CITS3401 - Data Warehousing - Project 2: Pattern Discovery and Building Predictive Models of Mobile Phone Price

Max Matthews, Frinze Erin Lapuz (22711649)

May 16, 2021

1 Pattern Discovery and Building Predictive Models

1.0.1 PROJECT 2 - CITS 3401

Authors: Frinze Erin Lapuz (22711649) and Max Matthews (21506225)Author: Max Matthews and Frinze Erin Lapuz (22711649)

2 Introduction

For this project, we would like to use the mobile price classification dataset as the source of data. The target of this project is to predict whether the price of a mobile phone is high or not.

2.0.1 Tasks and Scope

1) Data cleaning and analysis

- Read through the table and the table column descriptions. Understand the meaning of each column in the table.
- Distinguish the type of each attribute (e.g., nominal/categorical, numerical). You may need to discretise some attributes, when completing Task 2, 3 or 4.
- Determine whether an attribute is relevant to your target variable. You may remove some attributes if they are not helpful for Task 2, 3, or 4. You might create separate data files for Task 2, 3 and 4.
- Identify inconsistent data and take actions using the knowledge you have learnt in this unit.

2) Association rule mining

- Select a subset of the attributes (or all the attributes) to mine interesting patterns. To rank the degree of interesting of the rules extracted, use support, confidence and lift.
- Explain the top k rules (according to lift or confidence) that have the "price_category" on the right-hand-side, where k >= 1.
- Explain the meaning of the k rules in plain English.
- Given the rules, what recommendation will you give to a company willing to design a high price mobile phone (e.g., should the mobile phone equipped with bluetooth)?

3) Classification

- Use the "price_category" as the target variable and train two classifiers based on different machine learning algorithms (e.g. classifier 1 based on a decision tree; classifier 2 based on SVMs).
- Evaluate the classifiers based on some evaluation metrics (e.g., accuracy). You may use 10-fold cross-validation for the evaluation.

4) Clustering

- Run a clustering algorithm of your choice and explain how the results can be interpreted with respect to the target variable.

5) Data reduction

- Perform numerosity reduction and perform attribute reduction.
- Train the two classifiers in Task 3 on the reduced data.
- Answer the question: "Does data reduction improve the quality of the classifiers"?

6) Attribute selection

- Select the top-10 most important attributes manually based on your understanding of the problem; select the top-10 most important attributes based on Information Gain.
- Which attribute selection method is better and why?

Marking Scheme

```
[5 marks]   Explain the data processing operations (e.g., remove some attributes and action on
[5 marks]   Explain and interpret the top k association rules mined; based on the association r
            to design a high price mobile phone.
[5 marks]   Explain how you train the classifiers and your evaluation results.
[5 marks]   Clustering and interpretation of the clustering result (with respect to the target
[5 marks]   Explain the data reduction you have performed; compare the classifiers trained on r
[5 marks]   Your answer to Task 6.
```

2.0.2 Tools, Libraries and Packages

Python - Using throughout the project for data cleaning, data processing, and modelling. Weka - Used a discrepency check (to ensure we receive the same values)

Imports

[1]:
```
pip install pandas-profiling[notebook] mlxtend clusteval --user
```

```
Note: you may need to restart the kernel to use updated packages.

'C:\Users\Max' is not recognized as an internal or external command,
operable program or batch file.
```

[102]:
```
# Data analysis, manipulation, and profiling
import pandas as pd
import numpy as np
from pandas_profiling import ProfileReport
```

```
# Visualization
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("darkgrid", {"axes.facecolor": ".9"})

# Association Rule Mining
from mlxtend.frequent_patterns import apriori, association_rules

# Training Setups
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline

# Preprocessings and Attribute Selections
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
# selection of best attributes from by default using f-score https://
 ↪scikit-learn.org/stable/modules/generated/sklearn.feature_selection.
 ↪f_classif.html
from sklearn.feature_selection import SelectKBest, mutual_info_classif,␣
 ↪f_classif

# Classifiers
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.svm import SVC # Support Vector Machine Classifier

RANDOM_STATE = 1 # Used as a seed value
CROSS_VALIDATION_PARTITION = 10
NUMBER_OF_BEST_HYPER_PARAMS_TO_SHOW = 5

# Optimisation
from sklearn.model_selection import GridSearchCV # tuning the model
from sklearn.model_selection import cross_val_score, cross_validate

# Clustering
from clusteval import clusteval
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import KMeans, AgglomerativeClustering
from scipy.cluster import hierarchy
from scipy.cluster.hierarchy import dendrogram
```

3 Data Cleaning and Profiling

There are many different ways to perform data cleaning and profiling. For this process (in Project 2) we will be using an IPython Notebook, for the following reasons:

- The anaylsts are proficient in Python;
- The report can be integrated with code for specific sections of the analysis;
- The processes / procedures are highly repeatable and easily automated using scripts;
- Data exploration and anomaly detection can easily be performed through a variety of visualizations (charts, graphs, tables, etc);

The packages that will be used are built-in the Anaconda package, except for `pandas_profiling` (from https://github.com/pandas-profiling/pandas-profiling, which we used for doing detailed exploratory analysis of data), and `mlxtend` (from https://github.com/rasbt/mlxtend , which we used in our association rule mining). Data profiling is important to measure the quality of data, which in turn assists greately in the determination of data anomalies/inconsistencies, and as such, necessary data transformations. The data profiling reports will be generated with Pandas Profiling. Portions of these reports will be referenced in our project discussion, with the resource/s (data profiling reports) attached as an Appendix.

3.0.1 Profiling Report

With the imported metadata, and raw dataset, we can now populate and generate a `pandas_profiling` report. For reference, see `Raw Data Profiling Report`, in the apendix.

3.1 Initial (raw import data) Interpretation, and Cleaning Strategy

Here, we will walk through the fields presented in the profile of the raw data, and make the appropriate alterations / assumptions / cleaning steps. In the event of changes, we will apply them to the `staging_data` dataframe, and `staging_metadata` dictionary.

`ID` The id field is unique, and it seems to be a good candidate for primary key. Note, we will NOT be using ID for any analysis, as it is an irrelevant attribute.

`battery_power` The battery_power has 1094 distinct counts, approximately half the distinct counts of id. This data is continuous, and as such, we shall discretise for the later association rule mining.

[ ]:
```
# Discretizing battery_power
staging_data = create_discretized_col(staging_data, "battery_power",␣
 ↪is_zero_its_own_category = False)
```

`blue` This ("has bluetooth") field is a categorical data type (*boolean*), expressed (poorly) in string-form with inconsistencies. We observe 10 distinct values that all refer to either the phone has bluetooth (True) or not (False). The inconsistencies must be attributed to an appropriate boolean value, and as such, this field needs to be cleaned.

[ ]:
```
staging_data["has_bluetooth"] = staging_data["blue"].apply(string_to_bool)
staging_metadata.loc["blue","new_name"] = "has_bluetooth"   # Rename metadata
```

---

clock_speed The clock_speed attribute has 26 distinct values for which 413 of the records have value 0.5. Hence the distribution of values in the histogram is right skewed. This data is continuous, and will also need to be discretised if we wish to use it in the association rule mining process.

[ ]:
```
# Discretizing clock_speed
staging_data = create_discretized_col(staging_data, "clock_speed",␣
↪is_zero_its_own_category = False)
```

---

dual_sim The dual_sim attirbute is similar to `blue` attribute wherein it is a categorical datatype (*boolean*) with inconsistencies sppread to 10 string distinct values. Hence, the procedure of cleaning will be similar.

[ ]:
```
staging_data["has_dual_sim"] = staging_data["dual_sim"].apply(string_to_bool)
staging_metadata.loc["dual_sim","new_name"] = "has_dual_sim"  # Rename metadata
```

---

fc The (*Front Camera*) `fc` attribute has 474 (23.7% of the dataset) zero values. We will be interpreting the zero values as "this phone does not have a front camera" , (please see the ***Data Privacy Disclaimer*** below). In addition, this continous data will be discretised, for later (association rule mining) purposes, with the zero values as a seperate category.

[ ]:
```
staging_data["front_cam_resolution"] = staging_data["fc"]
staging_metadata.loc["fc","new_name"] = "front_cam_resolution" # Rename metadata
staging_data = create_discretized_col(staging_data, "front_cam_resolution",␣
↪is_zero_its_own_category = True) # Discretizing front_cam_resolution
```

---

four_g The `four_g` attribute is a categorical (*boolean*) attribute indicated by "yes=1" and "no=0" of having 4G capability. For consistency, this will be converted from numeric (0 or 1) to boolean type.

[ ]:
```
staging_data["has_four_g"] = staging_data["four_g"].apply(int_to_bool)
staging_metadata.loc["four_g","new_name"] = "has_four_g" # Rename metadata
```

---

int_memory The `int_memory` (*Internal Memory*) attribute has 63 distinct values that vary as shown in the *profiling-report* histogram. There are no records of phones having `0 GB` as the storage which would be expected, as phones would require a minimum amount of memory to host their respective operating software/s. This continuous data will be discretised for later use in the association rule mining.

```
[ ]: staging_data = create_discretized_col(staging_data, "int_memory",␣
     ↪is_zero_its_own_category = False) # Discretizing int_memory
```

---

m_dep The m_dep (*Mobile Depth*) attribute has 10 distinct values that vary as shown in the *profiling-report* histogram. The minimum values do not make much sense, however, please refer to the ***Data Privacy Disclaimer*** below. In addition, this continuous data will be discretised for later (association rule mining) purposes.

```
[ ]: staging_data["mobile_depth"] = staging_data["m_dep"]
     staging_metadata.loc["m_dep","new_name"] = "mobile_depth"  # Rename metadata
     staging_data = create_discretized_col(staging_data, "mobile_depth",␣
     ↪is_zero_its_own_category = False) # Discretizing mobile_depth
```

---

mobile_wt The mobile_wt (*Mobile Weight*) attribute has 121 distinct values that varies as shown in the *profiling-report* histogram. The minimum is 80 and the maximum is 200, measured in presumably grams. This continuous data will be discritised for later association rule mining purposes.

```
[ ]: staging_data["mobile_weight"] = staging_data["mobile_wt"]
     staging_metadata.loc["mobile_wt","new_name"] = "mobile_weight" # Rename metadata
     staging_data = create_discretized_col(staging_data, "mobile_weight",␣
     ↪is_zero_its_own_category = False) # Discretizing mobile_weight
```

---

n_cores The n_cores (*Number of Cores*) attribute looks almost uniform in the range 1-8. This is categorical (numerical) data.

```
[ ]: staging_data["number_of_cores"] = staging_data["n_cores"]
     staging_metadata.loc["n_cores","new_name"] = "number_of_cores" # Rename metadata
```

---

pc The pc (*Primary Camera Resolution*) attribute has 21 distinct values with 101 (5.1% of the dataset) with zero values. We will be interpreting the zero values as "this phone does not have a primary camera", (please see the ***Data Privacy Disclaimer*** below). In addition, this continous data will be discretised, for later (association rule mining) purposes, with the zero values as a seperate category.

```
[ ]: staging_data["primary_cam_resolution"] = staging_data["pc"]
     staging_metadata.loc["pc","new_name"] = "primary_cam_resolution"
     staging_data = create_discretized_col(staging_data, "primary_cam_resolution",␣
     ↪is_zero_its_own_category = True) # Discretizing primary_cam_resolution
```

---

px_height The px_height attribute has a right skewed normal distribution with a mean value of 645 pixels and 443.79 standard deviation in the range 0-1960. We will be interpreting the two zero values as *extremely small, but not zero* values, (please see the ***Data Privacy Disclaimer*** below). This notion also applies to the nonsensicle low values. In addition, this continous data will be discretised, for later (association rule mining) purposes.

[ ]:
```
# Discretizing px_height
staging_data = create_discretized_col(staging_data, "px_height",␣
↪is_zero_its_own_category = False)
```

---

px_width The px_width attribute has a varying distribution of values in the range 500 to 1998. This continous data will be discretised, for later (association rule mining) purposes.

[ ]:
```
# Discretizing px_width
staging_data = create_discretized_col(staging_data, "px_width",␣
↪is_zero_its_own_category = False)
```

---

ram The ram attribute has a varying distribution of values in the range 256-3998. This continous data will be discretised, for later (association rule mining) purposes.

[ ]:
```
# Discretizing ram
staging_data = create_discretized_col(staging_data, "ram",␣
↪is_zero_its_own_category = False)
```

---

sc_h The sc_h (*Screen Height*) attributes has 15 distinct values with a range of 5-19. This continous data will be discretised, for later (association rule mining) purposes.

[ ]:
```
staging_data["screen_height"] = staging_data["sc_h"]
staging_metadata.loc["sc_h","new_name"] = "screen_height" # Rename metadata
staging_data = create_discretized_col(staging_data, "screen_height",␣
↪is_zero_its_own_category = False) # Discretizing screen_height
```

---

sc_w The sc_w (*Screen Width*) attribute has 19 distinct values with a range of 0-18 with 180 zero values. These zero values do not make *real world sense*, and so we will be interpreting the zero values as representatives of sensible, lower values (please see the ***Data Privacy Disclaimer*** below). This continous data will be discretised, for later (association rule mining) purposes.

[ ]:
```
staging_data["screen_width"] = staging_data["sc_w"]
staging_metadata.loc["sc_w","new_name"] = "screen_width" # Rename metadata
staging_data = create_discretized_col(staging_data, "screen_width",␣
↪is_zero_its_own_category = False) # Discretizing screen_width
```

---

talk_time The talk_time attribute has 19 distinct values with a range 2-20. This continous data will be discretised, for later (association rule mining) purposes.

[ ]:
```
# Discretizing talk_time
staging_data = create_discretized_col(staging_data, "talk_time",␣
↪is_zero_its_own_category = False)
```

---

three_g The three_g attribute is similar to the blue and dual_sim attribute wherein it is a categorical (*boolean*) datatype, with inconsistencies spread to 10 (string) distinct values. Hence, the cleaning procedure will be similar.

[ ]:
```
staging_data["has_three_g"] = staging_data["three_g"].apply(string_to_bool)
staging_metadata.loc["three_g","new_name"] = "has_three_g" # Rename metadata
```

---

touch_screen The touch_screen attribute is a categorical (*boolean*) attribute ,similar to the representation of four_g. Hence, the cleaning procedure is similar.

[ ]:
```
staging_data["has_touch_screen"] = staging_data["touch_screen"].
↪apply(int_to_bool)
staging_metadata.loc["touch_screen","new_name"] = "has_touch_screen" # Rename␣
↪metadata
```

---

wifi The wifi attirbute is similar to blue,dual_sim and three_g attribute wherein it is a boolean datatype with inconsistencies spread to 10 (string) distinct values. Hence, the procedure of cleaning will be similar.

[ ]:
```
staging_data["has_wifi"] = staging_data["wifi"].apply(string_to_bool)
staging_metadata.loc["wifi","new_name"] = "has_wifi" # Rename metadata
```

---

price_category The price_category attribute is a categorical (*boolean*) attribute, similar to the representation of four_g and touch_screen. Hence, the cleaning procedure is similar.

[ ]:
```
staging_data["is_expensive"] = staging_data["price_category"].apply(int_to_bool)
staging_metadata.loc["price_category","new_name"] = "is_expensive" # Rename␣
↪metadata
```

3.1.1 *Data Privacy Disclaimer*

The data provided (`mobile_prices.csv`) has had some of it's *real world* values altered, for privacy/legal reasons. As such, decisions were made on how to best interpret unusual / nonsensicle `zero` values.

Appropriate zero values (`primary_camera_resolution`, `front_camera_resolution`) Zero values for these attributes can be observed as the phone not having the attribute. This makes sense, as not all phones have a front camera, or primary camera.

Inappropriate zero values (`px_height`, `screen_width`) Zero values for these attributes can be observed as **values altered for privacy reasons**, and should be interpreted as a representation of a low value (but **NOT** zero).

3.1.2 Set `cleaned_data`

- Declare our `cleaned_data`, `discretised_data` and `cleaned_metadata`;
- Generate "clean" pandas-profiling report;
- Export these to files;

[ ]:
```
cleaned_data = staging_data[["id","battery_power","has_bluetooth",
                             ␣
 ↪"clock_speed","has_dual_sim","front_cam_resolution",
                              "has_four_g",
                             "int_memory","mobile_depth","mobile_weight",
                              "number_of_cores", "primary_cam_resolution",␣
 ↪"px_height",
                              "px_width", "ram", "screen_height",
                              "screen_width", "talk_time", "has_three_g",
                              "has_touch_screen", "has_wifi", "is_expensive"
                             ]]
discretised_data = staging_data[["battery_power_category","has_bluetooth",
                             ␣
 ↪"clock_speed_category","has_dual_sim","front_cam_resolution_category",␣
 ↪"has_four_g",
                            ␣
 ↪"int_memory_category","mobile_depth_category","mobile_weight_category",
                              "number_of_cores",␣
 ↪"primary_cam_resolution_category", "px_height_category",
                              "px_width_category", "ram_category",␣
 ↪"screen_height_category",
                              "screen_width_category", "talk_time_category",␣
 ↪"has_three_g",
                              "has_touch_screen", "has_wifi", "is_expensive"
                             ]]
```

Updated Metadata:

3.1.3 Pandas-Profiling (`cleaned_data` and `discretised_data`) Reports

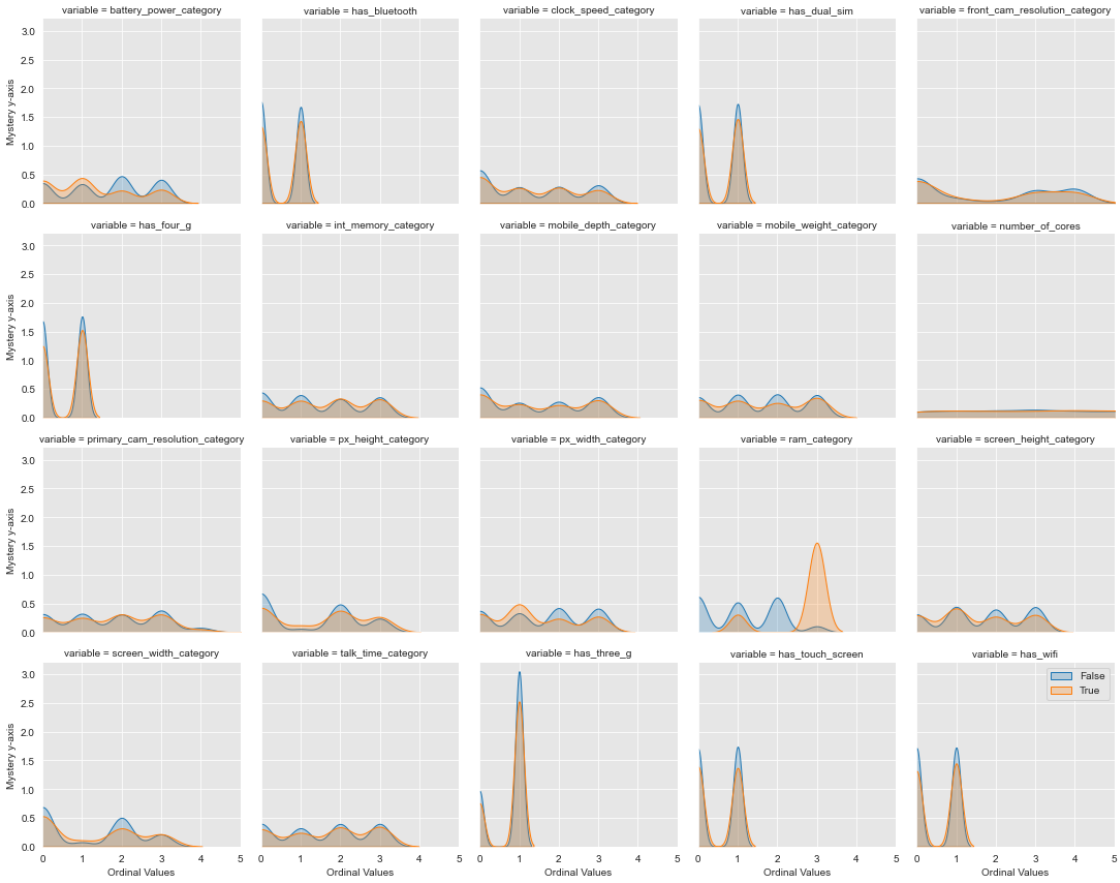
See `Clean Data Profiling Report`, and `Discretised Data Profiling Report` (in appendix).

[ ]:
```
# Sort all the columns based on value (this will determine the Ordinal Encoder)

discretised_ordered_data = discretised_data.copy()
for column in discretised_data.columns:
    if column != 'is_expensive':
        unique_sorted_values =np.sort(np.unique(discretised_data[column].
↪astype(str)))
        dictionary = {}
        for numeric_value in range(len(unique_sorted_values)):
            dictionary[unique_sorted_values[numeric_value]] = numeric_value
        discretised_ordered_data[column] = discretised_data[column].
↪apply(lambda value: dictionary[str(value)])

discretised_ordered_data
```

[32]:
```
melted_data = pd.melt(discretised_ordered_data, "is_expensive",␣
↪discretised_ordered_data.drop(["is_expensive"],axis="columns"),␣
↪value_name="Ordinal Value",)
g = sns.FacetGrid(melted_data, col="variable", hue="is_expensive", col_wrap=5)
g.map(sns.kdeplot, "Ordinal Value", shade=True)
g.set_axis_labels("Ordinal Values", "Mystery y-axis")
plt.xlim(0,5)
plt.legend()
plt.show()
```



This shows the relative frequency graph of the discretised data, in ordinal form, split by the `is_expensive` field. The variable `ram_category` displays a strong variance, regarding its seperation with the `is_expensive` field.

4 Association rule mining

The idea of association rule mining (ARM) is to gather the characteristics that occurs frequently. Our approach is to take the characteristics of each of the phone atrributes (in the record) with the following strategy:

- keep the current cleaned data especially the boolean values;
- discretise the numeric values;

Before the "apriori" runs, the data should be in the form similar to:

has_bluetooth	has_dual_sim	battery_power_0-100	...
True	False	True	True
True	True	False	True
False	True	True	False

4.0.1 ARM Pre-Processing

[33]:
```
# restructuring cleaned_data for association rule mining
ARM_data = discretised_data

# Boolean Tables
ARM_structured_data = ARM_data.copy()[[item[0] for item in zip(ARM_data.
 ↪columns,ARM_data.dtypes) if item[1] == np.bool]]

# Discretise data
non_boolean_values = ARM_data[[item[0] for item in zip(ARM_data.
 ↪columns,ARM_data.dtypes) if item[1] != np.bool]]

# Adding extra columns from discretise data into columns
for non_boolean_column in non_boolean_values.columns:
    unique_values_in_column = ARM_data[non_boolean_column].unique()
    for category_name in unique_values_in_column:

        # Assign true if the current category name matches the data
        ARM_structured_data[f"{non_boolean_column}_{category_name}"] =␣
 ↪ARM_data[non_boolean_column]==category_name

ARM_structured_data.head(3)
```

[33]:

	has_bluetooth	has_dual_sim	has_four_g	has_three_g	has_touch_screen	\
0	False	False	False	False	False	
1	True	True	True	True	True	
2	True	True	True	True	True	

	has_wifi	is_expensive	battery_power_category_(499.502, 875.25]	\
0	True	False	True	
1	False	False	False	
2	False	False	True	

	battery_power_category_(875.25, 1249.5]	\
0	False	
1	True	
2	False	

	battery_power_category_(1623.75, 1998.0]	…	\
0	False	…	
1	False	…	
2	False	…	

	screen_height_category_(4.984999999999999, 8.5]	\
0	False	
1	False	

```
2                                                False

   screen_height_category_(12.0, 15.5]  screen_width_category_(4.5, 9.0]  \
0                                False                               True
1                                False                              False
2                                False                              False

   screen_width_category_(-0.019, 4.5]  screen_width_category_(9.0, 13.5]  \
0                                False                               False
1                                 True                               False
2                                 True                               False

   screen_width_category_(13.5, 18.0]  talk_time_category_(15.5, 20.0]  \
0                               False                               True
1                               False                              False
2                               False                              False

   talk_time_category_(6.5, 11.0]  talk_time_category_(11.0, 15.5]  \
0                           False                            False
1                            True                            False
2                            True                            False

   talk_time_category_(1.981, 6.5]
0                            False
1                            False
2                            False

[3 rows x 69 columns]
```

This is what the data looks like (above), after pre-processing for association rule mining.

4.0.2 Frequent Pattern Mining: Apriori Algorithm

- Explination (apriori algo)
- Minimum support (expln) -> min amount for a rule to be considered a frequent (our case 10%). Probably not something that is good, but putting min support low, allows us to encompass ALL the rules (there are too many rules), whilst also allowing us to filter rules. If we set the support too high, then the program mines a stricter, limited amount of rules.
- Lift (expln)
- Confidence (expln)
- Belo

[35]:
```
# This defines what is considered as a minimum item
MINIMUM_SUPPORT = 0.1
```

```
frequent_items =␣
↪apriori(ARM_structured_data,min_support=MINIMUM_SUPPORT,use_colnames=True,␣
↪verbose=True)
frequent_items
```

```
Processing 570 combinations | Sampling itemset size 5 4
```

[35]:

	support	itemsets
0	0.4950	(has_bluetooth)
1	0.5095	(has_dual_sim)
2	0.5215	(has_four_g)
3	0.7615	(has_three_g)
4	0.5030	(has_touch_screen)
..	…	…
712	0.1165	(has_wifi, has_four_g, has_three_g, screen_wid…
713	0.1165	(has_four_g, is_expensive, has_three_g, ram_ca…
714	0.1000	(has_four_g, clock_speed_category_(0.497, 1.12…
715	0.1100	(px_height_category_(-1.9609999999999999, 490…
716	0.1050	(px_height_category_(-1.9609999999999999, 490…

[717 rows x 2 columns]

The (above) table demonstrates the item sets that meet the minimum defined support (0.1).

4.0.3 Rules by 'Minimum Confidence'

[94]:
```
MINIMUM_CONFIDENCE_THRESHOLD = 0.20 # this is the minimum confidence threshold␣
↪to mine
rules_by_confidence = association_rules(frequent_items, metric="confidence",␣
↪min_threshold=MINIMUM_CONFIDENCE_THRESHOLD)

# Get all the rules that has mobile price
rules = rules_by_confidence[rules_by_confidence["consequents"].
↪map(set(['is_expensive']).issubset)].
↪sort_values("confidence",ascending=False)
rules.head(10)
```

[94]:

	antecedents	\
1553	(has_four_g, ram_category_(3062.5, 3998.0])	
2225	(has_four_g, ram_category_(3062.5, 3998.0])	
2221	(has_four_g, has_three_g, ram_category_(3062.5…	
1883	(has_touch_screen, ram_category_(3062.5, 3998.0])	
1758	(has_three_g, ram_category_(3062.5, 3998.0])	
1889	(has_wifi, ram_category_(3062.5, 3998.0])	
1021	(has_bluetooth, ram_category_(3062.5, 3998.0])	
573	(ram_category_(3062.5, 3998.0])	
1231	(ram_category_(3062.5, 3998.0], has_dual_sim)	
1761	(ram_category_(3062.5, 3998.0])	

```
                     consequents  antecedent support  consequent support  \
1553                (is_expensive)              0.1360              0.2500
2225  (has_three_g, is_expensive)              0.1360              0.1925
2221                (is_expensive)              0.1360              0.2500
1883                (is_expensive)              0.1215              0.2500
1758                (is_expensive)              0.1935              0.2500
1889                (is_expensive)              0.1285              0.2500
1021                (is_expensive)              0.1300              0.2500
573                 (is_expensive)              0.2510              0.2500
1231                (is_expensive)              0.1395              0.2500
1761  (has_three_g, is_expensive)              0.2510              0.1925

      support  confidence      lift  leverage  conviction
1553   0.1165    0.856618  3.426471  0.082500    5.230769
2225   0.1165    0.856618  4.449962  0.090320    5.631795
2221   0.1165    0.856618  3.426471  0.082500    5.230769
1883   0.1030    0.847737  3.390947  0.072625    4.925676
1758   0.1635    0.844961  3.379845  0.115125    4.837500
1889   0.1080    0.840467  3.361868  0.075875    4.701220
1021   0.1090    0.838462  3.353846  0.076500    4.642857
573    0.2090    0.832669  3.330677  0.146250    4.482143
1231   0.1150    0.824373  3.297491  0.080125    4.270408
1761   0.1635    0.651394  3.383867  0.115183    2.316371
```

The above shows a min-confidence threshold of `0.65`, for the top 10 k-rules, and `0.43` for the top 16 k-rules (we have omitted 6, however see `association_rule_mining/rules_by_confidence_minimum_threshold_0.20.csv` for details.

4.0.4 What are `K-rules`??

K rules are the rules that defines a certain frequently appearing item set, to define another attribute/group of attributes.

For example, according to the k=1 rule (sorted by `confidence`), if the phone has a `ram_category` that belongs in the range `3062.5 - 3998.0`, and ALSO has 4G (`has_four_g = True`), then there is a probability of 86% that this phone is expensive (`is_expensive = True`).

It should also be noted that the top 16 rules (`k=16`) all have an antecedent that contains `ram_category` in the aformentioned (`3062.5 - 3998.0`) range.

Above shows the 17th to the 20th rules, it can be seen that the minimum confidence threshold confidence is `0.26`:

- These show rules that do not reference `ram_category`, showing other attributes that can predict is_expensive

4.0.5 Rules by 'Lift'

[39]:
```
MINIMUM_LIFT_THRESHOLD = -5

rules_by_lift = association_rules(frequent_items, metric="lift",␣
↪min_threshold=MINIMUM_LIFT_THRESHOLD)
rules_by_lift["translated_lift"] = rules_by_lift["lift"] - 1
rules_by_lift["is_translated_lift_negative"] = rules_by_lift["translated_lift"]␣
↪< 0
rules_by_lift["absolute_value_of_translated_lift"] = np.
↪abs(rules_by_lift["translated_lift"])

# Get all the rules that has mobile price
rules = rules_by_lift[rules_by_lift["consequents"].map(set(['is_expensive']).
↪issubset)].sort_values("absolute_value_of_translated_lift",ascending=False)
```

Above shows top (k-10) rules, and they all show the `ram_category` as the antecedent.

- Note: see `association_rule_mining/rules_by_lift_sorted_by_absolute_value_of_translated_lift` for the full list of rules.
- top 1 k-rule states that when a phone has ram category in the range `3062.5- 3998.0` and has four g, then there is a very strong positive correlation that
- explain that the rules are sorted by the absolute_value_of_translated_lift in order to show sorted strength of correlation
- top 17 the rules here shows that ram category in the range `3062.5- 3998.0`
- the 18th up to the 20th show different attribute with a lift that is positively correlated, but not as strong as the previous the rules.

4.1 Recommendation (from ARM)

- According ARM

1. RAM category belong between `3062.5- 3998.0`
2. four G
3. touch scren
4. threeg
5. wifi
6. bluetooth

- ranked by the strongest defining confidence

5 Classification

- brief explanation of classification
- talk a little bit about cross validation of folds of 10 (avoiding overfitting by using different parts of the data)

- talk a little bit about the RANDOM_STATE seeding (seed is defined to prevent uncontrolled changing of results)
- we will do it in Decision Tree and SVM
- the metric that will be used is accuracy (is_expensive and not expensive is equal in weighting)
- cleaned_data of continuous (not discrete). Methods such as SVM and Decision Trees are better at separating continuous attributes.

[43]:
```
# Preprocessing of Cleaned Data
learning_data = cleaned_data.drop("id",axis="columns")
target_data = learning_data["is_expensive"]
feature_data = learning_data.drop("is_expensive",axis="columns")
```

[103]:
```
# classification helpers

def show_top_results( gridsearch, target_results=["rank_test_score",␣
↪"mean_test_score", "params"], number_of_params=5 ):

    return pd.DataFrame(gridsearch.cv_results_).
↪sort_values(by="mean_test_score",ascending=False)[target_results].
↪head(number_of_params)
```

5.1 Decision Tree

- brief explanation what decision tree is decision on (gini index and information gain)

[98]:
```
dt_pipeline_1 = Pipeline([
    ("dt_classifier",DecisionTreeClassifier(random_state=RANDOM_STATE))
])
# Grid Search
param_grid = {
    'dt_classifier__criterion': ["gini", "entropy"],
    'dt_classifier__max_depth': [None] + list(range(1,len(feature_data.
↪columns)))
}
```

[ ]:
```
dt_pipeline_1_search = GridSearchCV(dt_pipeline_1, param_grid, n_jobs=-1,␣
↪cv=CROSS_VALIDATION_PARTITION)
dt_pipeline_1_search.fit(feature_data,target_data)
```

- Table above shows the top 5 hyper-parameters of the decision tree
- talk about the best test score, the params (information gain) and compare the result with gini index

6 Support Vector Machine

attribute reuction good with svm, not so great with the PCa -> Select k best...

[48]:
```
svm_pipeline_2 = Pipeline([
    ("svm_classifier",SVC())
])

# Grid Search
param_grid = {
    'svm_classifier__kernel': ['linear', 'poly', 'rbf', 'sigmoid']
}
```

[ ]:
```
svm_pipeline_2_search = GridSearchCV(svm_pipeline_2, param_grid, n_jobs=-1,␣
↪cv=CROSS_VALIDATION_PARTITION)
svm_pipeline_2_search.fit(feature_data,target_data)
```

- Talk about the top 5 best algorithm with their difference in parameters with emphasis on the best score

6.1 Comparison between DC and SVM

- Say that SVM is Chad between the two
- in terms of reasoning, it may be because SVM is robust to noise and outliers, hnece having a strong generalisation ability compared to a decision tree that fits better??

7 Clustering

- explain a little bit about clustering

7.1 Cluster of Size 2

- explain that we would expect that there are 2 groups and that is ideal for comparison with `is_expensive` field

7.1.1 Kmeans

- short explanation of what kmeans

Confusion Table

[52]:
```
kmeans_model = KMeans(n_clusters=2, random_state=RANDOM_STATE)
kmeans_model.fit(feature_data)
kmeans_prediction = kmeans_model.labels_
```

- explain that the clustering has 73.85% accuracy wherein the other 26.15% inaccuracy is attributed to kmeans predicting a phone `is_expensive` when it is not (false positive), and 0% for false negative.

7.1.2 Aglomerative Hierahical Clustering

- short explanation of what it is about

[54]:
```
ac_pipeline = Pipeline([
    ("ac", AgglomerativeClustering(distance_threshold=0,n_clusters=None)) #␣
 ↪Compute distances
])
```

[ ]:
```
ac_pipeline.fit(feature_data)
```

Dendrogram

- short explanation of the splitting of hierarchy into 4 parts, and also mentioning the size of 2 clusters (219+514) + (507+760)

Confusion Table

- explain hierarchical is not as great as kmeans. Although kmeans is sensitive to noise and outliers, agglometive is more sensitive as it can classify an outlier as its own hierachy or category.

7.1.3 Validation of Two Cluster Size

- in the previous heading, it is assumed that cluster of size 2 is expected. This is an extra exploratory to validate whether a cluster of size 2 is the best way to split the data.
- explain silhouette score in the number of cluster
- say that 2 cluster is best according to silhouette score

8 Data Reduction

- quick explanation data reduction and importance

8.1 Numerosity Reduction

- quick explanation to remove rows refers to like outliers or incomplete data

[60]:
```
# To be done by Frinze using DBScan
```

8.2 Attribute Reduction

8.2.1 Principal Component Analysis (PCA)

- quick explanation of PCA
- need scaling before doing PCA

Decision Tree with PCA

[ ]:
```
dt_pipeline_3 = Pipeline([
    ("scale", StandardScaler()),
    ("pca", PCA()),
    ("dt_classifier",DecisionTreeClassifier(random_state=RANDOM_STATE))
])
# Grid Search
```

```
param_grid = {
    'pca__n_components': range(1,len(feature_data.columns)),
    'dt_classifier__criterion': ["gini", "entropy"],
    'dt_classifier__max_depth': [None] + list(range(1,len(feature_data.
 ↪columns)))
}
```

[ ]:
```
dt_pipeline_3_search = GridSearchCV(dt_pipeline_3, param_grid, n_jobs=-1,␣
 ↪cv=CROSS_VALIDATION_PARTITION)
dt_pipeline_3_search.fit(feature_data,target_data)
```

- explain PCA worsened Decision Tree Classifier from 94.75% amount to 84% accuracy

SVM with PCA

[ ]:
```
svm_pipeline_3 = Pipeline([
    ("scale", StandardScaler()),
    ("pca", PCA()),
    ("svm_classifier",SVC())
])

# Grid Search
param_grid = {
    'pca__n_components': range(1,len(feature_data.columns)),
    'svm_classifier__kernel': ['linear', 'poly', 'rbf', 'sigmoid',␣
 ↪'precomputed']
}
```

[ ]:
```
svm_pipeline_3_search = GridSearchCV(svm_pipeline_3, param_grid, n_jobs=-1,␣
 ↪cv=CROSS_VALIDATION_PARTITION)
svm_pipeline_3_search.fit(feature_data,target_data)
```

- PCA improved SVM from 98.95 to 99.1%

8.2.2 Select K-Best with Information Gain (Entropy) and ANOVA F-statistic

- quick explanation information gain (used in decision trees) and ANOVA looking at variance for forward and backward selection

Decision Tree with Select K-Best

[69]:
```
dt_pipeline_4 = Pipeline([
    ("select_kbest", SelectKBest()),
    ("dt_classifier",DecisionTreeClassifier(random_state=RANDOM_STATE))
])
# Grid Search
param_grid = {
    'select_kbest__k': range(1,len(feature_data.columns)),
    'select_kbest__score_func': [f_classif,mutual_info_classif], # F-value␣
 ↪statistic and Information Gain (entropy)
```

```
    'dt_classifier__criterion': ["entropy"], # - Entropy is better than GINI␣
 ↪from previous results
    'dt_classifier__max_depth': [None] + list(range(1,len(feature_data.
 ↪columns)))
}
```

[ ]:
```
dt_pipeline_4_search = GridSearchCV(dt_pipeline_4, param_grid, n_jobs=-1,␣
 ↪cv=CROSS_VALIDATION_PARTITION)
dt_pipeline_4_search.fit(feature_data,target_data)
```

- explanation on the attributes (some comparison from the earlier exploratory data analaysis), explain there are 15 attributes
- improvement from 94.75% to 95.15% accuracy
- explain that in this scenario F-statistic is better than information gain in the transformation, but information gain is later used for the decision tree classifier

SVM with Select K-Best

[76]:
```
svm_pipeline_4 = Pipeline([
    ("select_kbest", SelectKBest()),
    ("svm_classifier",SVC())
])

# Grid Search
param_grid = {
    'select_kbest__k': range(10, 18),
    'select_kbest__score_func': [f_classif,mutual_info_classif], # F-value␣
 ↪statistic and Information Gain (entropy)
    'svm_classifier__kernel': ['linear', 'rbf'] # removed other attributes that␣
 ↪seems to be inaccurate
}
```

[ ]:
```
svm_pipeline_4_search = GridSearchCV(svm_pipeline_4, param_grid, n_jobs=-1,␣
 ↪cv=CROSS_VALIDATION_PARTITION)
svm_pipeline_4_search.fit(feature_data,target_data)
```

- improvement 98.95% to 99.1% (0.04% better than PCA)
- compare the previous attribute selection if it is the same

9 Attribute Selection

- do manual selection of attribute
- compare to previous
- also refer to the decision tree

[81]:
```
MANUALLY_SELECTED_ATTRIBUTES = [
    "ram",
    "has_four_g",
```

```
    "mobile_depth",
    "screen_width",
    "screen_height",
    "primary_cam_resolution",
    "number_of_cores",
    "int_memory",
    "px_width",
    "px_height",
] # based on domain-specific knowledge and the exploratory plots
feature_data_manually_selected = learning_data.
 ↪drop("is_expensive",axis="columns")[MANUALLY_SELECTED_ATTRIBUTES]
```

9.1 Decision Tree with Manually Selected Attribute

[82]:
```
dt_pipeline_1 = Pipeline([
    ("dt_classifier",DecisionTreeClassifier(random_state=RANDOM_STATE))
])
# Grid Search
param_grid = {
    'dt_classifier__criterion': ["gini", "entropy"],
    'dt_classifier__max_depth': [None] + list(range(1,len(feature_data.
 ↪columns)))
}
```

[ ]:
```
dt_pipeline_1_search = GridSearchCV(dt_pipeline_1, param_grid, n_jobs=-1,␣
 ↪cv=CROSS_VALIDATION_PARTITION)
dt_pipeline_1_search.fit(feature_data_manually_selected,target_data)
```

- discuss manually selected attribute lowered from 94.75% to 92.35%

9.2 SVM with Manually Selected Attributes

[86]:
```
svm_pipeline_2 = Pipeline([
    ("svm_classifier",SVC())
])

# Grid Search
param_grid = {
    'svm_classifier__kernel': ['linear', 'poly', 'rbf', 'sigmoid']
}
```

[ ]:
```
svm_pipeline_2_search = GridSearchCV(svm_pipeline_2, param_grid, n_jobs=-1,␣
 ↪cv=CROSS_VALIDATION_PARTITION)
svm_pipeline_2_search.fit(feature_data_manually_selected,target_data)
```

- discuss lowered accuracy from 98.95% to 92.95%

10 Appendix

Raw Data Profiling Report

[ ]:
```
raw_profile = ProfileReport(raw_data, explorative=True, orange_mode=True,␣
 ↪title="Raw Data Profiling Report")

# set the Metadata
metadata_dict = raw_metadata.to_dict()["Explaination"]

raw_profile.set_variable("variables.descriptions",metadata_dict)
raw_profile.to_file("./profile_reports/raw_data_profile.html")
# raw_profile
```

Clean Data Profiling Report

[92]:
```
# Generate new pandas-profiling
cleaned_data_profile = ProfileReport(cleaned_data, explorative=True,␣
 ↪orange_mode=True, title="Clean Data Profiling Report")
cleaned_data_profile.set_variable("variables.
 ↪descriptions",cleaned_metadata_dict) # Set Metadata
cleaned_data_profile.to_file("./profile_reports/cleaned_data_profile.html")
# cleaned_data_profile
```

```
C:\Users\Max Matthews\anaconda3\lib\site-packages\pandas\core\frame.py:4296:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  return super().rename(
```

```
HBox(children=(HTML(value='Summarize dataset'), FloatProgress(value=0.0, max=35.
 ↪0), HTML(value='')))
```

```
HBox(children=(HTML(value='Generate report structure'), FloatProgress(value=0.0,␣
 ↪max=1.0), HTML(value='')))
```

```
HBox(children=(HTML(value='Render HTML'), FloatProgress(value=0.0, max=1.0),␣
 ↪HTML(value='')))
```

```
HBox(children=(HTML(value='Export report to file'), FloatProgress(value=0.0,␣
 ↪max=1.0), HTML(value='')))
```

Discretised Data Profiling Report

[93]:
```
# Generate new pandas-profiling
discretised_data_profile = ProfileReport(discretised_data, explorative=True,␣
 ↪orange_mode=True, title="Discretised Data Profiling Report")
discretised_data_profile.set_variable("variables.
 ↪descriptions",cleaned_metadata_dict) # Set Metadata
discretised_data_profile.to_file("./profile_reports/discretised_data_profile.
 ↪html")
# discretised_data_profile
```

```
C:\Users\Max Matthews\anaconda3\lib\site-packages\pandas\core\frame.py:4296:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  return super().rename(
```

```
HBox(children=(HTML(value='Summarize dataset'), FloatProgress(value=0.0, max=34.
 ↪0), HTML(value='')))
```

```
HBox(children=(HTML(value='Generate report structure'), FloatProgress(value=0.0,␣
 ↪max=1.0), HTML(value='')))
```

```
HBox(children=(HTML(value='Render HTML'), FloatProgress(value=0.0, max=1.0),␣
 ↪HTML(value='')))
```

```
HBox(children=(HTML(value='Export report to file'), FloatProgress(value=0.0,␣
 ↪max=1.0), HTML(value='')))
```