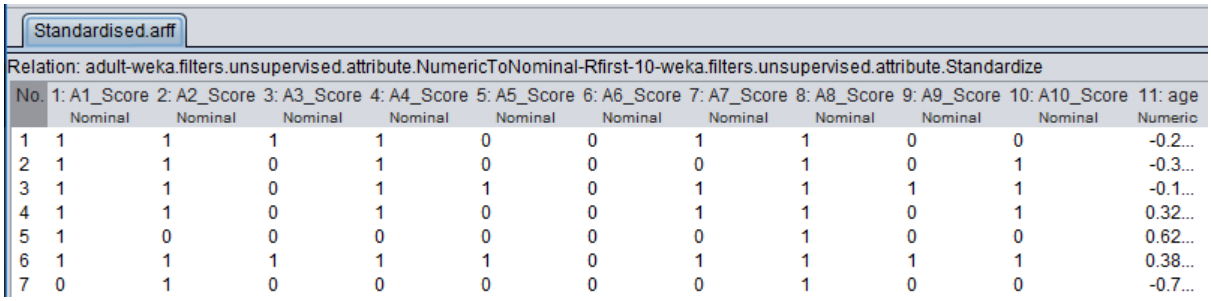
To normalise the view I first created a duplicate of the original dataset, then loaded it into weka and began the process in weka explorer. Data normalisation is the process of rescaling one or more numeric attributes to the range of 0 to 1. This is a good technique to use when you do not know the distribution of your data or when it is not a bell curved distribution. In weka explorer I chose the normalize filter and applied it to the autism dataset, renamed to normalised. From the final view you can see that the age numeric attribute has been normalised from ranges 0 to 1.



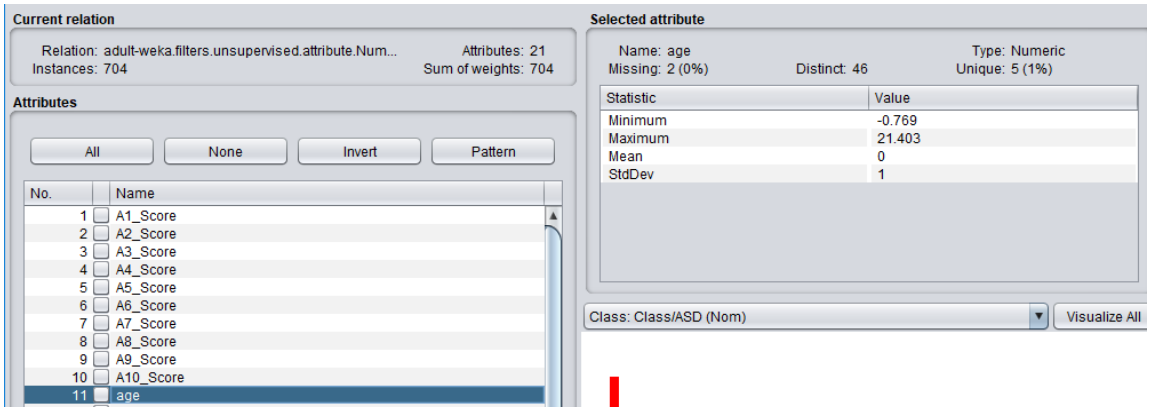
No.	1: A1_Score	2: A2_Score	3: A3_Score	4: A4_Score	5: A5_Score	6: A6_Score	7: A7_Score	8: A8_Score	9: A9_Score	10: A10_Score	11: age
	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal	Numeric
1	1	1	1	1	0	0	1	1	0	0	0.02...
2	1	1	0	1	0	0	0	1	0	1	0.01...
3	1	1	0	1	1	0	1	1	1	1	0.02...
4	1	1	0	1	0	0	1	1	0	1	0.04...
5	1	0	0	0	0	0	0	1	0	0	0.06...
6	1	1	1	1	1	0	1	1	1	1	0.05...
7	0	1	0	0	0	0	0	1	0	0	0.0
8	1	1	1	1	0	0	0	0	1	0	0.12...
9	1	1	0	0	1	0	0	1	1	1	0.03...
10	1	1	1	1	0	1	1	1	1	0	0.0
11	1	1	1	1	1	1	1	1	1	1	0.04...
12	0	1	0	1	1	1	1	0	0	1	0.00...
13	0	1	1	1	1	1	0	0	1	0	0.0
14	1	0	0	0	0	0	1	1	0	1	0.0
15	1	0	0	0	0	0	1	1	0	1	0.0
16	1	1	0	1	1	0	0	1	0	1	0.00...
17	1	0	0	0	0	0	1	1	1	1	0.03...
18	0	0	0	0	0	0	0	1	0	1	0.03...
19	0	0	1	0	1	1	0	0	0	0	0.04...
20	0	0	0	0	0	0	1	1	0	1	0.04...

Standardised View

To standardise the view I began by creating a copy of original dataset called standardised.arff and loaded it into weka explorer. Data standardisation is the process of rescaling one or more attributes so that they have a mean value of 0 and a standard deviation of 1. I standardised each of the attributes in weka by choosing the standardize filter and applied it accordingly. From the view you can see that the numeric attributes have been standardised, also in explorer you can see for example the age attribute has a mean of 0 and standard deviation 1.

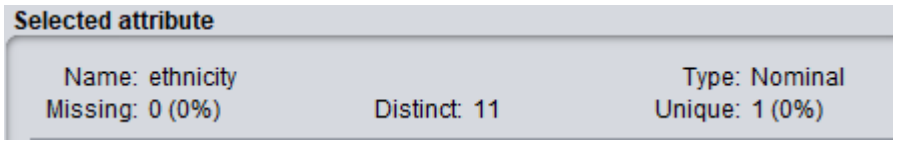


No.	1: A1_Score	2: A2_Score	3: A3_Score	4: A4_Score	5: A5_Score	6: A6_Score	7: A7_Score	8: A8_Score	9: A9_Score	10: A10_Score	11: age
	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal	Numeric
1	1	1	1	1	0	0	1	1	0	0	-0.2...
2	1	1	0	1	0	0	0	1	0	1	-0.3...
3	1	1	0	1	1	0	1	1	1	1	-0.1...
4	1	1	0	1	0	0	1	1	0	1	0.32...
5	1	0	0	0	0	0	0	1	0	0	0.62...
6	1	1	1	1	1	0	1	1	1	1	0.38...
7	0	1	0	0	0	0	0	1	0	0	-0.7...



Missing View

To impute the average into attributes with missing data I first loaded the dataset into weka and created a missing.arrf file. I then loaded it into weka explorer and used the ReplaceMissingValues filter. The results replaced all missing values with the average value. For example you can see below the attribute ethnicity's values were imputed with the average ethnicity.



Normal view

13: ethnicity
Nominal
White-Eu...
Latino
Latino
White-Eu...
Others
Black
White-Eu...
White-Eu...
Asian
White-Eu...
Middle E...
Middle E...
Middle E...
White-Eu...
Middle E...
Black
Middle E...
Middle E...

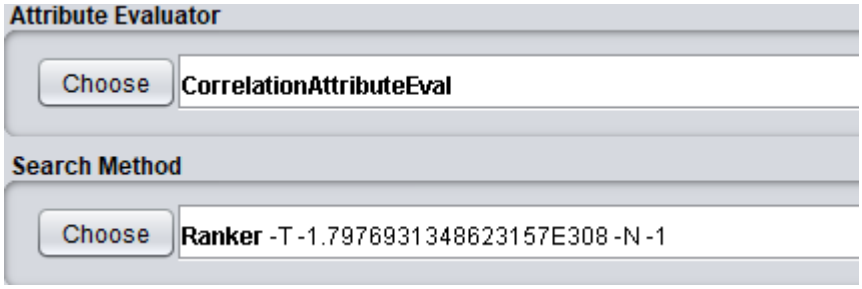
Missing View

13: ethnicity
Nominal
White-Eu...
Latino
Latino
White-Eu...
White-Eu...
Others
Black
White-Eu...
White-Eu...
Asian
White-Eu...
Middle E...
White-Eu...
White-Eu...
White-Eu...
Middle E...
Middle E...
White-Eu...
Middle E...
White-Eu...
White-Eu...
Black
Middle E...
Middle E...
White-Eu...
White-Eu...

3. Attribute Selection (Feature selection)

For attribute selection I divided the process into two parts: Attribute evaluator, this is used to evaluate the attribute in the context of the class variable. The Search Method, the technique to try or navigate combinations of attributes in the dataset in order to get on a short list of chosen features.

The first attribute selection method I used to determine which attributes to choose is the correlation attribute evaluation technique. I used this method with a Ranker search method which evaluates each attribute and lists the results in rank order. I configured both the attribute evaluator and search method to work with each other in Weka. I ran the algorithm on the dataset and go the following results.



Attribute selection output

```
Ranked attributes:
 0.63558    9 A9_Score
 0.59209    6 A6_Score
 0.537      5 A5_Score
 0.46995    4 A4_Score
 0.44107    3 A3_Score
 0.38592   10 A10_Score
 0.35143    7 A7_Score
 0.31138    2 A2_Score
 0.29763    1 A1_Score
 0.23716    8 A8_Score
 0.17741   15 austim
 0.15073   13 ethnicity
 0.11621   16 contry_of_res
 0.10215   14 jundice
 0.08038   12 gender
 0.05917   11 age
 0.04404   17 used_app_before
 0.00538   19 relation
 0         18 age_desc

Selected attributes: 9,6,5,4,3,10,7,2,1,8,15,13,16,14,12,11,17,19,18 : 19
```

The technique to use here is to take a look at the correlation in attributes. Correlation is calculated for each variable predictor variable and the process to follow is to select only those attributes that have a moderate to high positive or negative correlation (close to -1 or 1) and drop those attributes with a low correlation (values close to 0). The use of the ranker search method also displays the attributes in a ranked order. Looking at the output you can see that the predictor attribute A9_Score has the highest correlation with the class attribute (0.63558), A6_score, A5_Score, A4_Score and A3_Score also have a high correlation with the class/ASD variable. If we set the cut-off point at 0.2

for relevant attributes then the remaining 9 attributes could possibly be removed, but this seems like a lot of data to remove so I would either set the cut-off lower or use other techniques to make a final decision on what attributes to remove.

The second method I used for attribute selection is the Information gain technique, which is used to calculate the info gain for each attribute for the output variable ASD. Entry values go from 0 (no information) to 1 (maximum information). Those attributes that contribute more information will have a higher information gain value and can be selected whereas those that do not add much information will have a lower score and can be removed. In Weka I use the InfoGainAttributeEval in attribute evaluator and again use the Ranker search method. The following output displays the results.

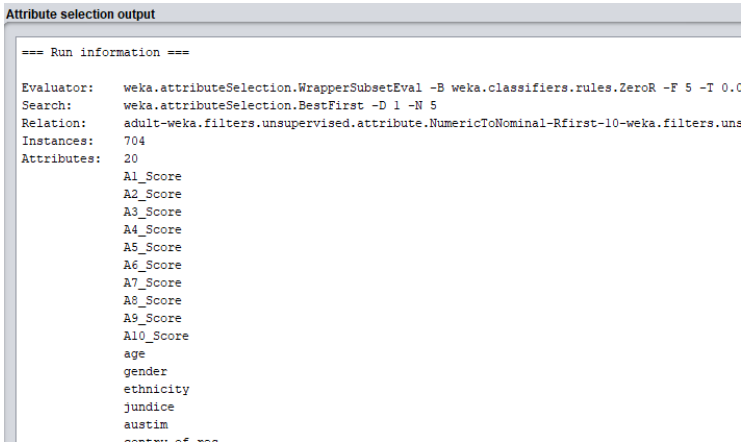
Attribute selection output

```
Ranked attributes:
 0.28855   9 A9_Score
 0.24099   6 A6_Score
 0.24037   5 A5_Score
 0.21054  16 contry_of_res
 0.17514   4 A4_Score
 0.1473    3 A3_Score
 0.12124  10 A10_Score
 0.08937   7 A7_Score
 0.08855  13 ethnicity
 0.07816   1 A1_Score
 0.07095   2 A2_Score
 0.04421   8 A8_Score
 0.02066  15 austim
 0.00698  14 jundice
 0.00466  12 gender
 0.00128  17 used_app_before
 0.00113  19 relation
 0        11 age
 0        18 age_desc

Selected attributes: 9,6,5,16,4,3,10,7,13,1,2,8,15,14,12,17,19,11,18 : 19
```

From the output you can see that most of the attributes have a low information gain, the top ranked attribute is A9_Score with an information gain of 0.28855, but if we set the cut-off point at 0.2 then we could keep other attributes like A6, A5_Score and contry_of_res and the rest could be removed from the dataset, this would be quite a high cut off point considering only 4 of the attributes are above 0.2, I think since most attributes have a low information gain it's probably best to use other methods for attribute selection.

The third selection method I used to determine what attributes to use in the modelling stage is the learner based feature selection. This is a powerful learning algorithm and is used to gather subsets of attributes, the subset with the best results and performance is taken as the selected subset. The feature selection method used is the WrapperSubsetEval technique and uses a BestFirst search method as it uses less compute time. I modified the configuration to instead of ZeroR, use J48 in trees, this gives a better preferred subset in the output.

The run information for this algorithm tells us that it ran the subset method with a best first search and used 704 instances over 20 attributes, it found 91 subsets and the merit of the best subset found was 0.732. There were no final selected attributes using this algorithm.




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

Search Method:
        Best first.
        Start set: no attributes
        Search direction: forward
        Stale search after 5 node expansions
        Total number of subsets evaluated: 91
        Merit of best subset found:    0.732

Attribute Subset Evaluator (supervised, Class (nominal): 20 Class/ASD):
        Wrapper Subset Evaluator
        Learning scheme: weka.classifiers.rules.ZeroR
        Scheme options:
        Subset evaluation: classification accuracy
        Number of folds for accuracy estimation: 5

Selected attributes:
```

Looking back over the three techniques, you can see that a few of the attributes overlapped into each of the methods used. Most of the other attributes didn't perform well in each of the feature selection methods but this doesn't necessarily mean you can get rid of all them, I would say it best to keep attributes that performed well in the first two techniques like A9_Score, A6_score, A5_Score, A4_Score and A3_Score as well as contry_of_res attributes. I would add each of these in the modelling stage.

4. Use kNearestNeighbor (IBK) Classifier on the dataset

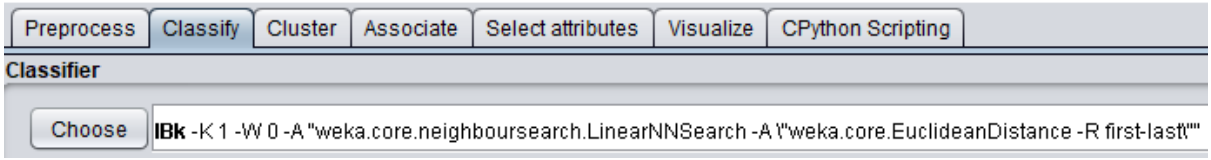
Nearest neighbour: to classify a new instance, search training set for one that's most like it. Searches the training set for the one that most like the new instance. We have our class variable with a yes or no answer, we now have the unknown instance, and the nearest neighbour method produces the new instance based on previous data most similar to it. It uses a similarity function by measuring the Euclidean distance, Manhattan distance, normalize the attributes to be between 0 and 1 and this gives us the kNearest Neighbor. KNearest protects us from noisy data, we choose the k nearest neighbours and it chooses the majority class among the neighbours.

First, I found the baseline accuracy for the dataset, I ran the rules->ZeroR algorithm on the dataset, and this gave an output of 73.15%. I was able to base my accuracy of kNearest results off of the ZeroR result of 73%.

ZeroR

```
Correctly Classified Instances         515               73.1534 %
Incorrectly Classified Instances       189               26.8466 %
```

KNearest



If any of the values for k have a greater accuracy than baseline performance then the more optimal that solution is. For k=1 the number of correctly classified instances was 655, incorrect classified instances was 49, with a 93% accuracy, k=3 the number of correctly classified instances was 664, incorrect instances was 40 with a 94.3% accuracy, k=5 the number of correctly classified instances was 662, incorrect instances was 42, with an accuracy of 94%, k=7 the number of correctly classified instances was 672, incorrect instances was 32, with an accuracy of 95.4% and k=15 the number of correctly classified instances 676, incorrect instances was 28, with an accuracy of 96%.

K=1

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

Correctly Classified Instances         655               93.0398 %
Incorrectly Classified Instances        49                6.9602 %
Kappa statistic                          0.8254
Mean absolute error                      0.0723
Root mean squared error                  0.2634
Relative absolute error                 18.3879 %
Root relative squared error             59.4413 %
Total Number of Instances              704
```

K=3

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

Correctly Classified Instances         664               94.3182 %
Incorrectly Classified Instances        40                5.6818 %
Kappa statistic                          0.8577
Mean absolute error                      0.0734
Root mean squared error                  0.2056
Relative absolute error                 18.6728 %
Root relative squared error             46.3917 %
Total Number of Instances              704
```

K=5

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

Correctly Classified Instances         662               94.0341 %
Incorrectly Classified Instances        42                5.9659 %
Kappa statistic                          0.8491
Mean absolute error                      0.0779
Root mean squared error                  0.1991
Relative absolute error                 19.8237 %
Root relative squared error             44.9324 %
Total Number of Instances               704
```

K=7

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

Correctly Classified Instances         672               95.4545 %
Incorrectly Classified Instances        32                4.5455 %
Kappa statistic                          0.8847
Mean absolute error                      0.0772
Root mean squared error                  0.1844
Relative absolute error                 19.6499 %
Root relative squared error             41.6007 %
Total Number of Instances               704
```

K=15

```
Correctly Classified Instances         676               96.0227 %
Incorrectly Classified Instances        28                3.9773 %
Kappa statistic                          0.8997
Mean absolute error                      0.0844
Root mean squared error                  0.1786
Relative absolute error                 21.4626 %
Root relative squared error             40.3103 %
Total Number of Instances               704
```

The optimal value of k I would say from looking at the results is 15 as it gave the most accurate % classified instances and more instances were produced (676). It seems that from looking at the results the greater the value for K then the better and more accurate the output, testing for higher values of k such as 20, the accuracy of these models were less than the model of k=15. Using 30 nearest neighbours for classification produced 679 correct instances, 96.4% accuracy, using 50, produced 680 correct instances, 96.5% accuracy, you could say this is the most accurate model to use and optimal value of k.

A few things should be noted though with choosing a high value of k, a higher k averages more voters in each prediction and is more resilient to outliers. Larger values of k will have smoother decision boundaries which mean lower variance but increased bias. Though this is true, when k is smaller we are restraining the region of a given prediction and forcing our classifier to be “more blind” to the overall distribution. A small value for k provides a flexible fit, which will have low bias but high variance.

5. Two Machine Learning Algorithms for Evaluation

Rules.DecisionTable

This machine learning algorithm uses a class for building and using a simple decision table majority classifier. Decision tables are one of the simplest hypotheses used in supervised machine learning algorithms. An algorithm inducing decision tables can sometimes outperform state of the art algorithms such as C4.5. It is one of many supervised machine learning algorithms that seek a hypothesis that will correctly predict the class of future unlabelled instances. In the case of the Autism-Adult dataset the decision table algorithm can be used to identify some of the unknown instances class identifier.

Decision tables give a visual representation which specifies which actions to perform depending on given conditions. The information expressed in decision tables could also be used in a programming language as a series of if-then-else and switch-case statements. Each decision in a decision table algorithm corresponds to a variable, relation or predicate whose possible values are listed among the condition alternatives. Each action is a procedure or operation to perform, and the entries specify whether (or in what order) the action is to be performed for the set of condition alternatives the entry corresponds to.

Tress RandomForest

The Ensemble method I have chosen to use in the evaluation stage is the trees.RandomForest classifier method. Ensemble algorithms are a powerful class of machine learning alforithm that combine the predictions from multiple models. Random Forest is an extension of the bagging algorithm for decision trees used for classification or regression. Random Forest is an improvement upon bagged decision trees that disrupts the greedy splitting algorithm during tree creation so that split points can only be selected from a random subset of the input attributes. This allows the algorithm to have a big effect decreasing the similarity between the bagged trees and in turn the resulting predictions.

Trees.SimpleChart

https://pdfs.semanticscholar.org/26d6/73f140807942313545489b38241c1f0401d0.pdf

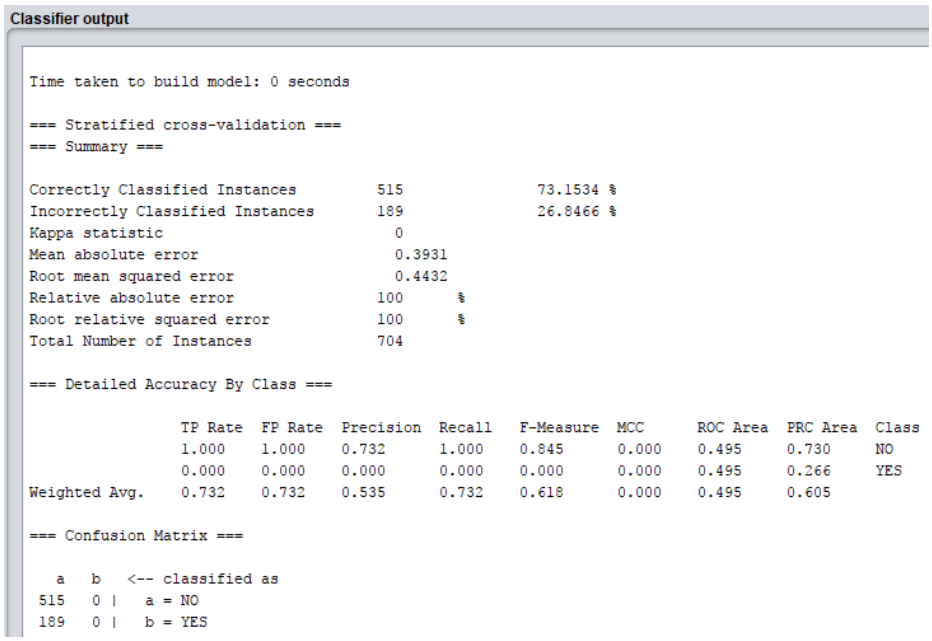
Simple chart is a classification tree algorithm used in weka. It is a data mining algorithm that creates a step by step guide for how to determine the output for a new data instance. The tree it creates is exactly that: a tree whereby each node in the tree represents a spot where a decision must be made based on the input, and you move to the next node and the next until you reach a leaf that tells you the predicted output.

Simple cart method is CART analysis which stands for Classification and regression trees. It is used for data exploration and predication. It is a classification technique that generates the binary decision tree. Output is a binary tree, it generates only two children. Entropy is used to choose the best splitting attribute. Simple Cart handles the missing data by ignoring that record. This algorithm is best for the training data. CART decision tree is a learning technique, which gives the results as either classification or regression trees, depending on categorical or numeric data in the dataset. It is a greedy algorithm in that it chooses the locally best discriminatory feature at each stage in the process. This is suboptimal but a full search for fully optimized search would be computationally very expensive. In the CART decision tree the dataset is split into two subgroups that are the most different with respect to the outcome. This procedure Is continued on each subgroup until some minimum subgroup size is reached.

6. Carry out an initial evaluation on Machine Learning algorithms

Rules.ZeroR

ZeroR is the simplest classification model and useful for determining a baseline performance for other classification methods. Without a baseline you do not know how well other algorithms are performing. Out of 704 instances in the dataset 515 were correctly classified (73%) and 189 were incorrectly classified (26%). With 73% accuracy this is useful for future classification methods to have a baseline performance, I can then evaluate if the other methods are useful in building a classification model. The ZeroR algorithm gives you a point of reference to which you can compare all other models that will be constructed.



```
Classifier output

Time taken to build model: 0 seconds

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

Correctly Classified Instances         515               73.1534 %
Incorrectly Classified Instances       189               26.8466 %
Kappa statistic                          0
Mean absolute error                      0.3931
Root mean squared error                  0.4432
Relative absolute error                100      %
Root relative squared error            100      %
Total Number of Instances              704

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               1.000    1.000    0.732      1.000    0.845      0.000    0.495     0.730     NO
               0.000    0.000    0.000      0.000    0.000      0.000    0.495     0.266     YES
Weighted Avg.  0.732    0.732    0.535      0.732    0.618      0.000    0.495     0.605

=== Confusion Matrix ===

   a   b   <-- classified as
 515   0 |   a = NO
 189   0 |   b = YES
```

Rules JRip

JRip algorithm produced a 92% correctly classified instances rate and 650 instances, with 7% incorrect, a total of 54 instances. The confusion matrix produced by the algorithm gave a total of aa + bb = 483 + 177 = 650 correctly classified instances, and ab + ba = 22 + 32 = 54 incorrectly classified instances. With a 92% accuracy using JRip this is a high accuracy but this does not take into account that the algorithm isn't sensitive to class distribution. Also with the fact that there is missing data in the dataset we don't have access to all possible information. Taking this into account though 92% is still a very high accuracy. The kappa statistic of 0.8079 tells us that the classifier is doing better than chance and this high mark is a good indication that this method is performing well on the dataset.

Classifier output

```
Time taken to build model: 0.09 seconds

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

Correctly Classified Instances         650               92.3295 %
Incorrectly Classified Instances        54                7.6705 %
Kappa statistic                          0.8079
Mean absolute error                      0.1004
Root mean squared error                  0.264
Relative absolute error                 25.5302 %
Root relative squared error             59.5664 %
Total Number of Instances              704

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0.938    0.116    0.956      0.938    0.947      0.808    0.923     0.951     NO
               0.884    0.062    0.839      0.884    0.861      0.808    0.923     0.827     YES
Weighted Avg.  0.923    0.102    0.925      0.923    0.924      0.808    0.923     0.918

=== Confusion Matrix ===

   a   b   <-- classified as
 483  32 |   a = NO
  22 167 |   b = YES
```

Bayes NaiveBayes

NaiveBayes classification is used as an estimator class. Using numeric attributes it chooses which values based on analysis of the dataset. The algorithm uses the normal distribution to model numeric attributes, it can also handle numeric attributes using supervised discretization. This methods output produced one that is better than he JRip output. With 669 correctly classified instances, 95% accuracy tells us that the algorithm performs well on the dataset and should be used in a classification model for this dataset. Only 4.9% inaccuracy and 35 incorrectly classified instances can be a good sign but taking into account the missing data a high accuracy could be a bad sign, how is the accuracy so high without imputing missing values. The true positives in NaiveBayes are very high at over 0.9, giving us a high count of correctly classified instances. The ROC Area value is also important, an optimal classifier will have ROC area values approaching 1, the ROC value produced by the NaiveBayes algorithm was a detailed accuracy by class of 0.999. This is an important value in terms of we know that the algorithm is making detailed and practical decisions on the class and is not random guessing, a ROC value of closer to 0.5 would mean that the classifier is random guessing.

Classifier output

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

Correctly Classified Instances         683               97.017  %
Incorrectly Classified Instances        21                2.983  %
Kappa statistic                          0.9262
Mean absolute error                      0.0326
Root mean squared error                  0.136
Relative absolute error                  8.2965 %
Root relative squared error             30.6985 %
Total Number of Instances              704

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0.963    0.011    0.996      0.963    0.979      0.928    0.999     1.000     NO
               0.989    0.037    0.908      0.989    0.947      0.928    0.999     0.997     YES
Weighted Avg.  0.970    0.018    0.972      0.970    0.971      0.928    0.999     0.999

=== Confusion Matrix ===

   a   b   <-- classified as
 496  19 |   a = NO
   2 187 |   b = YES
```

Functions SMO

The classifier SMO (Sequential Minimal Optimization) is implemented through globally replacing all missing values and transforms nominal attributes into binary ones. It also normalizes all attributes by default. The SMO algorithm also uses the RBF Kernel function and is used for solving the quadratic programming QP problem that arises during the training of support vector machines (SVM). Both the RBF Kernel and the SVM must be supplied by the user/machine in order for the SMO algorithm to run on a dataset. In our case the Autism-Adult dataset the SMO algorithm runs on the dataset and breaks the problem into a series of smallest possible sub-problems, which are then solved analytically. The times taken to run the algorithm was 0.06 seconds, the summary pointed out many things. Every instance (704) was correctly classified 100%, a = NO produced 515 instances and b = YES produced 189 instances. There was no error produced and the Kappa statistic was 1 meaning the classifier is performing well. Detailed accuracy by class report tells us that TP rate, Precision, Recall, F-measure MCC and ROC Area all performed at 1.00 and the false positive rate at 0.0.

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

Correctly Classified Instances         704               100      %
Incorrectly Classified Instances         0                 0      %
Kappa statistic                          1
Mean absolute error                      0
Root mean squared error                  0
Relative absolute error                  0      %
Root relative squared error              0      %
Total Number of Instances              704

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     NO
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     YES
Weighted Avg.    1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000

=== Confusion Matrix ===

   a   b   <-- classified as
 515   0 |   a = NO
   0 189 |   b = YES
```

Lazy IBk

Lazy IBk K-nearest neighbour's classifier is used as a classifier algorithm that selects the appropriate value of K based on cross-validation. Options for running the algorithm include the weight neighbours by the inverse of their distance, the weight neighbours by 1 – their distance, the number of nearest neighbours used in classification, etc.

The optimal value of k I determined before was 15 as it gave the most accurate % classified instances and more instances were produced (676). It seems that from looking at the results the greater the value for K then the better and more accurate the output but this is not the case, I also tested for greater values of k such as 20 and 50, the accuracy of these models were less than the model of k=15.

Trees J48

When running the J48 algorithm on the dataset the first number in the tree indicates the total number of instances reaching the leaf, the second number is the number (weight) of those instances that are misclassified. The number of correctly classified instances was 652 and 52 incorrect instances, 92% correct and 7% incorrect. Each of the detailed accuracy reports showed that TP Rate Precision and ROC area are all close to 1, this is a positive response and this classifier is one that should be using in a classification model.

```
=== Summary ===

Correctly Classified Instances         652               92.6136 %
Incorrectly Classified Instances        52                7.3864 %
Kappa statistic                          0.8144
Mean absolute error                      0.0932
Root mean squared error                  0.2556
Relative absolute error                 23.7005 %
Root relative squared error             57.6811 %
Total Number of Instances              704

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure  MCC      ROC Area  PRC Area  Class
               0.942     0.116     0.957      0.942     0.949      0.815    0.938     0.964     NO
               0.884     0.058     0.848      0.884     0.865      0.815    0.938     0.832     YES
Weighted Avg.  0.926     0.101     0.927      0.926     0.927      0.815    0.938     0.929

=== Confusion Matrix ===

   a   b   <-- classified as
 485  30 |   a = NO
  22 167 |   b = YES
```

Trees RandomForest

Random Forest uses Bagging techniques to achieve its final output. This is a statistical estimation technique where a statistical quantity for example the mean is estimated from multiple samples of your data and you are interested in a more robust estimate of a statistical quantity. Random samples of the training data are drawn with replacement and used to train multiple different machine learning models. Each model is then used to make a prediction and the results are averaged to give a more robust prediction.

```
=== Classifier model (full training set) ===

RandomForest

Bagging with 100 iterations and base learner

weka.classifiers.trees.RandomTree -K 0 -M 1.0 -V 0.001 -S 1 -do-not-check-capabilities

Time taken to build model: 0.29 seconds
```

From the default configuration random forests achieves an accuracy of 92%.

Classifier output

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

Correctly Classified Instances         649               92.1875 %
Incorrectly Classified Instances        55                7.8125 %
Kappa statistic                          0.796
Mean absolute error                      0.1498
Root mean squared error                  0.2408
Relative absolute error                 38.1163 %
Root relative squared error             54.3257 %
Total Number of Instances              704
```

The weka implementation of random forests performs classification on the dataset. The number of incorrectly classified instances was 55, this is a good performance and this algorithm has performed well on the model. The kappa and ROC statistics are high and this is a classifier I would use in the classification model.

Rules Decision Table

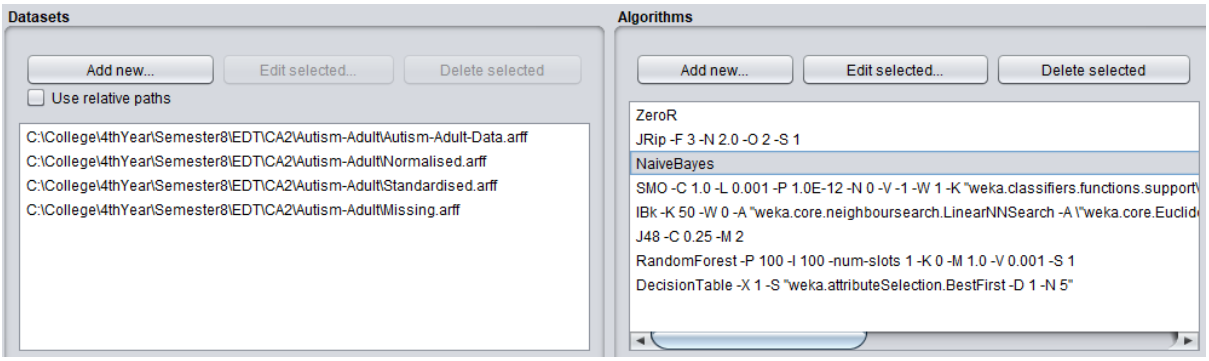
From the default configuration decision table achieves an accuracy of 91%, this is a very good accuracy when compared to our baseline accuracy of 73%. With a kappa statistic of 0.7856 this tells us that the classifier is performing better than chance.

```
Correctly Classified Instances         647               91.9034 %
Incorrectly Classified Instances        57                8.0966 %
Kappa statistic                          0.7856
Mean absolute error                      0.1686
Root mean squared error                  0.2658
Relative absolute error                 42.8882 %
Root relative squared error             59.9875 %
Total Number of Instances              704

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.967    0.212    0.926      0.967    0.946      0.789    0.959     0.983     NO
                 0.788    0.033    0.898      0.788    0.839      0.789    0.959     0.910     YES
Weighted Avg.    0.919    0.164    0.918      0.919    0.917      0.789    0.959     0.963

=== Confusion Matrix ===

   a   b   <-- classified as
 498  17 |   a = NO
  40 149 |   b = YES
```

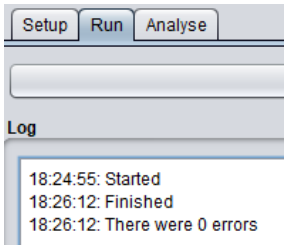
Experimenter View

I began this phase by loading the 4 .arff files into experimenter and setup the configuration for each algorithm. For this section I will be commenting on the performance, results, the best performing and the significant findings of each algorithm on each of the datasets.

I began the setup phase by loading each of the datasets into weka experimenter and then added each of the necessary algorithms.



I then ran the algorithm.



In the analyse step I got 3200 results in the source view. When configuring the test I tested with Paired T-Tester, Rows were the dataset and columns the scheme, I used a significance of 0.05 which gives us a confidence of 95% and I am analysing the Percent_correct.

```
Tester:     weka.experiment.PairedCorrectedTTester -G 4 -D 1 -R 2 -S 0.05
Analysing:  Percent_correct
Datasets:   4
Resultsets: 8
Confidence: 0.05 (two tailed)
Sorted by:  -
```

Results of the performance test showed that each dataset had a baseline accuracy of 73.15, the JRip classifier ran 100% accuracy on the Missing dataset and so did the SMO classifier, J48, Random Forest and Decision Table. Each of those algorithms was indicated with a 'v' beside their accuracy indicating that the result is significantly more/better than the base classifier. The 100 is based off the 10-fold cross validation that is performed 10 times. All of the algorithms performed significantly better than the baseline classifier. The NaiveBayes and IBk classifier with 50 nearest neighbours performed slightly differently on each dataset. I chose 50 nearest neighbours for the Nearest Neighbour because this model performed the best on the original dataset with 97% accuracy.

The Naive Bayes classifier performed best on the Missing.arff dataset, this is because any missing values were imputed with the average, running this algorithm on the training data will give a more accurate performance based on the fact all missing values were imputed with the average.

The IBk classifier also performed the best on the Missing.arff dataset, again due to the fact that there is no missing data in this dataset it is accurate that this dataset would perform the best under the IBk classifier.

The best performing algorithm is the functions.SMO classifier as it returned 100% accuracy on each model, I will use this classifier for creating a final version of the model is part 7.

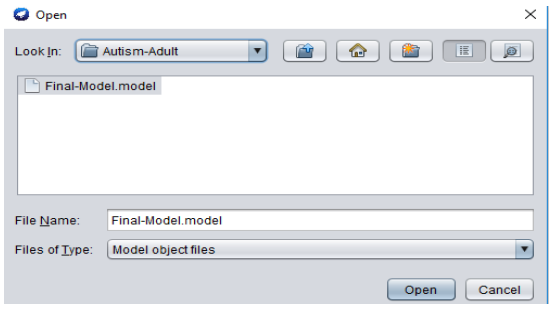
Dataset	(1) rules.Ze	(2) rules.	(3) bayes	(4) functi	(5) lazy.	(6) trees.	(7) trees.	(8) rules.
adult-weka.filters.unsupe(100)	73.15	92.40 v	95.21 v	100.00 v	96.08 v	92.19 v	92.48 v	91.92 v
adult-weka.filters.unsupe(100)	73.15	92.74 v	95.30 v	100.00 v	96.09 v	92.19 v	92.58 v	91.92 v
adult-weka.filters.unsupe(100)	73.15	92.49 v	95.10 v	100.00 v	96.11 v	92.19 v	92.64 v	91.92 v
adult-weka.filters.unsupe(100)	73.15	100.00 v	97.37 v	100.00 v	97.40 v	100.00 v	100.00 v	100.00 v
(v/ /*)		(4/0/0)	(4/0/0)	(4/0/0)	(4/0/0)	(4/0/0)	(4/0/0)	(4/0/0)

```
Key:
(1) rules.ZeroR
(2) rules.JRip
(3) bayes.NaiveBayes
(4) functions.SMO
(5) lazy.IBk
(6) trees.J48
(7) trees.RandomForest
(8) rules.DecisionTable
```

7. Final version of the model

I have now found a well performing machine learning model and have tuned it so that it is ready to be finalized so that I can make predictions on new data. I will be using the Unseen dataset in order to run a test using my final model to make predictions on whether or not the patient has autism or not. The classifier I am using is the functions.SMO algorithm as it performed the best on each of the datasets (Normal Data, Normalised, Standardised and Missing) in part 6.

I began by finalizing my model by running the SMO algorithm on the dataset, once this was done I saved the model and called it final-model.model. When this was complete I loaded the finalized model back into Weka's Result List.



Now that the model was ready to make prediction on Unseen data I began by looking at the unseen data so I knew what sort of unseen patients were in the dataset. A few things were noted in the unseen dataset, all patients are male, 3 are white European, all patients have different nationalities and are over the age of 18 and 3 of the patients are completing the test themselves.

Make Predictions on Unseen Data

I began the process of making predictions by loading the unseen dataset as our supplied test set, I then set up the configuration by unchecking all of the information we are not interested in, the output model, output per class stats, output confusion matrix, store predictions for visualization and I chose plaintext as the output predictions type. I then ran the SMO classification on the Unseen data.

The results output told us that the first 3 instances predicted a NO classification and 2 instances were predicted as YES patients. The error prediction for each instance was 1 which is a positive sign as this shows that the probability that the instance actually belongs to this class is estimated at 1. Each of the unseen cases was predicted successfully using the SMO classifier.

Classifier output

```
=== Re-evaluation on test set ===

User supplied test set
Relation:     adult-weka.filters.unsupervised.attribute.NumericToNominal-Rfirst-10
Instances:     unknown (yet). Reading incrementally
Attributes:   20

=== Predictions on user test set ===

    inst#     actual  predicted error prediction
        1        1:?       1:NO       1
        2        1:?       1:NO       1
        3        1:?       1:NO       1
        4        1:?      2:YES       1
        5        1:?      2:YES       1
```

Section Two

PART A, B AND C

The purpose of this section is to carry out unsupervised association rules on a transaction database from a local supermarket, I will hand trace the problem manually and use the Apriori algorithm to solve it. I am using Excel to assist me. I began by first creating a table with each of the items and creating an acronym for the item. I then put each of the transactions in the original table into a new table with the acronym for each transaction.

Conor Griffin	x00111602
Item	Acronym
Nikon Camera	NC
MICRO SD CARD	MSD
SHOOT TRIPOD	ST
PS4 CONSOLE	PC
PS4 GTA GAME	PGG
PS4 FIFA 19 GAME	PF19G
CHARGER	C
PS4 CONTROLLER	PCT
IPAD	I
AMAZON ECHO	AE
FITBIT	F
FITBIT WRIST BANDS	FWB

TRANSACTION	
ID	ITEMS PURCHASED
1	NC, MSD, ST, PC
2	MSD, ST, PC, PGG, PF19G
3	NC, ST, PF19G, C, PCT
4	MSD, ST, PC, PGG, PCT
5	PC, PGG, I, C, AE
6	NC, MSD, ST, PC, PGG, AE
7	NC, PC, PGG, I, AE
8	MSD, F, PCT, AE
9	ST, PC, I, AE
10	NC, MSD, PC, PGG, PCT
11	ST, PC, C, F, PCT
12	MSD, ST, PGG, PCT
13	MSD, ST, PC, I
14	NC, MSD, ST, PC
15	ST, C, F, FWB
16	NC, PGG, I, C, AE
17	C, PCT, AE
18	NC, MSD, PC, C, PCT
19	PC, PGG, PCT
20	MSD, ST, PC, PGG, C

The first step is to get a count of the amount of times each item appears throughout the transaction list. Then because we are applying the apriori algorithm to the dataset with minimum support 30% (i.e. (phi) = 6) I filtered out any of the items that were bought less than 6 times.

< 6 (MINIMUM SUPPORT 30%)	
STEP 2	COUNT
NC	8
MSD	11
ST	12
PC	14
PGG	10
C	8
PCT	9
AE	7

Step 3 involved getting the pairs and the count of each pair. And then filtering out what pairs are less than the minimum support count 30% (i.e. 6). This is the first iteration of frequent itemsets in the dataset that satisfy minimum support.

STEP 3	
ITEM PAIRS	COUNT
NC, MSD	5
NC, ST	4
NC, PC	6
NC, PGG	4
NC, C	3
NC, PCT	3
NC, AE	3
MSD, ST	8
MSD, PC	9
MSD, PGG	6
MSD, C	2
MSD, PCT	5
MSD, AE	2
ST, PC	9
ST, PGG	5
ST, C	4
ST, PCT	4
ST, AE	2
PC, PGG	8
PC, C	4
PC, PCT	5
PC, AE	3
PGG, C	3
PGG, PCT	4
PGG, AE	4
C, PCT	4
C, AE	3
PCT, AE	2

< 6 (MINIMUM SUPPORT 30%)		
ITERATION 1		
STEP 4 (LI)	COUNT	SUPPORT
NC, PC	6	6/20 = 30%
MSD, ST	8	8/20 = 40%
MSD, PC	9	9/20 = 45%
MSD, PGG	6	6/20 = 30%
ST, PC	9	9/20 = 45%
PC, PGG	8	8/20 = 40%

The next step was to get pairs of three in the transaction list. To find a self join we have to find two pairs with the same first acronym. The only items that appear like this are MSD, ST and MSD, PC. After this I then applied the same process of filtering out the pairs that are less than the minimum support. The only pair with higher percentage than the minimum support is the MSD, ST, PC pair with a count of 7.

PAIRS OF 3	
ITEM PAIRS	COUNT
MSD, ST, PC	7
MSD, PC, PGG	5

< 6 (MINIMUM SUPPORT 30%)		
ITERATION 2		
STEP 5 (LI)	COUNT	SUPPORT
MSD, ST, PC	7	7/20 = 35%

The set of items that are bought most frequently together are the Micro SD Card, Shoot Tripod and PS4 Console.

The next part involved using the frequent itemsets to generate and identify the top 10 association rules that that maximise support and confidence of the rule. To do this I used both iterations of itemsets that were above the minimum support. There were a total of 7 different itemsets that were above the minimum support. After this to find the association between these items we must find out

when one item is bought what is the probability that another item is bought. Looking at each set we are trying to identify if for example MSD is bought, what is the probability that ST and PC are also bought. Confidence = amount of times item is bought divided by the amount of times the pair is bought together. Finally when the confidence for each item is found I then listed the top 10 association rules. The items with the highest confidence of being bought together are the Micro SD Card and PS4 Console at 82%, the PS4 GTA Game and PS4 Console at 80% and the Shoot Tripod and Ps4 Console at 75%. Each of the pairs in the top 10 have over a 50% confidence of being bought together, using these data mining techniques it would make sense and be useful for the store to have these items close together in the store, through data mining techniques a small store can maximise profits.

STEP 4 (LI)	COUNT	SUPPORT
MSD, ST, PC	7	7/20 = 35%
NC, PC	6	6/20 = 30%
MSD, ST	8	8/20 = 40%
MSD, PC	9	9/20 = 45%
MSD, PGG	6	6/20 = 30%
ST, PC	9	9/20 = 45%
PC, PGG	8	8/20 = 40%

ASSOCIATION RULES	CONFIDENCE (ROUNDED)
{MSD} => {ST,PC}	7/11 = 64%
{ST} => {MSD, PC}	7/12 = 58%
{PC}=>{MSD, ST}	7/14 = 50%
{NC}=>{PC}	6/8 = 75%
{PC}=>{NC}	6/14 = 43%
{MSD} => {ST}	8/11 = 73%
{ST} => {MSD}	8/12 = 67%
{MSD}=>{PC}	9/11 = 82%
{PC}=>{MSD}	9/14 = 64%
{MSD}=>{PGG}	6/11 = 55%
{PGG}=>{MSD}	6/10 = 60%
{ST}=>{PC}	9/12= 75%
{PC}=>{ST}	9/14 = 64%
{PC}=>{PGG}	8/14 = 57%
{PGG}=>{PC}	8/10 = 80%

TOP 10	%CONFIDENCE
{MSD}=>{PC}	82%
{PGG}=>{PC}	80%
{ST}=>{PC}	75%
{NC}=>{PC}	75%
{MSD} => {ST}	73%
{ST} => {MSD}	67%
{MSD} => {ST,PC}	64%
{PC}=>{MSD}	64%
{PC}=>{ST}	64%
{ST} => {MSD, PC}	58%

References

ZeroR. 2018. ZeroR. [ONLINE] Available at: http://chem-eng.utoronto.ca/~datamining/dmc/zeror.htm. [Accessed 23 April 2018].

Machine Learning Mastery. 2018. How to Use Ensemble Machine Learning Algorithms in Weka. [ONLINE] Available at: https://machinelearningmastery.com/use-ensemble-machine-learning-algorithms-weka/. [Accessed 23 April 2018].

How to interpret weka classification? - Stack Overflow. 2018. How to interpret weka classification? - Stack Overflow. [ONLINE] Available at: https://stackoverflow.com/questions/2903933/how-to-interpret-weka-classification. [Accessed 23 April 2018].