COMP5009 Data Mining

Predictive Analytics Project

Unit Code : COMP5009

Name : Syed Muhammad Ahmed Zaidi

Student ID: 20972008

Table of Contents

SUMMARY

The assignment aims to teach how data classification techniques can be implemented on real-life data to predict answers for test data. By using the techniques taught in the class, this assignment gave students a practicing field to apply machine learning algorithms to a sqlite dataset that allowed the prediction of test data.

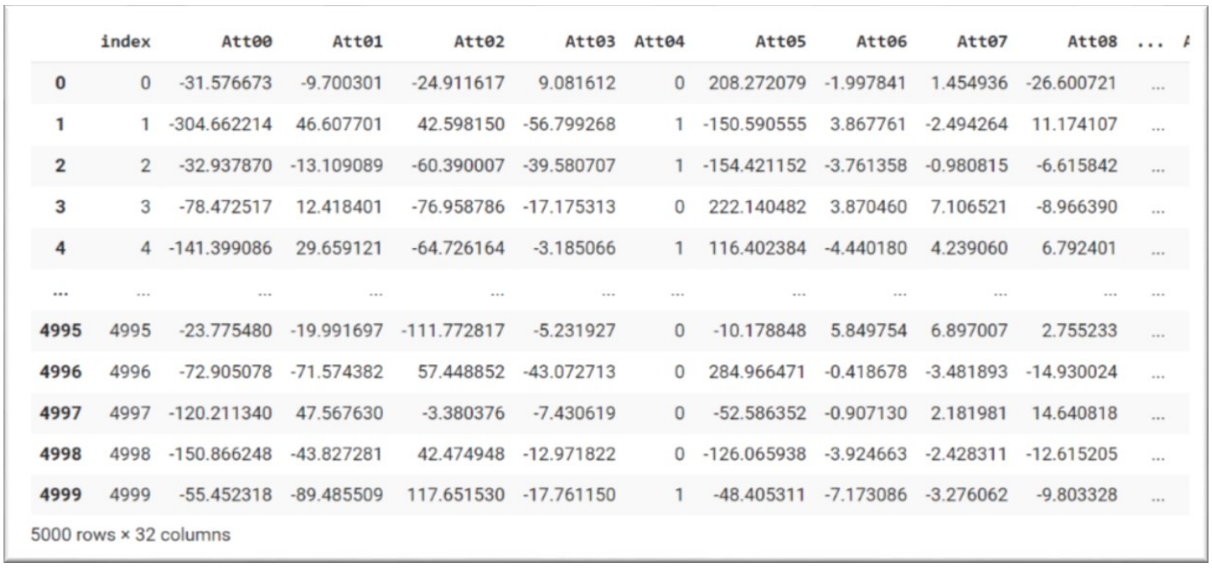
Within the assignment, there were multiple steps that had to be taken to reach the intended results. This starts with data exploration as the data first needs to be understood to know which attributes are directing what type of data. Having all columns with a similar name did add difficulty but taught lessons on how any dataset with any information can be trained without having domain knowledge about it. Once it was explored, the next step was to prepare it, which is the most important part of any project and also one of the most time-consuming (Pyle, 1999). This is done by finding all the missing values, making rules on how to treat them, and also search for duplicates and dealing with them. It also included learning of how we should standardize the data to give better results, dealing with different data types and ways to convert them into intended types for example, the process of binarization.

After the preparation, the steps of analysis are carried out. This involved the process of balanceing the class to give equal weightage allowing improved results of prediction. It also requires techniques to reduce dimensionality, which would reduce noise and improve the prediction power of the models. The assignment required us to use three different machine learning algorithms with the names K-Nearest Neighbors (KNN), Decision Tree Classifier, and Naive Bayes Classifier so that we can understand how hyperparameters can be selected and can also be tuned to find the best scores for any model. Finally, predictions were made to test data using train data with a par percentage of accuracy.

METHODOLOGY

DATA EXPLORATION

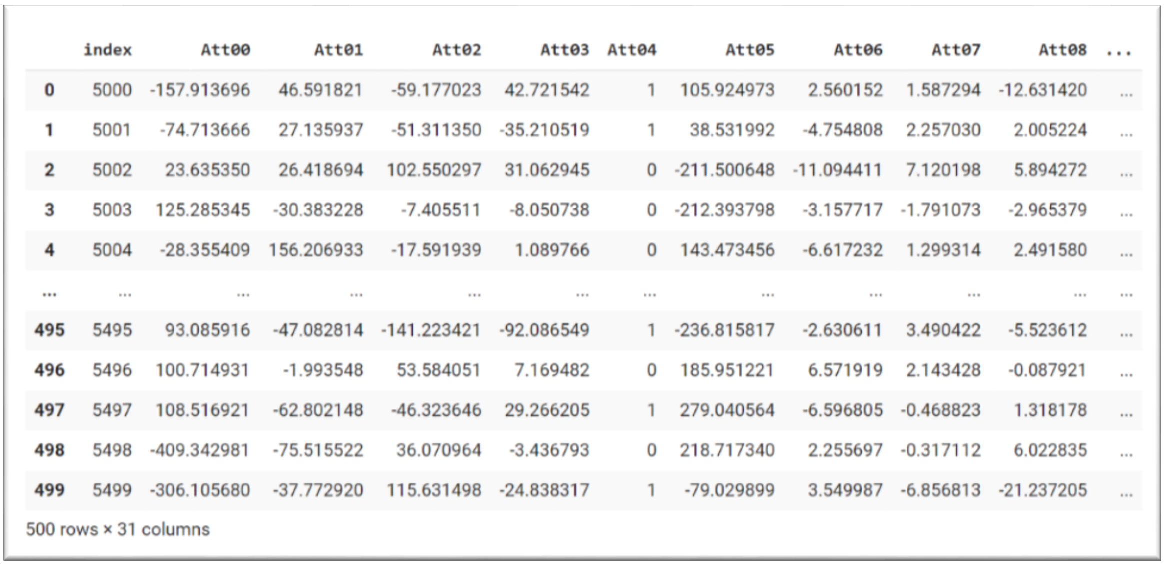
The dataset provided on which to run the prediction model was given to us in the form of a sqlite database. It was imported into Google Colab using Wget and providing it with a GitHub link where the assignment is stored. The code is created in a way that the provided file "Assignment2023.sqlite" would have to be imported into the platform every time it is intended to run the code from scratch. This is done to ensure consistent results and no errors are faced during runtime, even when assessors are running on a different machine. The data was then divided into two tables with the name of train_df and the other as test_df. Both tables had 32 Columns in total; however, train_df has 5000 rows while test_df has 500. Each of them had the last column as "class". For train_df, the column was filled with three classes ranging from 0,1,2, and for train_df, it was not filled as the intention of this assignment was to predict this. Below are screenshots of the data for train_df and test_df



	index	Att00	Att01	Att02	Att03	Att04	Att05	Att06	Att07	Att08	...
0	0	-31.576673	-9.700301	-24.911617	9.081612	0	208.272079	-1.997841	1.454936	-26.600721	...
1	1	-304.662214	46.607701	42.598150	-56.799268	1	-150.590555	3.867761	-2.494264	11.174107	...
2	2	-32.937870	-13.109089	-60.390007	-39.580707	1	-154.421152	-3.761358	-0.980815	-6.615842	...
3	3	-78.472517	12.418401	-76.958786	-17.175313	0	222.140482	3.870460	7.106521	-8.966390	...
4	4	-141.399086	29.659121	-64.726164	-3.185066	1	116.402384	-4.440180	4.239060	6.792401	...
...	...	...	...	...	...	...	...	...	...	...	...
4995	4995	-23.775480	-19.991697	-111.772817	-5.231927	0	-10.178848	5.849754	6.897007	2.755233	...
4996	4996	-72.905078	-71.574382	57.448852	-43.072713	0	284.966471	-0.418678	-3.481893	-14.930024	...
4997	4997	-120.211340	47.567630	-3.380376	-7.430619	0	-52.586352	-0.907130	2.181981	14.640818	...
4998	4998	-150.866248	-43.827281	42.474948	-12.971822	0	-126.065938	-3.924663	-2.428311	-12.615205	...
4999	4999	-55.452318	-89.485509	117.651530	-17.761150	1	-48.405311	-7.173086	-3.276062	-9.803328	...

5000 rows × 32 columns

Figure 1 : train_df

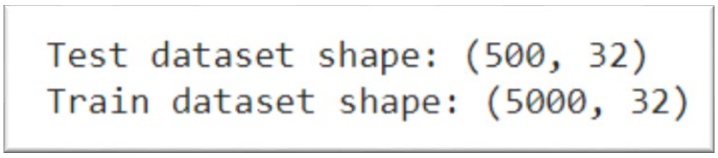


	index	Att00	Att01	Att02	Att03	Att04	Att05	Att06	Att07	Att08	...
0	5000	-157.913696	46.591821	-59.177023	42.721542	1	105.924973	2.560152	1.587294	-12.631420	...
1	5001	-74.713666	27.135937	-51.311350	-35.210519	1	38.531992	-4.754808	2.257030	2.005224	...
2	5002	23.635350	26.418694	102.550297	31.062945	0	-211.500648	-11.094411	7.120198	5.894272	...
3	5003	125.285345	-30.383228	-7.405511	-8.050738	0	-212.393798	-3.157717	-1.791073	-2.965379	...
4	5004	-28.355409	156.206933	-17.591939	1.089766	0	143.473456	-6.617232	1.299314	2.491580	...
...	...	...	...	...	...	...	...	...	...	...	...
495	5495	93.085916	-47.082814	-141.223421	-92.086549	1	-236.815817	-2.630611	3.490422	-5.523612	...
496	5496	100.714931	-1.993548	53.584051	7.169482	0	185.951221	6.571919	2.143428	-0.087921	...
497	5497	108.516921	-62.802148	-46.323646	29.266205	1	279.040564	-6.596805	-0.468823	1.318178	...
498	5498	-409.342981	-75.515522	36.070964	-3.436793	0	218.717340	2.255697	-0.317112	6.022835	...
499	5499	-306.105680	-37.772920	115.631498	-24.838317	1	-79.029899	3.549987	-6.856813	-21.237205	...

500 rows × 31 columns

Figure 2 : test_df

Once the tables were put in their respective variables, the method of shape was used to confirm the dimensional structure of both tables. As mentioned earlier, each of them had 32 attributes, while the rows were 5000 for train_df and 500 for test_df. Another screenshot of the result is provided below:



```
Test dataset shape: (500, 32)
Train dataset shape: (5000, 32)
```

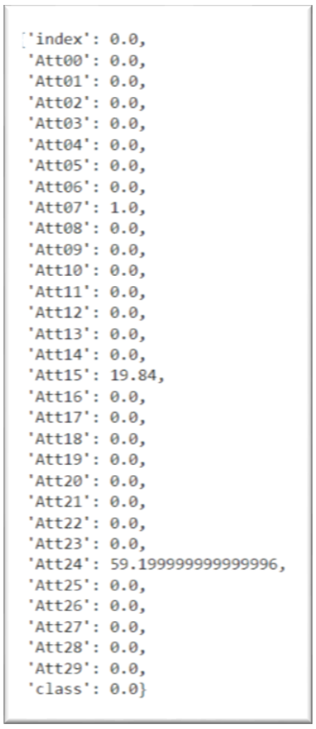
figure 3 : Shape of Datsets

Data exploration was further continued by understanding what the data offers. All column names did not intend to give out any information about the data as all of them were named after the index of the attributes. The only different column was the last one, which was class it was the intended prediction we were required to make using the knowledge learned within the unit. I further used the “describe” method to understand the data from a statistical point of view, giving an overall view of the count, mean, standard deviation, min, max and quartiles. It gave some interesting insights about how the data was structured and also gave an overview of which column of data is presenting what kind of data. A screenshot is attached below.

DATA PREPARATION

Missing Data

Once the data was explored from an aerial view, the process to prepare it for further analysis started. “Preparing data is an important and critical step in modeling for complex data analysis” (Yu et al., 2005). The first step[I took was to identify the missing values within the data. The approach I took was to see each column at a time to find out the percentage of missing values within it. For this purpose, I created a function with the name of missing, which looped each column and calculated the total missing values compared to its total. From this, the results showed a list of all of them and their respective percentages. All had 0 missing values except three columns. I then tried finding out if these three have more than 50% of the values that are missing. This is because I did not want to take that column into my prediction model as it will be adding an extra dimension, which would not be helping with the prediction power because of it being unknown for the majority of data points within. Hence, after finding, ATT24 had around 59% of the values as known, it was dropped from both train and test datasets. Below is the screenshot of the results.



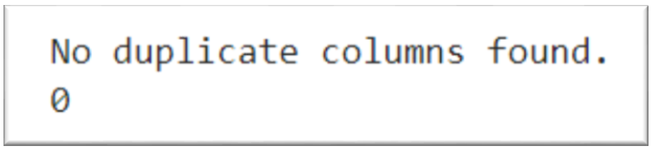
```
{'index': 0.0,
 'Att00': 0.0,
 'Att01': 0.0,
 'Att02': 0.0,
 'Att03': 0.0,
 'Att04': 0.0,
 'Att05': 0.0,
 'Att06': 0.0,
 'Att07': 1.0,
 'Att08': 0.0,
 'Att09': 0.0,
 'Att10': 0.0,
 'Att11': 0.0,
 'Att12': 0.0,
 'Att13': 0.0,
 'Att14': 0.0,
 'Att15': 19.84,
 'Att16': 0.0,
 'Att17': 0.0,
 'Att18': 0.0,
 'Att19': 0.0,
 'Att20': 0.0,
 'Att21': 0.0,
 'Att22': 0.0,
 'Att23': 0.0,
 'Att24': 59.199999999999996,
 'Att25': 0.0,
 'Att26': 0.0,
 'Att27': 0.0,
 'Att28': 0.0,
 'Att29': 0.0,
 'class': 0.0}
```

Figure 4 : Percentage of Missing Values

Once columns with excessive missing values were dealt with, I moved my attention toward the columns that had missing values but was less than 50%. These columns are identified to be ATT07 and ATT15. As their missing values were not too much that would require them to be dropped off, their availability within the data may add value to the overall model. There I decided to fill the missing values with a method called interpolation. I used the linear method of interpolation to solve the missing values. What this method does is to find the neighboring points of the data and then use them to find what the linear relationship would be to get to know the missing values. This method is simple yet very effective, as the values did allow great results in prediction, which will be discussed further on.

Duplicate Data (Rows and Columns)

As the data was now filled up with columns deleted and gaps filled, I moved on to check if there is any duplication in the data. So the approach I took was to consider rows with rows and columns against columns. This is because the database is quite large and finding just any duplicate value in any part of the dataset would not mean anything. Therefore, first, I checked whether any of the columns are identical to other columns. This was done by running a loop that takes into consideration all the values of a column as a whole and then compares it with other columns to find if there is any match. From the results, there was no duplicate columns in either train_df nor test_df. The printed results are shown below.



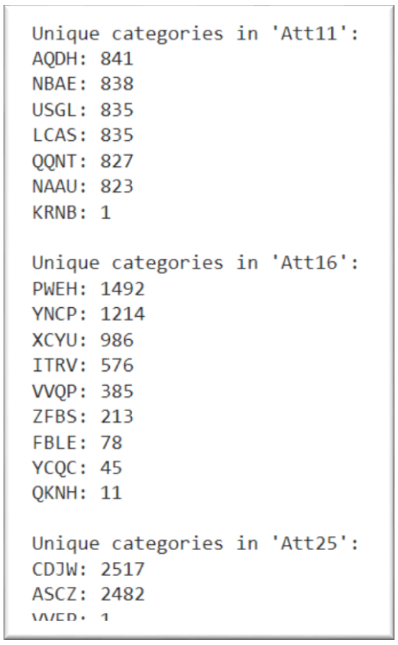
```
No duplicate columns found.
0
```

Figure 5 : Result of Duplicates

Then I moved on to the rows. Where I used the "duplicated" method to find if there is any duplicate in rows. There was none found in both tables. Just to verify I used other approaches as well by doing it according to the index number but even there was nothing that came up. Hence, by this analysis, it was confirmed that this dataset had no duplicates at all. Not in columns and not in rows in both tables.

Data Types

Next I tried understanding the types of data included within the tables. As the attributes had no name, it was very hard to distinguish what each of them represented. This is why relying on the data types was essential. I used the built-in method of dtypes to find which attribute is under which type. My ultimate is was to find all the attributes that are in a categorical nature so that they can be converted to numerical form for analysis purpose. This process of binarization can be done by creating dummies of each column that is the category. The way to find this was by getting to know which attributes have the dtype of object. As majority were either floats or integers, only three came out to be objects. Att11, Att16 and Att25 which were put into a variable called categorical columns. To get more information about them I used the nunique() method to find how many unique categories are within each column. Then I went a step forward and for these particular attributes tried to find the names of all categories included within. A screenshot of the result is attached below.



```
Unique categories in 'Att11':
AQDH: 841
NBAE: 838
USGL: 835
LCAS: 835
QQNT: 827
NAAU: 823
KRNB: 1

Unique categories in 'Att16':
PWEH: 1492
YNCP: 1214
XCYU: 986
ITRV: 576
VVQP: 385
ZFBS: 213
FBLE: 78
YCQC: 45
QKNH: 11

Unique categories in 'Att25':
CDJW: 2517
ASCZ: 2482
```

Figure 6: Categorical Variables

Once found, I used the function from Pandas called get_dummies to make all of the categories within these attributes to have their own columns in a binary form. Any yes would be regarded as 1.0 and No would be 0.0. This process of binarization of categorical variables was done on both train and test data. After completion, the old attributes were dropped off so they do not add noise in the analysis.

Scaling / Standardization

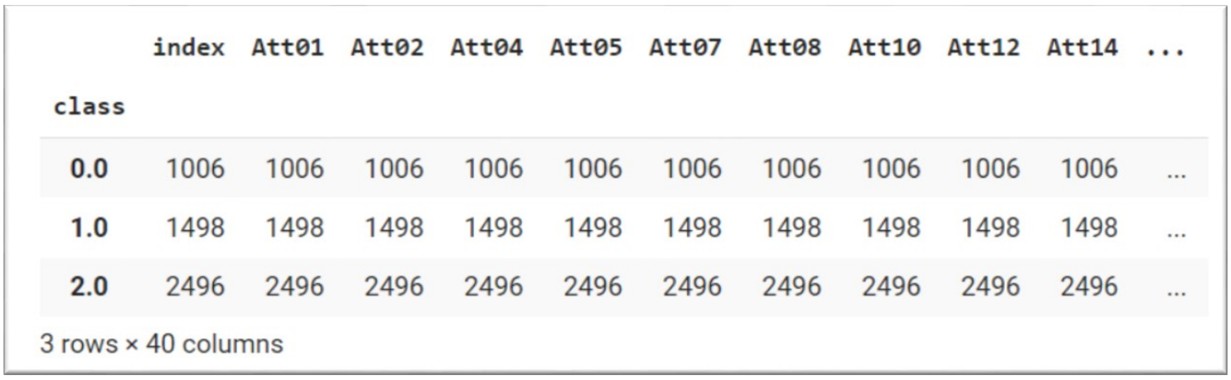
The process of standardization was delayed until the whole train_df was divided into a variable called X and Y in which contain 75% of the values while the other contains the rest.

This was to be needed when running a validation test once the model was complete. For standardization, I considered all options and ended up choosing the Z score method. This was an essential thing to do as the data was having many attributes, out of which any of them could be dominating the other, causing issues to how data is analyzed. By applying this method, it ensures that all features have a mean of 0 and a standard deviation of 1, preventing any domination to take place. As I plan on running algorithms like KNN which find results based on distance, the standardization will allow improved results. The results are very hard to interpret without the headers and hence having this method of standardization would also help to deal with the outliers which may cause the results to skew in a different direction. From the last step of binarization, the dataset was left with many columns that were floats. As there was no use of running standardization on binary datatypes, I tried distinguishing all non-binary attributes from floats. This was done by converting all of them into integers and picking all the floats to be stored in a variable called columns_to_convert. Once this was done, the StandardScaler was run on these attributes to make them scalled for further analysis.

DATA CLASSIFICATION

Data Imbalance

While exploring the data, I came across the class variable that showed an imbalance in terms of the categories it contained. To research further, I tried grouping the whole train_df by the attribute of class and used the count method to find the quantities. My research was confirmed as the results below show.



	index	Att01	Att02	Att04	Att05	Att07	Att08	Att10	Att12	Att14	...
class											
0.0	1006	1006	1006	1006	1006	1006	1006	1006	1006	1006	...
1.0	1498	1498	1498	1498	1498	1498	1498	1498	1498	1498	...
2.0	2496	2496	2496	2496	2496	2496	2496	2496	2496	2496	...

3 rows × 40 columns

Figure 7: Data Imbalance

In order to fully confirm it I also further created a pie chart with percentages to represent the imbalance that exists in the class attribute. From the pie chart below, we can see that the most occurring category was 2.0 which was followed by 1.0 and then 0. This class imbalance is would have resulted in a biased model. Since there are fewer instances of the minority class, the model would not have learned enough about it, and its predictive power on the minority class would have been poor.

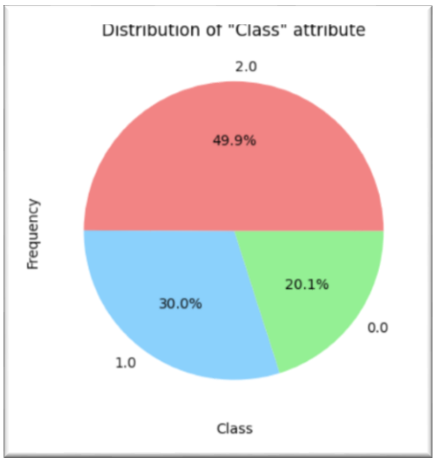
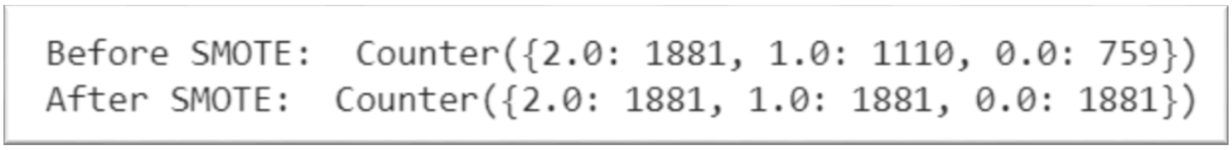


Figure 8: Data Imbalance Pie Chart

Due to this reason, I decided to balance the data by using a technique called SMOTE (Synthetic Minority Over-sampling Technique). SMOTE is one of the over-sampling methods which addresses the issue of imbalanced data (Han, Wang, and Mao 2005). This is a widely used method of balancing classification problems. It generates the samples for the minority class so that it equals the most occurred categorically. This equal percentage of representation between all categories of class allows the elimination of biasness and gives an authentic result taking into account all categories at equal terms. Once the method was called, the results were stored in different variables so that they could be tested separately from the normal way of using classification methods. A screenshot of the results is attached below, showing how the value of 2.0 was 1881 while others were much lower. However, once SMOTE was called there were new figures for all three categories showing an equal representation.

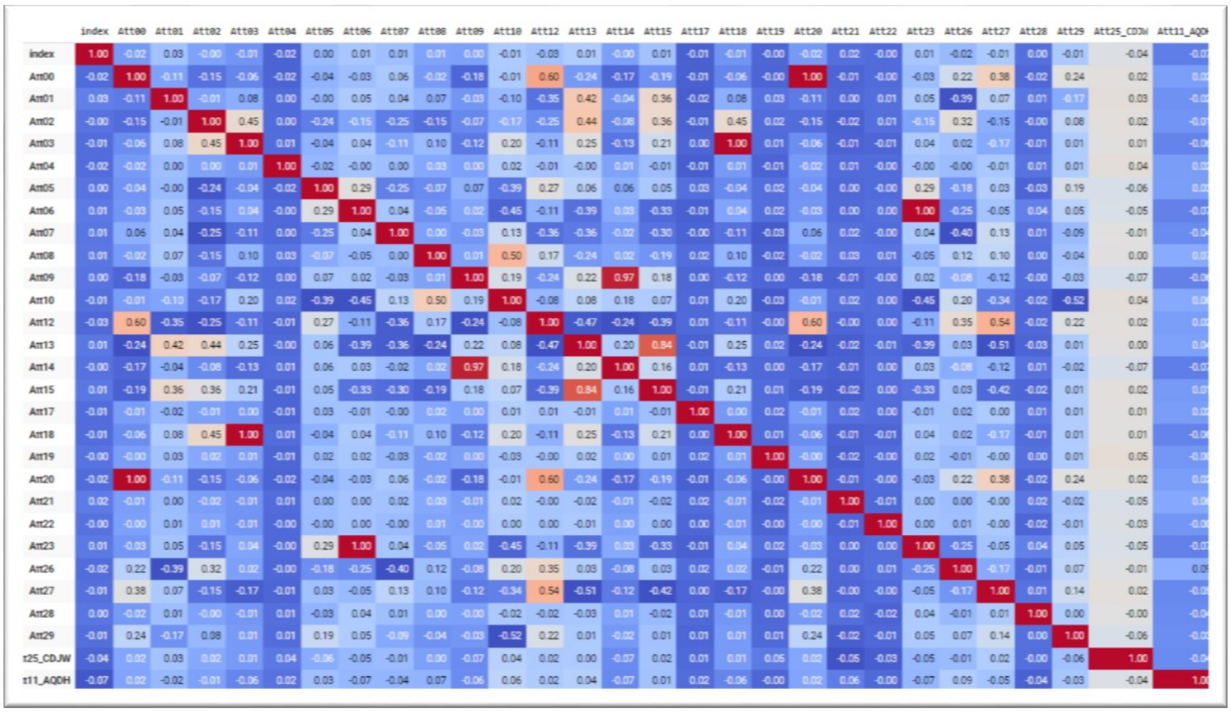


```
Before SMOTE:  Counter({2.0: 1881, 1.0: 1110, 0.0: 759})
After SMOTE:  Counter({2.0: 1881, 1.0: 1881, 0.0: 1881})
```

Figure 9: Results after Smote

Dimensionality

Dimensionaliliuty is an important concept for machine learning and predictive analytics. Its reduction methods are closely related to both feature selection and clustering. (Aggarwal & Aggarwal, 2015). If there are too many dimensions within a data, it may result in too much noise or may even hinder the results of the prediction power. Therefore I explored the data deeply to find irrelevant attributes that can be removed. One method I used which was very effective, was done a bit earlier. The part when binarization was completed, I ran a correlation of all the attributes with each other to find their respective correlation values. To my surprise, there were a few who had very high values. A screenshot of the results is shown below. Due to the size Iam unable to attach the full figure but the screenshot does show the attributes that perfectly correlated.



	index	Att00	Att01	Att02	Att03	Att04	Att05	Att06	Att07	Att08	Att09	Att10	Att12	Att13	Att14	Att15	Att17	Att18	Att19	Att20	Att21	Att22	Att23	Att26	Att27	Att28	Att29	Att25_CDJW	Att11_AQD
index	1.00	-0.02	0.03	-0.00	-0.01	-0.02	0.00	0.01	0.01	0.01	0.00	-0.01	-0.03	0.01	-0.00	0.01	-0.01	-0.01	-0.00	-0.02	0.02	-0.00	0.01	-0.02	-0.01	0.00	-0.01	-0.04	-0.0
Att00	-0.02	1.00	-0.11	-0.15	-0.06	-0.02	-0.04	-0.03	0.06	-0.02	-0.18	-0.01	0.60	-0.24	-0.17	-0.19	-0.01	-0.06	-0.00	1.00	-0.01	-0.00	-0.03	0.22	0.38	-0.02	0.24	0.02	0.0
Att01	0.03	-0.11	1.00	-0.01	0.08	0.00	-0.00	0.05	0.04	0.07	-0.03	-0.10	-0.35	0.42	-0.04	0.36	-0.02	0.08	0.03	-0.11	0.00	0.01	0.05	-0.39	0.07	0.01	-0.17	0.03	-0.0
Att02	-0.00	-0.15	-0.01	1.00	0.45	0.00	-0.24	-0.15	-0.25	-0.15	-0.07	-0.17	-0.25	0.44	-0.08	0.36	-0.01	0.45	0.02	-0.15	-0.02	0.01	-0.15	0.32	-0.15	-0.00	0.08	0.02	-0.0
Att03	-0.01	-0.06	0.08	0.45	1.00	0.01	-0.04	0.04	-0.11	0.10	-0.12	0.20	-0.11	0.25	-0.13	0.21	0.00	1.00	0.01	-0.06	-0.01	-0.01	0.04	0.02	-0.17	-0.01	0.01	0.01	-0.0
Att04	-0.02	-0.02	0.00	0.00	0.01	1.00	-0.02	-0.00	0.00	0.03	0.00	0.02	-0.01	-0.00	0.01	-0.01	-0.01	0.01	-0.01	-0.02	0.01	-0.00	-0.00	-0.00	-0.01	0.01	0.01	0.04	0.0
Att05	0.00	-0.04	-0.00	-0.24	-0.04	-0.02	1.00	0.29	-0.25	-0.07	0.07	-0.39	0.27	0.06	0.06	0.05	0.03	-0.04	0.02	-0.04	0.00	-0.00	0.29	-0.18	0.03	-0.03	0.19	-0.06	0.0
Att06	0.01	-0.03	0.05	-0.15	0.04	-0.00	0.29	1.00	0.04	-0.05	0.02	-0.45	-0.11	-0.39	0.03	-0.33	-0.01	0.04	0.02	-0.03	0.00	0.00	1.00	-0.25	-0.05	0.04	0.05	-0.05	-0.0
Att07	0.01	0.06	0.04	-0.25	-0.11	0.00	-0.25	0.04	1.00	0.00	-0.03	0.13	-0.36	-0.36	-0.02	-0.30	-0.00	-0.11	-0.03	0.06	0.02	-0.00	0.04	-0.40	0.13	0.01	-0.09	-0.01	-0.0
Att08	0.01	-0.02	0.07	-0.15	0.10	0.03	-0.07	-0.05	0.00	1.00	0.01	0.50	0.17	-0.24	0.02	-0.19	0.02	0.10	-0.02	-0.02	0.03	0.01	-0.05	0.12	0.10	0.00	-0.04	0.00	0.0
Att09	0.00	-0.18	-0.03	-0.07	-0.12	0.00	0.07	0.02	-0.03	0.01	1.00	0.19	-0.24	0.22	0.97	0.18	0.00	-0.12	0.00	-0.18	-0.01	-0.00	0.02	-0.08	-0.12	-0.00	-0.03	-0.07	-0.0
Att10	-0.01	-0.01	-0.10	-0.17	0.20	0.02	-0.39	-0.45	0.13	0.50	0.19	1.00	-0.08	0.08	0.18	0.07	0.01	0.20	-0.03	-0.01	0.02	0.00	-0.45	0.20	-0.34	-0.02	-0.52	0.04	0.0
Att12	-0.03	0.60	-0.35	-0.25	-0.11	-0.01	0.27	-0.11	-0.36	0.17	-0.24	-0.08	1.00	-0.47	-0.24	-0.39	0.01	-0.11	-0.00	0.60	-0.00	0.00	-0.11	0.35	0.54	-0.02	0.22	0.02	0.0
Att13	0.01	-0.24	0.42	0.44	0.25	-0.00	0.06	-0.39	-0.36	-0.24	0.22	0.08	-0.47	1.00	0.20	0.84	-0.01	0.25	0.02	-0.24	-0.02	-0.01	-0.39	0.03	-0.51	-0.03	0.01	0.00	0.0
Att14	-0.00	-0.17	-0.04	-0.08	-0.13	0.01	0.06	0.03	-0.02	0.02	0.97	0.18	-0.24	0.20	1.00	0.16	0.01	-0.13	0.00	-0.17	-0.01	0.00	0.03	-0.08	-0.12	0.01	-0.02	-0.07	-0.0
Att15	0.01	-0.19	0.36	0.36	0.21	-0.01	0.05	-0.33	-0.30	-0.19	0.18	0.07	-0.39	0.84	0.16	1.00	-0.01	0.21	0.01	-0.19	-0.02	0.00	-0.33	0.03	-0.42	-0.02	0.01	0.02	0.0
Att17	-0.01	-0.01	-0.02	-0.01	0.00	-0.01	0.03	-0.01	-0.00	0.02	0.00	0.01	0.01	-0.01	0.01	-0.01	1.00	0.00	0.02	-0.01	0.02	0.00	-0.01	0.02	0.00	0.01	0.01	0.01	0.0
Att18	-0.01	-0.06	0.08	0.45	1.00	0.01	-0.04	0.04	-0.11	0.10	-0.12	0.20	-0.11	0.25	-0.13	0.21	0.00	1.00	0.01	-0.06	-0.01	-0.01	0.04	0.02	-0.17	-0.01	0.01	0.01	-0.0
Att19	-0.00	-0.00	0.03	0.02	0.01	-0.01	0.02	0.02	-0.03	-0.02	0.00	-0.03	-0.00	0.02	0.00	0.01	0.02	0.01	1.00	-0.00	-0.02	-0.00	0.02	-0.01	-0.00	0.00	0.01	0.05	-0.0
Att20	-0.02	1.00	-0.11	-0.15	-0.06	-0.02	-0.04	-0.03	0.06	-0.02	-0.18	-0.01	0.60	-0.24	-0.17	-0.19	-0.01	-0.06	-0.00	1.00	-0.01	-0.00	-0.03	0.22	0.38	-0.02	0.24	0.02	0.0
Att21	0.02	-0.01	0.00	-0.02	-0.01	0.01	0.00	0.00	0.02	0.03	-0.01	0.02	-0.00	-0.02	-0.01	-0.02	0.02	-0.01	-0.02	-0.01	1.00	-0.01	0.00	0.00	-0.00	0.02	-0.02	-0.05	0.0
Att22	-0.00	-0.00	0.01	0.01	-0.01	-0.00	-0.00	0.00	-0.00	0.01	-0.00	0.00	0.00	-0.01	0.00	0.00	0.00	-0.01	-0.00	-0.00	-0.01	1.00	0.00	0.01	-0.00	-0.02	-0.01	-0.03	-0.0
Att23	0.01	-0.03	0.05	-0.15	0.04	-0.00	0.29	1.00	0.04	-0.05	0.02	-0.45	-0.11	-0.39	0.03	-0.33	-0.01	0.04	0.02	-0.03	0.00	0.00	1.00	-0.25	-0.05	0.04	0.05	-0.05	-0.0
Att26	-0.02	0.22	-0.39	0.32	0.02	-0.00	-0.18	-0.25	-0.40	0.12	-0.08	0.20	0.35	0.03	-0.08	0.03	0.02	0.02	-0.01	0.22	0.00	0.01	-0.25	1.00	-0.17	-0.01	0.07	-0.01	0.0
Att27	-0.01	0.38	0.07	-0.15	-0.17	-0.01	0.03	-0.05	0.13	0.10	-0.12	-0.34	0.54	-0.51	-0.12	-0.42	0.00	-0.17	-0.00	0.38	-0.00	-0.00	-0.05	-0.17	1.00	0.01	0.14	0.02	-0.0
Att28	0.00	-0.02	0.01	-0.00	-0.01	0.01	-0.03	0.04	0.01	0.00	-0.00	-0.02	-0.02	-0.03	0.01	-0.02	0.01	-0.01	0.00	-0.02	0.02	-0.02	0.04	-0.01	0.01	1.00	0.00	-0.00	-0.0
Att29	-0.01	0.24	-0.17	0.08	0.01	0.01	0.19	0.05	-0.09	-0.04	-0.03	-0.52	0.22	0.01	-0.02	0.01	0.01	0.01	0.01	0.24	-0.02	-0.01	0.05	0.07	0.14	0.00	1.00	-0.06	-0.0
t25_CDJW	-0.04	0.02	0.03	0.02	0.01	0.04	-0.06	-0.05	-0.01	0.00	-0.07	0.04	0.02	0.00	-0.07	0.02	0.01	0.01	0.05	0.02	-0.05	-0.03	-0.05	-0.01	0.02	-0.00	-0.06	1.00	-0.0
t11_AQDH	-0.07	0.02	-0.02	-0.01	-0.06	0.02	0.03	-0.07	-0.04	0.07	-0.06	0.06	0.02	0.04	-0.07	0.01	0.02	-0.06	-0.00	0.02	0.06	-0.00	-0.07	0.09	-0.05	-0.04	-0.03	-0.04	1.0

Figure 10: Correlation Table

The rule I set was that all attributes having a correlation of more than 80% whether positive or negative will be deleted. This is because having all of them within the model will cause multicollinearity which results in increased variability in coeffcients and also inflated standard errors. By dropping any of them would reduce noise and also make the model much more stable. Due to this reason, I picked 5 of them which were positively correlated with a value of more than 80% and 1 which was perfectly negatively correlated. The names of these attributes include Att00, Att03, Att06, Att09, Att13, and Att25_ASCZ. These were dropped from both the train table as well as the test table.

Hyperparameters Tuning

As the requirement of the assignment was to use different models of machine learning to predict the results, I have used different hyperparameter selections in methods. For K-Nearest Neighbouts(KNN) the parameters selected was n_neighbors and weights. n_neighbors is a way to represent the number of nearest neighbors that will be considered while making predictions. This is done as it controls the clash between the bias and variance in the model. It allows the smaller values within the data to be more flexible while the larger values allow smoother decision boundaries which reduces the chances of overfitting. On the other hand for the same model, I used weights as the other hyperparameter which can go two ways. One is uniform which means it treats all its neighbors as equals, while the other is distance, which means that it will give weightage to neighbors on the basis of its distance from the data point. Now once these hyperparameters were selected, I want on to the next step of tuning it. For this I used GridSearchCV which systematically explores combinations of hyperparameter values. For n_neighbors I tried with the options of 1,3,7 and 11 and for weight both uniform and distance to see which ones gives the best results.

The next model used was decision tree. The first hyperparameter selected within this was criterion as it determines the quality of a split. For this gini and entropy were adopted. Gini measures the impurity based on the Gini Index while the entropy method uses the information gain. On the other hand, the second hyperparameter selected was the min_samples_split. This controls the minimum number of samples required to split a node. I tried with 3, 13, 15 and 20. Usage of this hyperparameter allows to create balance between model complexity and generalization. In order to tune this, ive used the same GridSeachCV method of tuning. It systematically explores all the combinations and give the results that is best for prediction accuracy.

Validation

For validation, I did not rely on only creating the model on train data and validating it on the test data. I went for a two tier approach in which I also divided the train data as X and Y which would encompass a ratio of 75% and 25% of the train data. The Y would be used in all three models as well to validate the accuracy of the model created. Apart from this, I used three techniques for cross validation. These were taken up from the scikit-learn library and are called Stratified K-Fold, K-fold and shuffle split. The strafied K-fold technique is a way if cross validating data by ensuring that each fold maintains the same class distribution as the original dataset. For this reason I used the n_splits to be equal to 10 which was hyperparameter tuned by GridzSearchCV. Similarly, K-fold was also used with the same number of n_splits and was applied on both models KNN and decision tree. Lastly, the shufflesplit was also put under the same tuning to see which would produce the best results. This a way of randomly shuffling and splitting dataset into multiple train and test pairs to validate the model in depth. Cross validation was applied as the primary validation scheme to evaluate the performance of each classifier. They were then evaluated by GridSearchCV by their scores to give out as accuracy.

N-Models

As required by the assignment, we were asked to predict the class of train dataset by using three differentn machine learning models. Below is a description of each of them and how I implemented them to reach the results.

K-Nearest Neighbors (KNN):

This is a very simple yet effective machine-learning algorithm for classification tasks such as this one. "In k-Nearest Neighbor, a database is searched for the most similar elements to a given query element, with similarity defined by a distance function." (Batista & Silva, 2009). It uses the votes of its neighbors to classify an instance. It assgins the instance to the class that is most common among the nearest neighbors. For this particular algorithm, I used two hyperparameters which included the n_neighbors and the other being weights. For the first one I used the number of neighbors ranging from 1,3,7 and 11. While for the weights, both uniform and distance were kept. The cross validation was done using strafied k fold, k fold and shuffle split. The folds were set to be 10 for all. The tuning was done using GridSearchCV from scikit-Learn which allowed better understanding of accuracy and scores. Lastly once the the results were out. I also used the same algorithm of the data that was balanced by SMOTE to see which one gives a better result for prediction.

Decision Tree Classifer:

The second algorithm used is the Decision Tree Classfier. “A decision tree is a series of questions systematically arranged so that each question queries an attribute (e.g., Outlook) and branches” (Steinbach & Tan, 2009). based on the value of the attribute. In this, the data set is partitioned into subsets based on the values of input features. The aim of this machine learning model is to create a tree structure where leaves represent class labels. Within the code, I implemented the model with two hyperparameters. The first one was criterion which had the options of gini and entropy. The other hyperparameter was min_samples_split, which was assigned the splitting numbers to be 3,13,15 and 20. The cross validation methods were similar to KNN and the splits were also set to 10 which is identical to what was used in KNN. After tuning them using GridSearchCV, the model accuracy and performance was further evaluated using the confusion matrix. Lastly, once the results of the model was achieved, I also tried using the SMOTE data which was balance, to see if the results can be further improved.

Naïve Bayes Classifier:

The third algorithm is quite different from the first two as it is a probabilistic classification algorithm. It assumed the features are conditionally independent and follow a Gaussian distribution based on the theory. The model was trained using the the training dataset and then the performance and accuracy was checked. Confusion matrix was also applied to understand it more deeply. In this model, there was no usage of hyperparameter tuning as there were no hyperparameters included to keep the model simple. SMOTE was also not run on this model.

PREDICTION

From the first algorithm of KNN, I initially used parameter tuning using GridSearchCV to find which hyperparameters are producing the best results. Based on this, the output indicated that the best parameters for KNN are n_neighbors to be 11 and the weights to be ‘distance’. Based on this the best accuracy score came out to be **86.75%**. Then I trained my model (X_train and Y_train) using the best output that was obtained. Then I predicted the results for X_test and it came out to be **86.48%.** Finally, once this was achieved I was able to move forward and try the complete same model on the SMOTE balanced data. The results went down to approximately **81.92%**.

The next algorithm of decision tree was also given multiplier hyperparameters which was then tuned using the GridSearchCv just like KNN. Based on this the best score was achieved when we set the criterion to be entropy and the min_samples_split to be 13. This achieved an accuracy score of **78.67%**. This was above the par line of 75% which is why I trained the data sets using the best parameters and then tried to find the accuracy of this. It came out to be **73.56%**. once this was achieved, I further tried using the balances SMOTE data to see if this could be further improved, however the results were similar to KNN as the accuracy dropped to **71.36%**.

Last algorithm tested was the Naïve Bayes Classifier. This was a simple method as no hyperparameters were adjusted. The model is trained on the X_train and Y_train and then predictions are made on the X_test. Based on this, the accuracy comes out to be **73.2%**. Below is an attached confusion matrix that gives a detailed view of the performance by showing the number of true positives (correctly predicted instances), true negatives (correctly predicted non-instances), false positives (incorrectly predicted instances), and false negatives (incorrectly predicted non-instances) for each class ("0.0," "1.0," and "2.0").

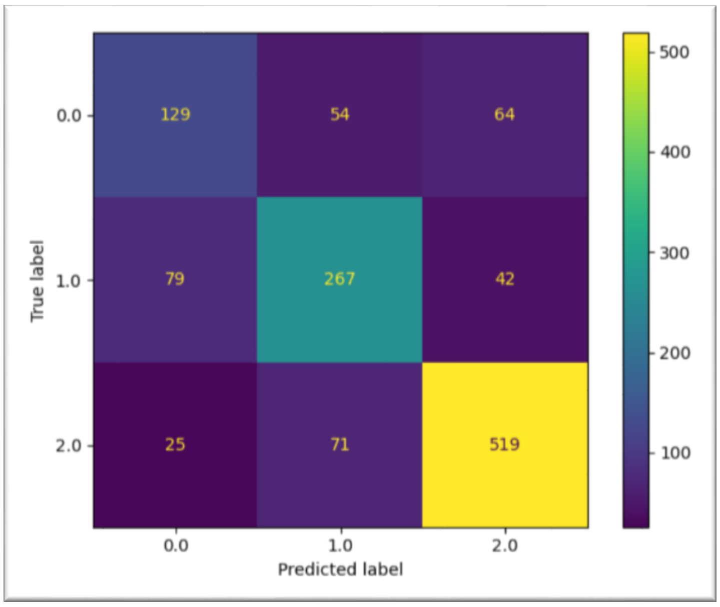


Figure 11 : Confusion Matrix

To give an overview of all the algorithms used and their respective accuracies, I've created a table below for easy understanding.

	Model Accuracy	Prediction Accuracy	Balanced Data Accuracy
K-Nearest Neighbors (KNN)	86.75%	86.48%	81.92%
Decision Tree Classifer	78.67%	73.56%	71.36%
Naïve Bayes Classifier	-	73.2%	-

Figure 12 : Prediction Table

Based on the table above, we were required to select only the two best models and use them to predict the classes for the test data. Hence, Ive chosen KNN and Decision Tree as the best ones out of the three. Based on these two, a new sqlite is created with the name Answers.sqlite which will contain the index along with the Predicition 1 and Prediction 2 for all the 500 rows in the test data.

CONCLUSION

Data mining is a process of nontrivial extraction of implicit, unknown information from data in a database (Romero & Ventura, 2013). In this assignment, we explored the working of three popular classification models which are K-Nearest Neighbors (KNN), Decision Trees, and Naive Bayes. The primary objective was to learn how data is prepared, the intricacies of each of these models, perform hyperparameter tuning and evaluate the results once a prediction is made on a real world dataset.

These findings have practical implications for real-world applications. KNN demonstrated the highest accuracy, making it suitable for scenarios where accuracy is essential. Decision Trees came second as it achievd slightly lower accuracy, offering transparency and interpretability, making them useful in scenarios where model explainability is crucial. Naive Bayes, although simple, provides a baseline for comparison and may be preferred when computational resources are limited.

The models can always be further improved with trial and error. Starting from missing data, by changing what percentage of missing column values to keep and drop may allow changes to the results. Methods of recognizing duplicate values and then changing ways on how to treat them may also effect results. The process of scalling used in this assignment was Z Scoring, however opting to use min-max or any other may also result in different output. Lastly, Changes in the hyperparameter selection may improve or worsen the accuracy results. There the results are subjective to the choices made. As the above-mentioned scores were above the par 75% accuracy, therefore were chosen as final.

REFERENCES

Aggarwal, C. C., & Aggarwal, C. C. (2015). *Data classification*. Springer.

Batista, G., & Silva, D. F. (2009). How k-nearest neighbor parameters affect its performance. Argentine symposium on artificial intelligence,

Han, Hui, Wen-Yuan Wang, and Bing-Huan Mao. 2005. "Borderline-SMOTE: A New OverSampling Method in Imbalanced Data Sets Learning." Advances in Intelligent Computing, Berlin, Heidelberg, 2005

Pyle, D. (1999). *Data preparation for data mining*. morgan kaufmann.

Romero, C., & Ventura, S. (2013). Data mining in education. *Wiley Interdisciplinary Reviews: Data mining and knowledge discovery*, *3*(1), 12-27.

Steinbach, M., & Tan, P.-N. (2009). kNN: k-nearest neighbors. *The top ten algorithms in data mining*, 151-162.

Yu, L., Wang, S., & Lai, K. K. (2005). An integrated data preparation scheme for neural network data analysis. *IEEE Transactions on Knowledge and Data Engineering*, *18*(2), 217-230.