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

Max Matthews, Frinze Erin Lapuz (22711649)

May 21, 2021

Author: Max Matthews and Frinze Erin Lapuz (22711649)

This is the IPython Jupyter Notebook for generating the PDF report, documentation, and generating the staging_data.

To generate html, pdf and tex file, run the following command `generate_report.bat` in the command line. You may need to have Anaconda activated, Jupyter NBconvert, and LaTex renderer (eg. MikTex) in your environment to do this.

1 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.

1.1 Tasks and Scope

1.1.1 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.

1.1.2 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)?

1.1.3 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.

1.1.4 4) Clustering

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

1.1.5 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”?

1.1.6 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?

1.2 Marking Scheme

[5 marks] Explain the data processing operations (e.g., remove some attributes and action on inconsistent data) that you have done.

[5 marks] Explain and interpret the top k association rules mined; based on the association rules, provide a recommendation for a company willing 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 variable).

[5 marks] Explain the data reduction you have performed; compare the classifiers trained on reduced data with the classifiers trained on the original data.

[5 marks] Your answer to Task 6.

1.3 Tools, Libraries and Packages

Python - Used throughout the project for data cleaning, data processing, and modelling.

1.3.1 Imports

https://graphviz.org/download/

[ ]:
```
pip install pandas-profiling[notebook] mlxtend clusteval pca graphviz dtreeviz␣
↪--user
```

[2]:
```python
# Data analysis, manipulation, and profiling
import pandas as pd
pd.options.display.max_colwidth = 100

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
from sklearn import tree
import graphviz
from dtreeviz.trees import dtreeviz

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

# Data Reduction
from sklearn.ensemble import IsolationForest
from sklearn.svm import OneClassSVM
from sklearn.covariance import EllipticEnvelope
from sklearn.cluster import DBSCAN
from pca import pca
```

2 Data Cleaning and Profiling

There are many ways to approach a project's data cleaning, and data profiling steps. For these processes (in Project 2) we will be using an IPython Notebook, for the following reasons:

- Both group members 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 to the default Anaconda package, with exception to `pandas_profiling`, `mlxtend`, and `pca`.

- `pandas_profiling` (from https://github.com/pandas-profiling/pandas-profiling) was leveraged to provide a detailed exploratory analysis of our data, and the attributes we would be working with. Data profiling is crucial to the measurement of the quality of data, which in turn greatly assists the analyst team in their discovery of data anomalies/inconsistencies, and as such, appropriate data transformations and/or pre-processing actions. The data-profiling reports (generated by this package) will be referenced in our project discussion, with the full resource located in the appendix.

- `mlxtend` (from https://github.com/rasbt/mlxtend) was used in our Association Rule Mining section, to assist in the mining of such associated rules.

- `pca` (from https://github.com/erdogant/pca) was used for graphical presentation of the Principal Component Analysis.

[ ]:
```
raw_data = pd.read_csv("./data/raw/mobile_price.csv")
raw_metadata = pd.read_excel("./data/raw/ColumnDescription.xlsx",␣
↪index_col="Column")

raw_data
```

[ ]:
```
raw_metadata.to_dict()["Explaination"]
```

2.1 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.

2.2 Raw Data - Interpretation, Analysis, and Cleaning

Here, we will walk through the fields presented in the raw data, stating assumptions/decisions, and undertaking any appropriate cleaning steps. In the event of changes, we will apply them to the `staging_data` data frame, and `staging_metadata` dictionary.

[ ]:
```python
# Clone the raw data & metadata
staging_data = raw_data.copy()
staging_metadata = raw_metadata.copy()

# By default the new_name is the old_name
staging_metadata["new_name"] = staging_metadata.index
```

[ ]:
```python
# helper functions

def string_to_bool(value):
    """
    Converts String representation of Boolean ("yes", "has")/("no","not") to␣
 ↪True/False
    """
    if value.lower() in ["yes","has"]:
        return True
    elif value.lower() in ["no","not"]:
        return False
    return None


def int_to_bool(value):
    """
    Converts int representation of Boolean (1, 0) to True/False
    """
    return value == 1


def create_discretised_col(df, target_col, new_col_name=None,␣
 ↪is_zero_its_own_category=True):
    """
    This function discretises the column
    """
    # Default value of new col
    if new_col_name is None:
        new_col_name = f"{target_col}_category"
```

```
    col_holder = df[target_col]
    df[new_col_name] = 0 # Initialise

    if is_zero_its_own_category:
        df.loc[col_holder>0, new_col_name] = pd.
↪cut(col_holder[col_holder>0],bins=4,include_lowest=True)
    else:
        df.loc[:, new_col_name] = pd.cut(col_holder,bins=4,include_lowest=True)

    return df
```

`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.

[ ]:
```
staging_data = create_discretised_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 many inconsistencies. We observe 10 distinct values that all refer to either the phone having bluetooth (True), or not having bluetooth (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 a value of 0.5. We note that the distribution of values in the histogram is right skewed. This data is continuous and will need to be discretised if we wish to use it in the association rule mining process.

[ ]:
```
staging_data = create_discretised_col(staging_data, "clock_speed",␣
↪is_zero_its_own_category = False)
```

---

dual_sim The dual_sim attirbute is similar to the **blue** / **has_bluetooth** attribute, wherein it is a categorical datatype (*boolean*) with its inconsistencies forming 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). This continous data will be discretised for later (association rule mining) purposes, with present zero values (as a result of our domain interpretation) populating their own level within this category.

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

---

four_g The four_g attribute is a categorical (*boolean*) attribute, with values indicated as `yes = 1` and `no = 0` regarding (the phone's) 4G capability. For consistency, this will be converted from current numeric form 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 phase.

```
[ ]: staging_data = create_discretised_col(staging_data, "int_memory",␣
      ↪is_zero_its_own_category = False)
```

---

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. Additionally, 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_discretised_col(staging_data, "mobile_depth",␣
↪is_zero_its_own_category = False)
```

---

mobile_wt The mobile_wt (*Mobile Weight*) attribute has 121 distinct values that vary as shown in the *profiling-report* histogram. The respective min and max values are 80 and 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_discretised_col(staging_data, "mobile_weight",␣
↪is_zero_its_own_category = False)
```

---

n_cores The n_cores (*Number of Cores*) attribute looks almost uniform in the range 1-8. This is categorical (numerical) data, and requires no further attention at this moment.

[ ]:
```
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 displays 21 distinct values, with 101 zero values (5.1% of the data). We will be interpreting the zero values as “this phone does not have a primary camera”, (please see the ***Data Privacy Disclaimer*** below). This continuous data will be discretised for later (association rule mining) purposes, with present zero values (as a result of our domain interpretation) populating their own level within this category.

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

---

px_height The px_height (*Pixel Height*) attribute has a right-skewed, normal distribution, with a mean and standard deviation of 645 and 443.79 pixels (respectively), and values falling 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 nonsensical low values. For association rule mining purposes, this continuous data will be discretised, wherein (as per the assumption above) the zero values will fall within the first category, shared by other values (relatively) close to zero.

[18]:
```
staging_data = create_discretised_col(staging_data, "px_height",␣
↪is_zero_its_own_category = False)
```

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

```
[ ]: staging_data = create_discretised_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 continuous data will be discretised, for later (association rule mining) purposes.

```
[ ]: # Discretizing ram
     staging_data = create_discretised_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 continuous 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_discretised_col(staging_data, "screen_height",␣
      ↪is_zero_its_own_category = False)
```

`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 any zero values as representatives of sensible, low values (please see the ***Data Privacy Disclaimer*** below). Note that this notion also applies to any non-zero, nonsensical low values. For association rule mining purposes, this continuous data will be discretised, wherein (as per the assumption above) the zero values will fall within the first category, shared by other values (relatively) close to zero.

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

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

```
[ ]: staging_data = create_discretised_col(staging_data, "talk_time",␣
      ↪is_zero_its_own_category = False)
```

---

three_g The three_g attribute is similar to the has_bluetooth and dual_sim attribute, wherein it is a categorical (*boolean*) datatype, with inconsistencies spread to 10 (string) distinct values. As such, 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 initial state of the imported four_g attribute. 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 (*has wifi*) attribute 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 categorical (*boolean*) data, 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
```

2.3 *Data Privacy Disclaimer*

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

2.3.1 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.

2.3.2 Inappropriate zero values

- px_height
- screen_width

Zero / close to zero values for these attributes can be observed as **values altered for privacy reasons** and should be interpreted as a representation of a relatively low value (but **NOT** taken literally as the stated value).

2.4 Set cleaned_data and discretised_data

- Declare our cleaned_data, discretised_data and cleaned_metadata;
- Generate "clean" pandas-profiling report;
- Export these to .csv 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:

[29]:
```
# Cleaned metadata
cleaned_metadata = staging_metadata.set_index("new_name")
cleaned_metadata_dict = cleaned_metadata.to_dict()["Explaination"]
cleaned_metadata
```

[29]:
```
                                  Explaination
new_name
id
ID
battery_power                                                         Total energy a
battery can store in one time measured in mAh
has_bluetooth
Has bluetooth or not
clock_speed                                                                     speed at
which microprocessor executes instructions
has_dual_sim
```

```
Has dual sim support or not
front_cam_resolution
Front Camera mega pixels
has_four_g
Has 4G or not. 1 = yes , 0 = no
int_memory
internal Memory in Gigabytes
mobile_depth
Mobile Depth in cm
mobile_weight
Weight of mobile phone
number_of_cores
Number of cores of processor
primary_cam_resolution
Primary Camera mega pixels
px_height
Pixel Resolution Height
px_width
Pixel Resolution Width
ram
Random Access Memory in Mega Bytes
screen_height
Screen Height of mobile in cm
screen_width
Screen Width of mobile in cm
talk_time                                               longest time that a
single battery charge will last when you are
has_three_g
Has 3G or not
has_touch_screen
Has touch screen or not, 1 = yes, 0 = no
has_wifi
Has wifi or not
is_expensive            This is the target variable with indicating if the
mobile phone got a high price. 1 = yes, 0 = no
```

[ ]:
```
cleaned_data.to_csv("./data/cleaned/mobile_price.csv",index=False) # Export␣
↪cleaned_data
discretised_data.to_csv("./data/cleaned/mobile_price_discretised.
↪csv",index=False) # Export discretised_data
```

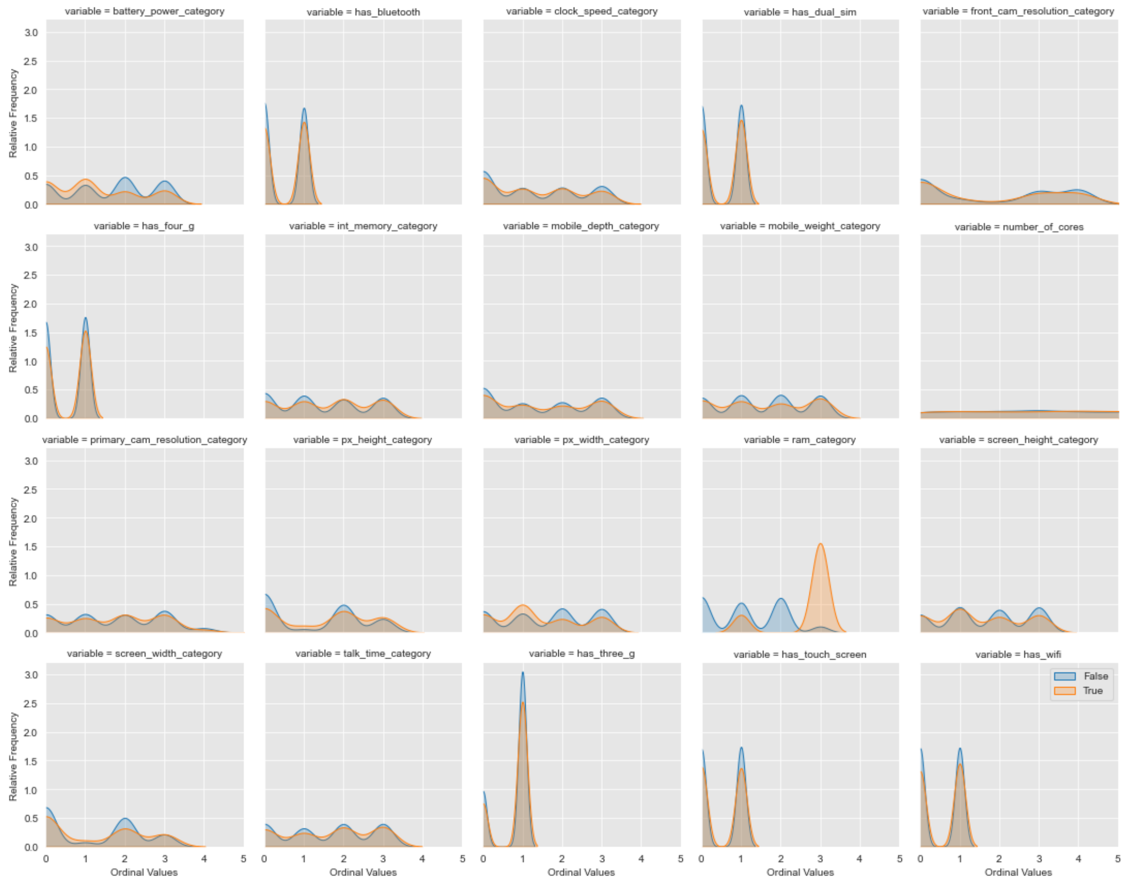
2.4.1 Pandas-Profiling (`cleaned_data` and `discretised_data`) Reports

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", "Relative Frequency")

plt.xlim(0,5)
plt.legend()
plt.show()
```



This (above) shows the relative frequency graph of the discretised data, in ordinal form, split by the phone-data's `is_expensive` field. The variable `ram_category` displays a strong variance regarding its seperation with `is_expensive`, which following steps will investigate further.

3 Association Rule Mining

3.1 Information

Association rule mining ('ARM') is a technique used investigate and uncover frequently occurring patterns, correlations, or associations, between the characteristics of a given dataset/s or source. Association rules comprise of two parts: - antecedent (if) → An antecedent is something that is found in data; - consequent (then) → A consequent is an item that is found in combination with the antecedent.

3.2 ARM Approach

Our approach is to take the characteristics of each of the phone attributes (from our data) with the following strategy:

- keep the (current) cleaned data, especially the boolean values;
- discretise any numeric values;

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

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

For completed pre-processing, see IPython Notebook or `./data/cleaned/ARM_item_characteristics.csv`

```
[ ]: # 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)
```

```
[ ]: ARM_structured_data.to_csv("./data/cleaned/ARM_item_characteristics.csv",␣
      ↪index=False) # export to csv
```

3.3 Apriori Algorithm

Apriori (*Apriori Algorithm*) is an algorithm used in frequent-item-set / pattern mining over relational datasets. It seeks to identify frequent, individual items in the data, and then extend them to (incrementally) larger item sets, on the proviso that such item sets appear sufficiently often enough in said data.

3.4 Support

The recognition of frequently occurring patterns (in a dataset) must first be tuned by certain analyst-defined parameters. The minimum support acts as an 'entry requirement value' that rules must meet/exceed, for them to be considered 'frequent'. A minimum support value set too low, will not filter out patterns. If it is set too high, this strict program's mining expedition will return no rules, or only extremely dominant attribute-patterns. Example below, where we set `MINIMUM_SUPPORT = 0.1`, and then begin to mine the associated rules.

3.4.1 Rules by 'Minimum Support'

[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  \
0     0.4950
1     0.5095
2     0.5215
3     0.7615
4     0.5030
..       …
712   0.1165
713   0.1165
714   0.1000
715   0.1100
716   0.1050

                itemsets
0
(has_bluetooth)
1
(has_dual_sim)
2
(has_four_g)
3
(has_three_g)
4
(has_touch_screen)
..
…
712                             (has_three_g, has_four_g, has_wifi,
screen_width_category_(-0.019, 4.5])
713                               (has_three_g, has_four_g,
ram_category_(3062.5, 3998.0], is_expensive)
714  (has_three_g, has_four_g, clock_speed_category_(0.497, 1.125],
screen_width_category_(-0.019, 4.5])
715  (has_three_g, has_four_g, px_height_category_(-1.9609999999999999, 490.0],
front_cam_resolution_…
```

```
716  (has_three_g, has_four_g, px_height_category_(-1.9609999999999999, 490.0],
screen_width_category…

[717 rows x 2 columns]
```

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

3.5 Confidence

In ARM, confidence is an indication of how often the rule has been found to be true. The confidence value of a rule, { $\boldsymbol{X}$ ==> $\boldsymbol{Y}$ } with respect to a set of **z**'s {$\boldsymbol{Z}$}, is the proportion of the **z**'s that contains $\boldsymbol{X}$, which also contains $\boldsymbol{Y}$.

3.5.1 Rules by 'Minimum Confidence'

[36]:
```python
ARM_COLS_OF_INTEREST = ['antecedents', 'consequents', 'support', 'confidence',␣
↪'lift']
```

[37]:
```python
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[ARM_COLS_OF_INTEREST].reset_index(drop=True).head(10)
```

[37]:
```
                                          antecedents  \
0                (has_four_g, ram_category_(3062.5, 3998.0])
1                (has_four_g, ram_category_(3062.5, 3998.0])
2   (has_three_g, has_four_g, ram_category_(3062.5, 3998.0])
3          (has_touch_screen, ram_category_(3062.5, 3998.0])
4               (has_three_g, ram_category_(3062.5, 3998.0])
5                  (ram_category_(3062.5, 3998.0], has_wifi)
6             (ram_category_(3062.5, 3998.0], has_bluetooth)
7                            (ram_category_(3062.5, 3998.0])
8              (has_dual_sim, ram_category_(3062.5, 3998.0])
9                            (ram_category_(3062.5, 3998.0])

                   consequents  support  confidence      lift
0               (is_expensive)   0.1165    0.856618  3.426471
1  (has_three_g, is_expensive)   0.1165    0.856618  4.449962
2               (is_expensive)   0.1165    0.856618  3.426471
3               (is_expensive)   0.1030    0.847737  3.390947
4               (is_expensive)   0.1635    0.844961  3.379845
```

```
5                (is_expensive)   0.1080    0.840467  3.361868
6                (is_expensive)   0.1090    0.838462  3.353846
7                (is_expensive)   0.2090    0.832669  3.330677
8                (is_expensive)   0.1150    0.824373  3.297491
9  (has_three_g, is_expensive)   0.1635    0.651394  3.383867
```

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

3.5.2 Explained: `top-k` Rules

The `top-k` rules are the number (***k***) of rules (attribute patterns) that based on the confidence/support parameters, occur with the highest frequency in the dataset.

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 `3062.5 - 3998.0` range.

[38]:
```
rules[ARM_COLS_OF_INTEREST].reset_index(drop=True).iloc[17:21]
```

[38]:
```
                                    antecedents                     consequents  \
17  (front_cam_resolution_category_(0.981, 5.5])                  (is_expensive)
18                         (has_three_g, has_four_g)                  (is_expensive)
19                                      (has_four_g)  (has_three_g, is_expensive)
20                                      (has_four_g)                  (is_expensive)

    support  confidence      lift
17   0.1160    0.264840  1.059361
18   0.1375    0.263663  1.054650
19   0.1375    0.263663  1.369675
20   0.1375    0.263663  1.054650
```

[ ]:
```
rules.to_csv("association_rule_mining/rules_by_confidence_minimum_threshold_0.
↪20.csv")
```

Comment: The (above) table shows the 17th to the 20th rules, when the minimum confidence threshold is set to `0.26`. Here, we observe rules that do not reference `ram_category`, showing other attributes that can predict `is_expensive`.

3.6 Lift

Lift quantifies an association rule's ability to predict cases (as) possessing an improved response (against the population), measured against a random choice targeting model. An association rule is doing well (according to lift) if the response (within the target) exceeds the average for the population as a whole. Simply put, lift assesses the degree to which the occurrence of one (characteristic) "lifts" the occurrence of the other.

```
lift (A, B) = P( A U B ) / P(A) P(B)
```

(`lift < 1`) → The occurrence of A is **negatively** correlated with the occurrence of B;

(`lift > 1`) → The occurrence of A is **positively** correlated with the occurrence of B;

(`lift = 1`) → The occurrence of A is **independent** of the occurrence of B;

3.6.1 Rules by 'Lift'

[40]:
```
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)
```

[ ]:
```
rules.to_csv("association_rule_mining/
 ↪rules_by_lift_sorted_by_absolute_value_of_translated_lift.csv")
```

[42]:
```
rules[ARM_COLS_OF_INTEREST].reset_index(drop=True).head(10)
```

[42]:
```
                                            antecedents  \
0               (has_four_g, ram_category_(3062.5, 3998.0])
1              (has_three_g, ram_category_(3062.5, 3998.0])
2                           (ram_category_(3062.5, 3998.0])
3               (has_four_g, ram_category_(3062.5, 3998.0])
4  (has_three_g, has_four_g, ram_category_(3062.5, 3998.0])
5          (has_touch_screen, ram_category_(3062.5, 3998.0])
6                           (ram_category_(3062.5, 3998.0])
7              (has_three_g, ram_category_(3062.5, 3998.0])
8                           (ram_category_(3062.5, 3998.0])
9                           (ram_category_(3062.5, 3998.0])

                     consequents  support  confidence      lift
0      (has_three_g, is_expensive)   0.1165    0.856618  4.449962
1       (is_expensive, has_four_g)   0.1165    0.602067  4.378670
2     (is_expensive, has_dual_sim)   0.1150    0.458167  3.457867
3                   (is_expensive)   0.1165    0.856618  3.426471
4                   (is_expensive)   0.1165    0.856618  3.426471
5                   (is_expensive)   0.1030    0.847737  3.390947
6      (has_three_g, is_expensive)   0.1635    0.651394  3.383867
```

```
7                              (is_expensive)   0.1635    0.844961  3.379845
8                   (is_expensive, has_four_g)   0.1165    0.464143  3.375589
9   (has_three_g, has_four_g, is_expensive)   0.1165    0.464143  3.375589
```

Comment: The (above) table shows top (`k=10`) rules, where we notice that `ram_category` is present in all (10) antecedents (see `association_rule_mining/rules_by_lift_sorted_by_absolute_value_of_translated_lift.csv` for the full list of rules).

We have ordered these rules by `absolute_value_of_translated_lift`, in order to better highlight the strongest attribute correlations (whether they be positive, or negative).

- The top `k=1` rule states that when a phone has `ram_category` in the range `3062.5 - 3998.0`, and also has 4g (`has_four_g=true`), then these attributes are highly positively correlated to the `has_three_g` and `is_expensive` attribute pair (lift=`4.449`).
- It should be noted that the top (`k=17`) rules all reference `ram_category` in the range `3062.5-3998.0`.

[43]:
```
rules[ARM_COLS_OF_INTEREST].reset_index(drop=True)[18:21]
```

[43]:
```
     antecedents                                           consequents  \
18  (has_four_g)  (has_three_g, ram_category_(3062.5, 3998.0], is_expensive)
19  (has_three_g)                              (is_expensive, has_four_g)
20  (has_three_g)   (is_expensive, has_four_g, ram_category_(3062.5, 3998.0])

    support  confidence      lift
18   0.1165    0.223394  1.366324
19   0.1375    0.180565  1.313198
20   0.1165    0.152988  1.313198
```

Comment: The 18th up to the 20th (above) show different attribute with a lift that is positively correlated, but not as strong as the previous the rules.

3.7 Association Rule Mining - Recommendation for Designing an Expensive Phone

Based on our results from association rule mining, we can advise a manufacturer (who wishes to design an expensive phone) on the following features…

The most notable attribute in an item-set, that resulted in an expensive phone, contained RAM that fell in the RAM category (range) `3062.5- 3998.0`. However, within the same item-set/s as RAM, we noted the following (most frequent) attributes:

1. four G
2. touch scren
3. three G
4. wifi
5. bluetooth

…ranked by the strongest defining confidence. This discovery upholds the initial patterns that we observed in the graph of relative frequency of all attributes (in their ordinal form), divided by the is_expensive field.

4 Classification

Classification is the supervised-learning process of determining and assigning classes to data rows. This process can identify the class of an unknown row, based upon data row/s that a model has been trained upon.

Classification in CITS 3401 - Project 2 will be done with cross validation of ten folds, using differing partitions of our dataset for testing and training, to avoid overfitting our model. Furthermore, the training will be done with a RANDOM_STATE seeding, for the express purpose of mitigating the uncontrollable inconsistency of algorithm that uses randomisation.

The classification will be done with both Decision Tree ('DT') and Support Vector Machine ('SVM'), with hyperparameter optimisation. The metric will be accuracy, as the attribute is_expensive is weighted equally (between the possible True or False values).

We will be using the cleaned_data dataset (not the wholly discretised dataset that was used in the Association Rule Mining), as SVM works well in seperating continuous attributes, whilst DT can take either continuous ('Regression Tree') or discrete ('Classification Tree') data.

[44]:
```
# Seperation of data into features and target
learning_data = cleaned_data.drop("id",axis="columns")
target_data = learning_data["is_expensive"]
feature_data = learning_data.drop("is_expensive",axis="columns")
```

[ ]:
```
# 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)
```

4.1 Decision Tree

A decision tree is an algorithm that separates the data using different thresholds, within different attributes. There are two main criterium for choosing an attribute and a threshold:

- Gini Index -> Information Gain calculates effective change in entropy after making a decision based on the value of an attribute.
- Information Gain (entropy) -> The gini index, calculates the amount of probability of a specific feature that is classified incorrectly when selected randomly.

We will now configure and test the aforementioned classifiers…

[102]:
```
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)
```

[104]:
```
show_top_results(dt_pipeline_1_search)
```

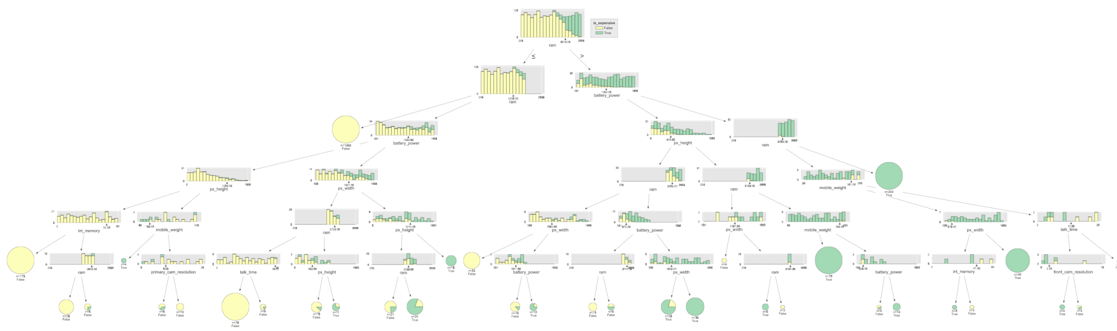
[104]:
```
    rank_test_score  mean_test_score  \
26                1           0.9475
28                2           0.9435
29                3           0.9415
27                4           0.9405
20                5           0.9400

                                                               params
26    {'dt_classifier__criterion': 'entropy', 'dt_classifier__max_depth': 6}
28    {'dt_classifier__criterion': 'entropy', 'dt_classifier__max_depth': 8}
29    {'dt_classifier__criterion': 'entropy', 'dt_classifier__max_depth': 9}
27    {'dt_classifier__criterion': 'entropy', 'dt_classifier__max_depth': 7}
20 {'dt_classifier__criterion': 'entropy', 'dt_classifier__max_depth': None}
```

[ ]:
```
print(f"Best Test Scores: {dt_pipeline_1_search.best_score_}")
print(f"Best Params: {dt_pipeline_1_search.best_params_}")
```

Comment: The (above) table shows the top 5 hyper-parameters of the decision tree, and the mean test scores of all CV folds. The best mean test score (`0.9475`) has Information Gain as a criterium, and a `decision_tree_max_depth = 6`. We see **no** test in the listed top 5 containing gini-index criterium.

4.1.1 Decision Tree Visualization

[106]:
```
viz = dtreeviz(dt_pipeline_1_search.best_estimator_[0], feature_data,␣
 ↪target_data,
                target_name="is_expensive",
                feature_names=feature_data.columns,
                class_names=list(target_data.unique()))

viz.save("decision_tree.svg");
#viz
```



Comment: The (above) diagram shows the decision tree visualisation.

4.2 Support Vector Machine

The 'SVM' algorithm finds the best hyperplane in the multi-atribute dimension, to seperate data into multiple, relevant classes. The parameter/s available in SVM are:

- `Seperation Degree` of the Hyperplane (also known as `Kernel`);

[51]:
```
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)
```

[53]:
```
show_top_results(svm_pipeline_2_search)
```

[53]:

	rank_test_score	mean_test_score	params
0	1	0.9895	{'svm_classifier__kernel': 'linear'}
1	2	0.9825	{'svm_classifier__kernel': 'poly'}
2	3	0.9785	{'svm_classifier__kernel': 'rbf'}
3	4	0.5300	{'svm_classifier__kernel': 'sigmoid'}

[54]:
```
print(f"Best Mean Test Scores of different Parameters: {svm_pipeline_2_search.
↪best_score_}")
print(f"Best Params: {svm_pipeline_2_search.best_params_}")
```

```
Best Mean Test Scores of different Parameters: 0.9894999999999999
Best Params: {'svm_classifier__kernel': 'linear'}
```

Comment: The (above) table displays the top 5 hyperparameters, of the SVM. We observe the best mean test score of (`0.9895`) attributable to the `kernel = linear` hyperplane.

4.3 Comparison `DT` vs `SVM`

Our tests conclude that SVM yields results with higher accuracies, than DT. Though both (methods) are exposed to the same data, and their most valuable hyperparameters are optimized, we must acknowledge that SVM is a more appropriate algorithm for this dataset, and experiment.

5 Clustering

Clustering is the unsupervised-learning process of assigning data to a group, or 'cluster'. Clustering identifies similarities between objects, which it groups according to these characteristics in common, and which differentiate them from other groups of data. The clustering process is very similar to the classification process, aside from the classes not being known / labelled (in clustering).

[ ]:
```
# clustering helper functions

def print_confusion_table(crosstab_df, accuracy_val):
    print(crosstab)
    print("--------------------------")
    print("Accuracy: {} ({:.2f}%)".format(accuracy, accuracy*100))
    print("--------------------------")
```

5.1 Cluster of Size 2

The clustering method will begin with `CLUSTER_SIZE = 2`, enabling us to compare the two distinct values of `is_expensive`, and the relevant clusters. This **DOES NOT** mean that the `CLUSTER_SIZE` set at 2, is the optimal cluster number.

Example: *There may be a phone that sits in between `is_expensive=True` and `is_expensive=False` (a medium priced phone) that a cluster size `n=2` would not be able to properly represent.*

5.2 Kmeans

`K-means` clustering aims to partition observations (data rows) into ***k*** clusters, in which each observation belongs to the cluster with the nearest cluster-mean / cluster-centroid.

Confusion Table

[56]:
```
kmeans_model = KMeans(n_clusters=2, random_state=RANDOM_STATE)
kmeans_model.fit(feature_data)
kmeans_prediction = kmeans_model.labels_
```

[ ]:
```
confusion_matrix = pd.DataFrame()
# target_data
confusion_matrix["target"] = target_data
confusion_matrix["prediction"] = kmeans_prediction
```

```
crosstab = pd.crosstab(confusion_matrix['target'],␣
↪confusion_matrix['prediction'])
accuracy = np.diagonal(crosstab).sum() / crosstab.to_numpy().sum()
```

[58]:
```
print_confusion_table(crosstab, accuracy)
```

```
prediction    0    1
target
False       977  523
True          0  500
--------------------------
Accuracy: 0.7385 (73.85%)
--------------------------
```

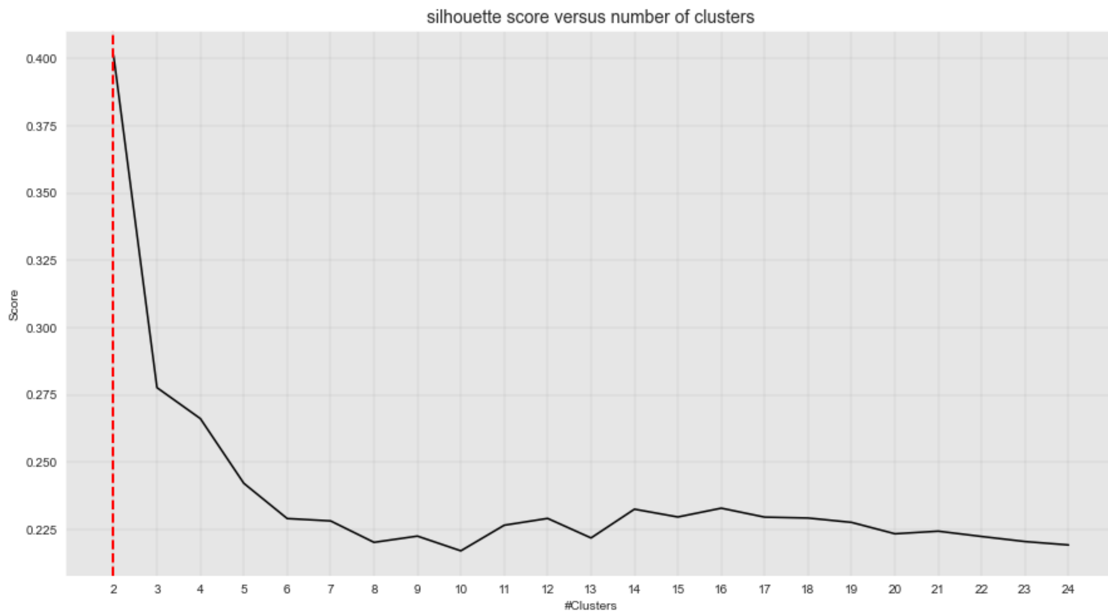
Comment: The confusion table (above) shows a `73.85%` accuracy when matched with the labelled groups, wherein the other 26.15% inaccuracy can be explained by kmeans predicting a phone `is_expensive` when it is in fact not (false positive), and 0% for false negative.

5.3 Validation of Two Cluster Size

In the previous heading, it is assumed that the cluster of size 2 is expected. This is an extra exploratory step to validate whether a cluster of size 2 is the best way to split the data.

[59]:
```
cluster_evaluation = clusteval(method='silhouette',cluster="kmeans")
cluster_evaluation.fit(np.array(feature_data))
cluster_evaluation.plot()
```

```
  4%|
| 1/23 [00:00<00:03,  6.58it/s]


[clusteval] >Fit using kmeans with metric: euclidean, and linkage: ward
[clusteval] >Evaluate using silhouette.

100%|
      | 23/23 [00:07<00:00,  2.91it/s]

[clusteval] >Optimal number clusters detected: [2].
[clusteval] >Fin.
```




```
[59]: (<Figure size 1080x576 with 1 Axes>,
        <AxesSubplot:title={'center':'silhouette score versus number of clusters'},
       xlabel='#Clusters', ylabel='Score'>)
```

5.3.1 Silhouette Score

Silhouette score refers to a method of interpretation and validation of consistency within clusters of data. The higher the score, the better the overall cluster measure, such as within-variability and between-variability of the clusters. According to the chart (above) a `CLUSTER_SIZE = 2` is most appropriate.

6 Data Reduction

More data is not always the solution. When working with 'big data' we must consider the issues around redundancy, outliers, and storage. Data reduction (as the name suggests) actively encourages the reduction of data, where possible, through the elimination of data-rows (numerosity), and data-columns (features). This reduction must be weighed in against the overall quality/information that would be forfeited.

6.1 Reasons for Reducing Data

- Increases storage capacity
- Easy and efficient Mining, reduces time and memory requirement
- Easy visualisation
- Help to eliminate irrelevant /redundant features
- Reduces noise

6.2 Numerosity Reduction

Numerosity reduction involves the replacement of voluminous data, with an alternate, smaller form of data representation. This exchange can be achieved via parametric and non-parametric methods.

- `Parametric Numerosity Reduction` -> These techniques include linear regression and log linear models, to which the model's parameters can be stored, instead of the FULL data representation.
- `Non-Parametric Numerosity Reduction` -> Sampling, histograms, clustering, data cube aggregation.

6.2.1 Numerosity Reduction - Sampling with Decision Tree

[ ]:
```
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)))
}
```

[61]:
```
sampled_learning_data = learning_data.sample(frac=0.5,random_state=RANDOM_STATE)
sampled_target_data = sampled_learning_data["is_expensive"]
sampled_feature_data = sampled_learning_data.drop("is_expensive",axis="columns")
```

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

[63]:
```
# Results in Tabular format
show_top_results(dt_pipeline_1_search)
```

[63]:
```
    rank_test_score  mean_test_score  \
0                 1            0.928
10                1            0.928
19                1            0.928
18                1            0.928
17                1            0.928

                                                                   params
0   {'dt_classifier__criterion': 'gini', 'dt_classifier__max_depth': None}
10    {'dt_classifier__criterion': 'gini', 'dt_classifier__max_depth': 10}
19    {'dt_classifier__criterion': 'gini', 'dt_classifier__max_depth': 19}
18    {'dt_classifier__criterion': 'gini', 'dt_classifier__max_depth': 18}
17    {'dt_classifier__criterion': 'gini', 'dt_classifier__max_depth': 17}
```

Comment: Though we see (above) a drop in `mean_test_score` by `1.2%` (which is not surprising), the amount of data needed for testing and training has been cut by `50%`.

6.2.2 Numerosity Reduction - Sampling with Support Vector Machine

[64]:
```
svm_pipeline_2 = Pipeline([
    ("svm_classifier",SVC())
])

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

[65]:
```
svm_pipeline_2_search = GridSearchCV(svm_pipeline_2, param_grid, n_jobs=-1,␣
↪cv=CROSS_VALIDATION_PARTITION)
svm_pipeline_2_search.fit(sampled_feature_data,sampled_target_data)
```

[65]:
```
GridSearchCV(cv=10, estimator=Pipeline(steps=[('svm_classifier', SVC())]),
             n_jobs=-1,
             param_grid={'svm_classifier__kernel': ['linear', 'poly', 'rbf',
                                                    'sigmoid']})
```

[66]:
```
# Results in Tabular format
show_top_results(svm_pipeline_2_search)
```

[66]:

	rank_test_score	mean_test_score	params
0	1	0.982	{'svm_classifier__kernel': 'linear'}
1	2	0.977	{'svm_classifier__kernel': 'poly'}
2	3	0.974	{'svm_classifier__kernel': 'rbf'}
3	4	0.548	{'svm_classifier__kernel': 'sigmoid'}

Comment: (Above) Similar to the results from the numerosity reduction with the Decision Tree, we have observed an insignificant drop in accuracy from the default SVM model (accuracy of `98.95%`) to the numerosity reduced SVM model (accuracy `98.2%`).

6.3 Attribute Reduction & Attribute Selection

Attribute reduction focuses on (no surprise) the reduction of the data's columns/attributes. Data in a high-dimensional space will be transformed (by attribute reduction methods) into a low-dimensional space, so that this new, low-dimensional representation may retain appropriate amounts of meaningful properties (from the original data), whilst **also** heavily reducing the computational-power-requirements (storage, navigation, transformation etc).

It achieves this mainly through discovering the attributes that possess the lowest 'predictive value', i.e. they contribute little to the domain specific question or are insignificant in aiding attributes that *are* contributing.

6.3.1 Principal Component Analysis (PCA)

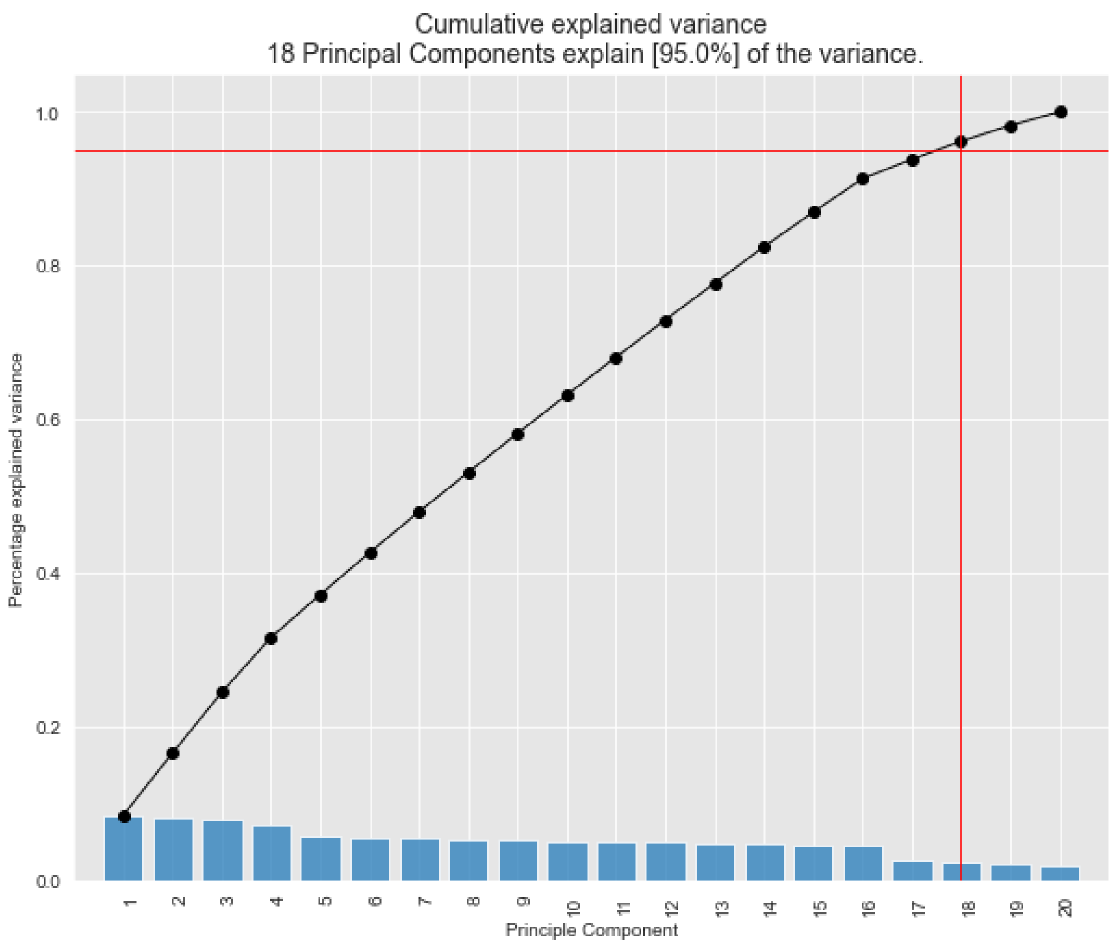
Variance -> In the field of statistics, variance is one of the most important measures, as it examines how the data varies within (internally) and inbetween (interactively) attributes.

PCA summarises the attributes of the model through linear combinations expressed as 'principal components' ('PC') that maximises the variance. The variance contribution of the principal components (to the data) are ordered such that the first principal component is the 'most valuable' or 'highest contributory' attribute. Furthermore, one of the requirements of PCA is that the data is scaled, as the linear combination is sensitive to (attribute value) ranges (values with large magnitude value ranges will dominate the other PCs).

It should be noted that although PCA does a good job of multidimensional to low-dimensional data summarisation, it makes it ***very*** difficult to determine which domain specific attribute/s is the one/s contributing to the model.

[67]:
```
model = pca(n_components=0.95) # explains 95% of variance
scaled_data = StandardScaler().fit_transform(feature_data)
results = model.fit_transform(scaled_data)
model.plot()
```

```
[pca] >Column labels are auto-completed.
[pca] >Row labels are auto-completed.
[pca] >The PCA reduction is performed to capture [95.0%] explained variance
using the [20] columns of the input data.
[pca] >Fitting using PCA..
[pca] >Computing loadings and PCs..
[pca] >Computing explained variance..
[pca] >Number of components is [18] that covers the [95.00%] explained variance.
[pca] >Outlier detection using Hotelling T2 test with alpha=[0.05] and
n_components=[5]
[pca] >Outlier detection using SPE/DmodX with n_std=[2]
```




```
[67]: (<Figure size 720x576 with 1 Axes>,
       <AxesSubplot:title={'center':'Cumulative explained variance\n 18 Principal
      Components explain [95.0%] of the variance.'}, xlabel='Principle Component',
      ylabel='Percentage explained variance'>)
```

```
<Figure size 432x288 with 0 Axes>
```

Comment: According to the total contributions, to be able to explain 95% of the data's variance, there is a requirement to retain 18 principal components. It should be noted that this is most unusual for PCA, with most PCA's having 95% of the (data's) variance explained by only a tiny subset of the data's total dimensions.

6.3.2 Attribute Reduction - 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)
```

[70]:
```
show_top_results(dt_pipeline_3_search)
```

[70]:
```
     rank_test_score  mean_test_score  \
568                1           0.8395
682                2           0.8385
567                3           0.8385
566                3           0.8385
663                5           0.8380

                 params
568   {'dt_classifier__criterion': 'entropy', 'dt_classifier__max_depth': 9,
'pca__n_components': 18}
682  {'dt_classifier__criterion': 'entropy', 'dt_classifier__max_depth': 15,
'pca__n_components': 18}
567   {'dt_classifier__criterion': 'entropy', 'dt_classifier__max_depth': 9,
'pca__n_components': 17}
566   {'dt_classifier__criterion': 'entropy', 'dt_classifier__max_depth': 9,
'pca__n_components': 16}
663  {'dt_classifier__criterion': 'entropy', 'dt_classifier__max_depth': 14,
'pca__n_components': 18}
```

[ ]:
```
print(f"Best Test Scores: {dt_pipeline_3_search.best_score_}")
print(f"Best Params: {dt_pipeline_3_search.best_params_}")
```

Comment: The results from our Decision Tree attribute reduction with PCA demonstrated a **significant** decrease in accuracy (from un-reduced accuracy of 94.75%, to 83.95%. These results come as no surprise, as per results earlier, there is an unusual contribution to total variance of PCA. We should also consider the possibility that the summarisation of attributes to principal components increased the impact that outliers exerted upon the (outlier sensitive) decision tree model.

6.3.3 Attribute Reduction - Support Vector Machine 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)
```

[74]:
```
show_top_results(svm_pipeline_3_search)
```

[74]:
```
    rank_test_score  mean_test_score  \
85                1           0.9910
80                1           0.9910
90                3           0.9895
75                4           0.9840
88                5           0.9785


                                                             params
85   {'pca__n_components': 18, 'svm_classifier__kernel': 'linear'}
80   {'pca__n_components': 17, 'svm_classifier__kernel': 'linear'}
90   {'pca__n_components': 19, 'svm_classifier__kernel': 'linear'}
75   {'pca__n_components': 16, 'svm_classifier__kernel': 'linear'}
88  {'pca__n_components': 18, 'svm_classifier__kernel': 'sigmoid'}
```

[ ]:
```
print(f"Best Test Scores: {svm_pipeline_3_search.best_score_}")
print(f"Best Params: {svm_pipeline_3_search.best_params_}")
print(f"Best CV Index: {svm_pipeline_3_search.best_index_}")
```

Comment: The results (above) show the minimal improvement in accuracy, when the SVM model undertook the attribute reduction process. SVM is more robust to noise and outliers (than Decision Trees), and so it may be possible that contrary to above, it increased the impact of the "good quality" data, rather than the outlier data.

Prior to reduction, we observed an accuracy of 98.95%, which then increased to 99.1%.

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

There are multiple ways to select the best attribute/s of a dataset, mainly through the removal of redundant (highly-corellated) or irrelevant features. The common methods include 'F-statistic', and 'Information Gain'. F-statistic iteratively picks the attribute that *maximises* the varaiance (rather than a combination of attributes, seen in PCA).

Information gain measures the entropy-information available in a probability distribution. E.g :

- Skewed Probability Distribution (unsurprising) -> Low entropy.
- Balanced Probability Distribution (surprising) -> High entropy.

6.4.1 Attribute Reduction - Decision Tree with Select K-Best

[76]:
```
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)
```

[78]:
```
show_top_results(dt_pipeline_4_search)
```

[78]:
```
     rank_test_score  mean_test_score  \
332                1           0.9515
250                2           0.9515
260                2           0.9515
256                2           0.9515
248                5           0.9510

                          params
332  {'dt_classifier__criterion': 'entropy', 'dt_classifier__max_depth': 8,
'select_kbest__k': 15, 's…
250  {'dt_classifier__criterion': 'entropy', 'dt_classifier__max_depth': 6,
'select_kbest__k': 12, 's…
260  {'dt_classifier__criterion': 'entropy', 'dt_classifier__max_depth': 6,
'select_kbest__k': 17, 's…
```

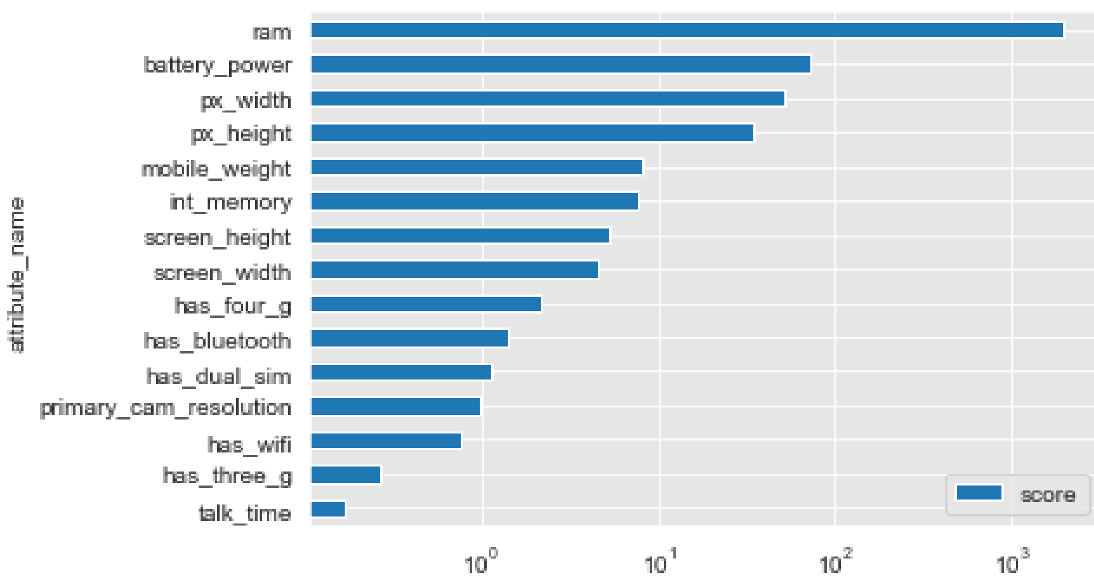
```
256  {'dt_classifier__criterion': 'entropy', 'dt_classifier__max_depth': 6,
'select_kbest__k': 15, 's…
248  {'dt_classifier__criterion': 'entropy', 'dt_classifier__max_depth': 6,
'select_kbest__k': 11, 's…
```

[ ]:
```
print(f"Best Test Scores: {dt_pipeline_4_search.best_score_}")
print(f"Best Params: {dt_pipeline_4_search.best_params_}")
print(f"Best CV Index: {dt_pipeline_4_search.best_index_}")
```

6.4.2 Attribute Selection with F-Statistic prior to Decision Tree

[80]:
```
select_k_best = dt_pipeline_4_search.best_estimator_[0]
attributes_df = pd.DataFrame({"attribute_name": feature_data.
↪columns[select_k_best.get_support()],"score": select_k_best.
↪scores_[select_k_best.get_support()]})
attributes_df.sort_values("score").
↪plot(x="attribute_name",y="score",kind="barh", logx=True)
print(f"Number of Attributes: {len(attributes_df)}")
```

```
Number of Attributes: 15
```



Comments: The table and graph (above) display the small improvement in the accuracy percentge, when the Decision Tree model experienced the 'select k-best' attribute reduction (an increase from `94.75%` to `95.15%`).

We see the reduction method has left us with 15 attributes, in comparison to the 18 attributes in PCA.

In this scenario, it appears that F-statistic is 'better' than information gain, with regard to the transformation. Note however, that information gain is later used for the decision tree classifier.

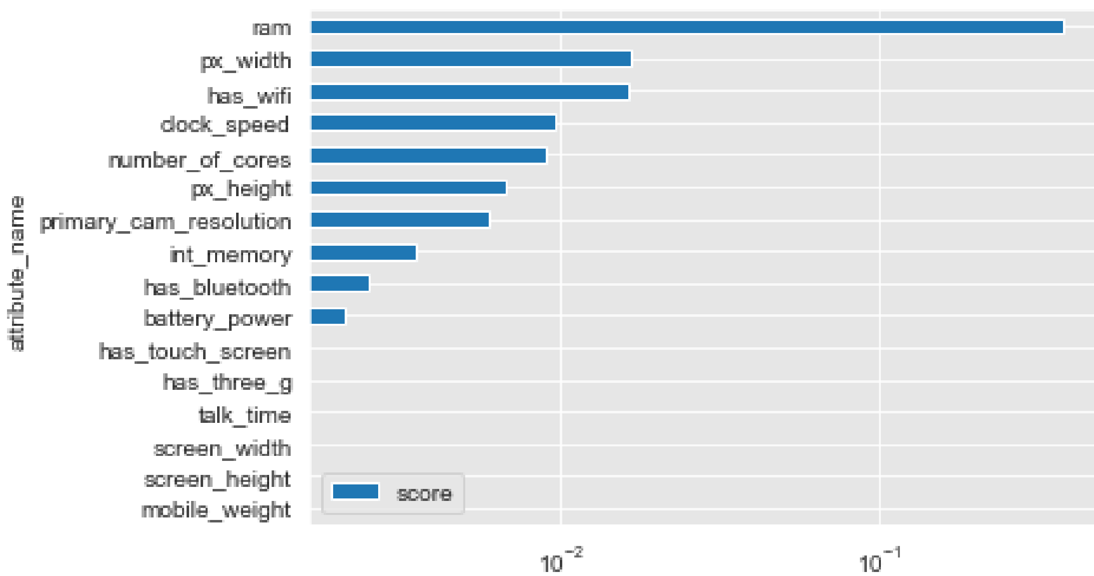
6.4.3 Attribute Selection - With Information Gain, Prior to Decision Tree

[ ]:
```
dt_pipeline_4 = Pipeline([
    ("select_kbest", SelectKBest()),
    ("dt_classifier",DecisionTreeClassifier(random_state=RANDOM_STATE))
])
# Grid Search - REDUCED BECAUSE IT IS TAKING A LONG TIME
param_grid = {
    'select_kbest__k': range(10,18),
    'select_kbest__score_func': [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)
```

[83]:
```
select_k_best = dt_pipeline_4_search.best_estimator_[0]
attributes_df = pd.DataFrame({"attribute_name": feature_data.
 ↪columns[select_k_best.get_support()],"score": select_k_best.
 ↪scores_[select_k_best.get_support()]})
attributes_df.sort_values("score").
 ↪plot(x="attribute_name",y="score",kind="barh", logx=True)
print(f"Number of Attributes: {len(attributes_df)}")
```

```
Number of Attributes: 16
```



Comment: The analysis above shows that F-statistic proved to be the method that makes the accuracy of the model higher (`94.90%`). This is an improvement, but not as 'good' as the improvement experienced with F-statistic. However, for exploratory, examining the attributes there are differences in the attributes prioritised by information gain and F-statistic.

6.4.4 Attribute Reduction - Support Vector Machine with Select K-Best

[84]:
```
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)
```

[86]:
```
show_top_results(svm_pipeline_4_search)
```

[86]:

```
    rank_test_score  mean_test_score  \
0                 1           0.9915
4                 1           0.9915
12                3           0.9910
8                 4           0.9905
16                4           0.9905

                          params
0   {'select_kbest__k': 10, 'select_kbest__score_func': <function f_classif at
0x000002443639F4C0>, …
4   {'select_kbest__k': 11, 'select_kbest__score_func': <function f_classif at
0x000002443639F4C0>, …
12  {'select_kbest__k': 13, 'select_kbest__score_func': <function f_classif at
0x000002443639F4C0>, …
8   {'select_kbest__k': 12, 'select_kbest__score_func': <function f_classif at
0x000002443639F4C0>, …
16  {'select_kbest__k': 14, 'select_kbest__score_func': <function f_classif at
0x000002443639F4C0>, …
```

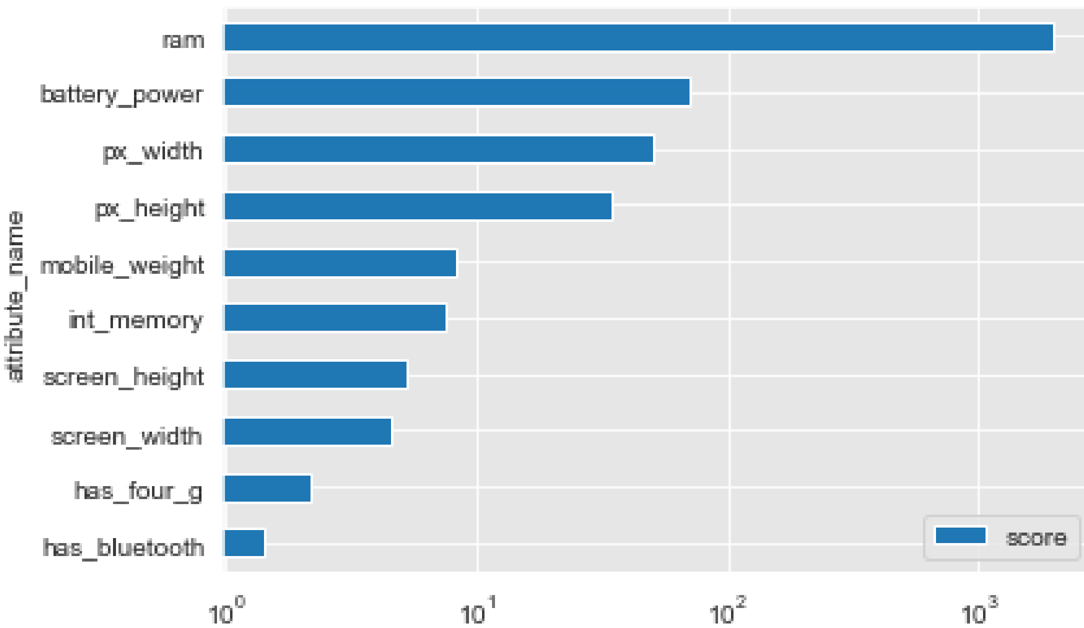
[ ]:

```
print(f"Best Test Scores: {svm_pipeline_4_search.best_score_}")
print(f"Best Params: {svm_pipeline_4_search.best_params_}")
print(f"Best CV Index: {svm_pipeline_4_search.best_index_}")
```

[88]:

```
select_k_best = svm_pipeline_4_search.best_estimator_[0]
attributes_df = pd.DataFrame({"attribute_name": feature_data.
 ↪columns[select_k_best.get_support()],"score": select_k_best.
 ↪scores_[select_k_best.get_support()]})
attributes_df.sort_values("score").
 ↪plot(x="attribute_name",y="score",kind="barh", logx=True)
print(f"Number of Attributes: {len(attributes_df)}")
```

```
Number of Attributes: 10
```



Comment: The output (above) indicates a minor accuracy improvement, from `98.95%` to `99.1%`. Note that this increase is `0.04%` better than the observed accuracy for SVM, with PCA.

We also observe that the suggested best number of attributes (for SVM) from F-statistic is `10`, a notably smaller value than the suggested 'best number' of attributes from F-statistic for Decision Tree (`15`).

6.5 Data Reduction - Conclusion

As seen in our experiments above (with our chosen measure of *classifier-accuracy* as accuracy), numerosity reduction **heavily** decreases the required data, in exchange for a small decrease in overall model accuracy. Attribute reduction via transformation (such as PCA) is seen in varying instances to both increase / decrease the model accuracy, while attribute reduction (by selection) demonstrated an overall net increase in accuracy . Furthermore, with attribute reduction, we also observed a decrease in required model-training time.

6.6 Attribute Selection - Manual Selection

With user-side specific domain knowledge (regarding expensive phones), we manually selected the following attributes. Note that the accuracy of this "model" is heavily dependent on the depth of the user's domain specific knowledge. In most real-world data science projects, manual attribute selection (performed by subject matter experts), mitigates the possibility of over-fitting, that arises with pure criterium-based attribute selection.

For example, in electrical-engineering problems (where the voltage attribute is crucial) the data may suggest that voltage could/should be eliminated (for the purpose of increasing accuracy). If relied upon, the resultant model has the 'best' accuracy, but it may inappropriate and/or unusable in the real world.**Comment:** When the Decision Tree was reduced with our manually-selected-attribute model, we note a relatively large drop in accuracy from 94.75% to 92.35% (2.4%).

[89]:
```
# Based on domain-specific knowledge and the exploratory plots
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"]

feature_data_manually_selected = learning_data.
 ↪drop("is_expensive",axis="columns")[MANUALLY_SELECTED_ATTRIBUTES]
```

6.7 Decision Tree with Manually Selected Attribute

[ ]:
```
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)
```

[92]:
```
show_top_results(dt_pipeline_1_search)
```

[92]:
```
    rank_test_score  mean_test_score  \
4                 1           0.9235
2                 2           0.9215
1                 2           0.9215
3                 4           0.9205
25                5           0.9175

                                                              params
4      {'dt_classifier__criterion': 'gini', 'dt_classifier__max_depth': 4}
2      {'dt_classifier__criterion': 'gini', 'dt_classifier__max_depth': 2}
```

```
1     {'dt_classifier__criterion': 'gini', 'dt_classifier__max_depth': 1}
3     {'dt_classifier__criterion': 'gini', 'dt_classifier__max_depth': 3}
25 {'dt_classifier__criterion': 'entropy', 'dt_classifier__max_depth': 5}
```

[ ]:
```
print(f"Best Test Scores: {dt_pipeline_1_search.best_score_}")
print(f"Best Params: {dt_pipeline_1_search.best_params_}")
```

Comment: When the Decision Tree was reduced with our manually-selected-attribute model, we note a relatively large drop in accuracy from `94.75%` to `92.35%` (`2.4%`).

6.8 SVM with Manually Selected Attributes

[ ]:
```
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)
```

[96]:
```
show_top_results(svm_pipeline_2_search)
```

[96]:

	rank_test_score	mean_test_score	params
2	1	0.9295	{'svm_classifier__kernel': 'rbf'}
0	2	0.9265	{'svm_classifier__kernel': 'linear'}
1	2	0.9265	{'svm_classifier__kernel': 'poly'}
3	4	0.5190	{'svm_classifier__kernel': 'sigmoid'}

[ ]:
```
print(f"Best Test Scores: {svm_pipeline_2_search.best_score_}")
print(f"Best Params: {svm_pipeline_2_search.best_params_}")
```

Comment: When the SVM was reduced with our manually-selected-attribute model, we observe a notably large drop in accuracy from `98.95%` to `92.95%`. This decrease in accuracy with SVM (`6.0%`) is much larger than the decrease experienced by Decision Tree.

6.9 Attribute Selection - Conclusion

Through our experiments, F-statistic attribute selection demonstrated the best overall accuracy for both SVM and Decision Tree, as defined by the hyperparameter grid search.

Our results would indicate that Information Gain ranks second in accuracy, increasing both SVM and DT's accuracy. This can be contrasted with manual attribute selection, which showed poor results, with decreases in the accuracy of **both** of our classifiers.

7 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

[ ]:
```
# 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
```

Discretised Data Profiling Report

[ ]:
```
# 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
```