Data Mining: Association Analysis and Outlier detection

Background:

Data mining is a process to extract useful information from a vast amount of data. It is used to discover new, accurate and useful patterns in data, looking for meaning.

Data: refers to characteristics /numerical/categorical which are collected through observation.

Datasets: collections of items which are described by a set of attributes.

Data Mining starts with collection or sourcing of raw data. Raw data can have multiple issues like poor data quality such as noisy data, dirty data, missing values, inexact or incorrect values, inadequate data size and poor representation in data sampling.

Association Analysis:

Goal: To come up with a set of rules that can capture an association between items or events. The rules are used to determine when items or events occur together.

Q1:
Apriori algorithm there are 3 components to it:

- Confidence: computes the reliability of rule how often X and Y are true in data.
- Support: It is used to prune/remove itemsets that do not occur frequently.
- Lift: Increase in ratio of an itemset when put in combination with other item. It is computed as (Confidence (itemsets)/ (Support (1-itemset))

Steps to implement Apriori algorithm:

- C: Candidate item set of size k
- L: Frequent item set of size k

Join step: C_k is generated by joining L_{k-1} with itself
Prune step: Any k-1 itemset that is not frequent cannot be a subset of a frequent k-itemset.

The algorithm begins from set of frequent 1-itemsets and then moving forward to find next set of frequent k-itemsets. The join step provides us with combinations of candidate itemsets that are differently ordered and are pruned from the search tree. The non-frequent candidates are further pruned from the k-itemsets as there is not enough support or the subset is not frequent. Therefore, Apriori algorithm is used to improve efficiency of level wise generation of frequent itemsets to reduce search space.

Q2:
L1 is used to create frequent k-itemsets, and the children are created using the join operation within the parent itemset. If all but the last item of the itemsets are the same, the Apriori Algorithm joins two sets from L(k-1) at each join step, merging the last item of the (k-1) itemset. As a result, the merge operation's children become the parent's superset. Going backwards from Lk to L1, each child is the superset of the parent in turn, and 1-itemset is the root parent. As a result, each frequent k-itemset is a superset of an L1 itemset. Within the parent itemset, the search can branch out with possible join combinations.

In terms of frequency, the parent's members are frequent at each stage, and non-frequent children developed from parent itemsets are pruned. Children itemsets (if parent is at level k-1; children will be at level k) will also obey the Apriori property because the children are closed under the frequentness

rule. The items inside the (children) k-itemset will follow the parent k-1 itemset with the new item added at level k.

Making a superset from the parent will propagate to older ancestors using induction; and because L1 is the root of this search tree, every k-itemset will propagate to L1. Even though k-itemset does not subset itself, they are built from the root L1, this will work for any branch. As a result, each k-itemset is a superset of an L1 itemset.

Q3:

$L2$={{1,2},{1,4},{2,3},{2,4},{3,5}}

$C3$ = {{1,2,4},{2,3,4}}

Q4:

Formula to compute Association rule:

$$S_{\mathcal{A}\Rightarrow\mathcal{B}} = S_{\mathcal{A}\cup\mathcal{B}} = \frac{N_{\mathcal{A}\cup\mathcal{B}}}{N} = \frac{1}{N}\sum_{i=1}^{N}\mathbb{I}\left[(\mathcal{A}\cup\mathcal{B})\subseteq\mathcal{T}_i\right],$$

Figure 1. Support of an Association Rule

S1: support of the association rule {boarding pass, passport}⇒{flight}.

Confidence rule: support({boarding pass, passport, flight})
support({boarding pass, passport})

As per S1 it can be noticed that flight is associated with boarding pass and passport. The boarding pass, passport is subset of boarding pass. If assumed that the support of the (boarding pass, passport) itemset is less than or equal to the support of the (passport) itemset due to Apriori property.

S2: support of the association rule {boarding pass}⇒{flight}

Confidence rule: support({boarding pass, flight})
support({boarding pass})

As per S2 it can be noticed that flight is associated with boarding pass.

In the question, there is no indication of support count . In terms of sets, however, because boarding pass is a subset of boarding pass, passport, every association with boarding pass, passport will implicitly include boarding pass association. As a result, S2 will receive more support than S1 in terms of resources. The information held by subsets (parents) is more generalised than that held by supersets (children). The fewer items in an itemset, the more individual transactions can support it. As itemsets grow in size, the likelihood of seeing a transaction involving all of the items in the itemset decreases, resulting in less support. Supersets have their subsets within them in association analysis, which gives implicit information about association about the subsets as well.

Q5:

The empty set is the antecedent, and "Eggs" is the consequent. "What is the support of "Eggs" in 1-itemset?" is an equivalent question to this one. As seen in cell 4 (or cell 5), the “Eggs” 0.8. It's worth noting that the cell numbers refer to running the entire tutorial script in a new notebook.

Q6:

With the adjustment of putting min_support = 0.2 in cell 5, the maximum length of frequent itemset becomes 6 : (Yogurt, Onion, Kidney Beans, Nutmeg, Eggs, Dill) and (Yogurt, Onion, Kidney Beans, Milk, Nutmeg, Eggs). 149 frequent itemsets are generated. As seen in the figure below:

```
frequent_itemsets = apriori(df, min_support=0.2, use_colnames=True)
display(frequent_itemsets)
```

	support	itemsets
0	0.2	(Apple)
1	0.4	(Corn)
2	0.2	(Dill)
3	0.8	(Eggs)
4	0.2	(Ice cream)
...	...	...
144	0.4	(Yogurt, Onion, Kidney Beans, Nutmeg, Eggs)
145	0.2	(Yogurt, Onion, Milk, Nutmeg, Eggs)
146	0.2	(Yogurt, Onion, Kidney Beans, Milk, Nutmeg)
147	0.2	(Yogurt, Onion, Kidney Beans, Nutmeg, Eggs, Dill)
148	0.2	(Yogurt, Onion, Kidney Beans, Milk, Nutmeg, Eggs)

149 rows × 2 columns

Q7:

```
def cal_kulczynski(itemset, antecendents, consequents):

    rules = association_rules(itemset, metric="confidence", min_threshold=0)

    # Forward & Backward Rule:- Assumed always to return a strong rule
    fwd_rule = rules[(rules['antecedents'] == antecendents) & (rules['consequents'] == consequents)]
    bck_rule = rules[(rules['antecedents'] == consequents) & (rules['consequents'] == antecendents)]

    # Confidences:
    conf_fwd = fwd_rule['confidence'].values[0]
    conf_bck = bck_rule['confidence'].values[0]
    kulc = (conf_bck + conf_fwd) / 2
    return kulc

ants = frozenset(['Onion'])
cons = frozenset(['Kidney Beans', 'Eggs'])
print('Test 1: ' + str(cal_kulczynski(frequent_itemsets, ants, cons)))
```

Output:

```
[53] def cal_kulczynski(itemset, antecendents, consequents):

         rules = association_rules(itemset, metric="confidence", min_threshold=0)

         # Forward & Backward Rule: - Assumed always to return a strong rule
         fwd_rule = rules[(rules['antecedents'] == antecendents) & (rules['consequents'] == consequents)]
         bck_rule = rules[(rules['antecedents'] == consequents) & (rules['consequents'] == antecendents)]

         # Confidences:
         conf_fwd = fwd_rule['confidence'].values[0]

         conf_bck = bck_rule['confidence'].values[0]

         kulc = (conf_bck + conf_fwd) / 2

         return kulc

     ants = frozenset(['Onion'])
     cons = frozenset(['Kidney Beans','Eggs'])
     print('Kulczynski measure: ' + str(cal_kulczynski(frequent_itemsets, ants, cons)))

     Kulczynski measure: 0.875
```

Q8:

```
def cal_imbal_ratio(itemset, antecendents, consequents):

    ab_set = frozenset.union(antecendents,consequents)

    a = itemset[itemset['itemsets'] == antecendents]
    b = itemset[itemset['itemsets'] == consequents]
    ab = itemset[itemset['itemsets'] == ab_set]

    supporta = a['support'].values[0]
    print(supporta)
    supportb = b['support'].values[0]
    print(supportb)
    supportab = ab['support'].values[0]
    print(supportab)

    return abs(supporta-supportb) / (supporta+supportb-supportab)

ants = frozenset(['Onion'])
cons = frozenset(['Kidney Beans','Eggs'])
print('Test 1: ' + str(cal_imbal_ratio(frequent_itemsets, ants, cons)))
```

Output:

Outlier detection:

An outlier is an object that deviates significantly from the rest of the objects. They can be caused by measurement or execution errors.

Global Outliers:
They are also known as Point Outliers. These are the simplest form of outliers. If, in a given dataset, a data point strongly deviates from all the rest of the data points, it is known as a global outlier. Mostly, all of the outlier detection methods are aimed at finding global outliers.

Collective Outliers:
As the name suggests, if in a given dataset, some of the data points, as a whole, deviate significantly from the rest of the dataset, they may be termed as collective outliers. Here, the individual data objects may not be outliers, but when seen as a whole, they may behave as outliers. To detect these types of outliers, we might need background information about the relationship between those data objects showing the behaviour of outliers.

Contextual (Conditional) Outliers:
In the same context when the data point deviates significantly from the rest of the data points , it is considered as a contextual outlier. It means that the same value cannot be considered as an outlier if it has occurred in different contexts. The application of contextual outliers can be found in time series data where the context is always almost temporal as the records are of a specific quantity over time.

The example for contextual outlier has been demonstrated in the Q1 below:

Q1:

Contextual outliers in credit card fraud detection:

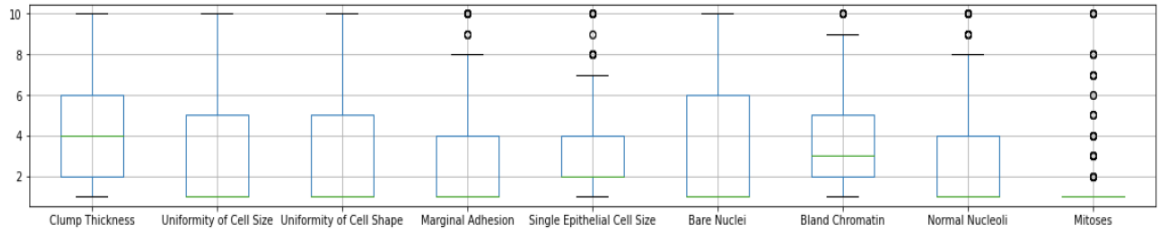
Examples	Types of attributes		Explanation
	Contextual	Behavioural	
1	Transaction time	Transaction amount	The amount of transaction is considered behavioural attribute to identify if a transaction is an outlier or not. To check if the transaction of huge amount will be done either in the middle of night or in a later time of day.
2	Transaction type	Date of Transaction	Depending on the needs of the customers, the usage of cards varies as per seasonality. A usual example would be of: in September: school, Holiday season: tourism, winter period: heating. Here, if a certain type of costs occurs as not expected then it would be considered as an outlier.

Note:- The attributes can be combined as this is not an exhaustive list.

Q2:
An univariate approach could be used if the outliers are going to be detected within the context of an attribute as seen in Week 3 lab. In that the outliers were detected by fitting the samples of that attribute to normal distribution and samples were considered as outliers if the assigned samples are 3 or greater than 3 standard deviations away. It is shown in box plots below. As a pre-processing step, the rows with outliers are dropped later on. In case all attributes are used to determine the outliers, then this multivariate outliers problem can be turned into univariate outlier using Chi-squared statistic(under assumption of normal distribution). The rest is followed by the univariate outlier analysis.

```
data2 = data.drop(['Class'],axis=1)
data2['Bare Nuclei'] = pd.to_numeric(data2['Bare Nuclei'])
data2.boxplot(figsize=(20,3))
```

```
/usr/local/lib/python3.7/dist-packages/numpy/core/_asarray.py:83: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which i
  return array(a, dtype, copy=False, order=order)
<matplotlib.axes._subplots.AxesSubplot at 0x7f0d4c4b2fd0>
```



Q3:

To extract the outliers the univariate statistical approaches has been used in this case. With the proper examination of data the outlier is revealed. It is the value "3.25" and is (June) belonging to dataset's sixth item. After normalization, the range of 3 standard deviations is presumably to be typical. The results outside this range are deemed to be outliers. In the sixth item, value -3.14 relates to June and is more than 3 standard deviations distant. Thus, June might be considered as an outlier.

#CODE

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats

#Construct the dataset:
rainfall_data = np.array([22.93, 20.69, 25.75, 23.84, 25.34, 3.25, 23.55, 28.28, 23.72, 22.42, 26.83, 23.82])

#Data assumed to fit normal distribution - Use MLE to identify outlier values:
mu = np.mean(rainfall_data)
sigma = np.std(rainfall_data)

Z_data = (rainfall_data - mu)/sigma
print('All values: ')
print(Z_data)
```

Output:

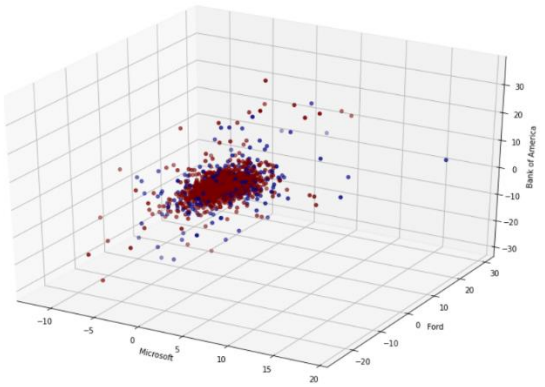
```
#Q3

import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats

# Construct the dataset:
rainfall_data = np.array([22.93, 20.69, 25.75, 23.84, 25.34, 3.25, 23.55, 28.28, 23.72, 22.42, 26.83, 23.82])

# Data assumed to fit normal distribution - Use MLE to identify outlier values:
mu = np.mean(rainfall_data)
sigma = np.std(rainfall_data)

Z_data = (rainfall_data - mu)/sigma
print('All values: ')
print(Z_data)# 5th index is an outlier
```

```
All values:
[ 0.06443672 -0.30097656  0.52446592  0.21288586  0.45758224 -3.14597989
  0.16557789  0.93718716  0.19331015 -0.01876006  0.70064732  0.20962324]
```

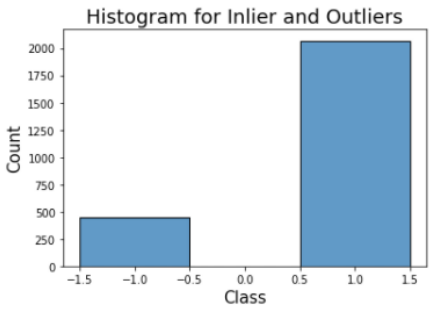
Q4:

```
#Code
import pandas as pd
import numpy as np
from sklearn.svm import OneClassSVM
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
%matplotlib inline
# Load CSV file, set the 'Date' values as the index of each row, and display the first rows of the dataframe
stocks = pd.read_csv('stocks.csv', header='infer')
stocks.index = stocks['Date']
stocks = stocks.drop(['Date'],axis=1)
stocks.head()
# Percentage change in daily closing price:
N,d = stocks.shape
# Compute delta, which denotes the percentage of changes in the daily closing price of each stock
stocks = pd.DataFrame(100*np.divide(stocks.iloc[1:,:].values-stocks.iloc[:N-1,:].values, stocks.iloc[:N-1,:].values),
                      columns=stocks.columns, index=stocks.iloc[1:].index)
stocks.head()
# Train one class classifier:
svm = OneClassSVM(nu=0.01,gamma='auto')
outliers = svm.fit_predict(stocks) # Perform fit on input data and returns labels for that input data.
# print(outliers) # Print labels: -1 for outliers and 1 for inliers.
fig = plt.figure(figsize=(15,10)).gca(projection='3d')
fig.scatter(stocks.MSFT,stocks.F,stocks.BAC,c=outliers,cmap='jet')
fig.set_xlabel('Microsoft')
fig.set_ylabel('Ford')
fig.set_zlabel('Bank of America')
plt.show()
import seaborn as sns
#print(outliers) # Print labels: -1 for outliers and 1 for inliers.
# Plot the histogram:
sns.histplot(outliers, discrete=True)
plt.title('Histogram for Inlier and Outliers', fontsize=18)
plt.xlabel('Class', fontsize=15)
plt.ylabel('Count', fontsize=15)
print('Outlier Frequency Count (Divided to Total Amount): ', str(np.sum(outliers == -1)/len(outliers)))
print('Inlier Frequency Count (Divided to Total Amount): ', str(np.sum(outliers == 1)/len(outliers)))
```

Output:

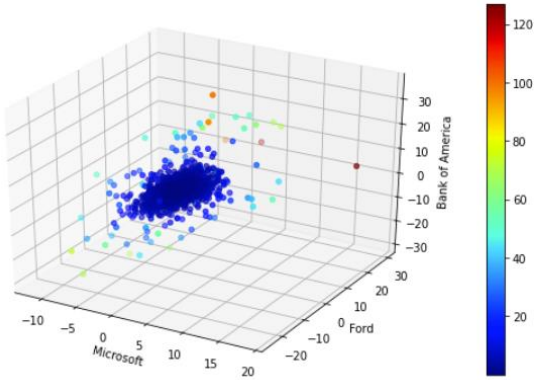



```
Outlier Frequency Count (Divided to Total Amount):  0.17798967024235202
Inlier Frequency Count (Divided to Total Amount):  0.822010329757648
```

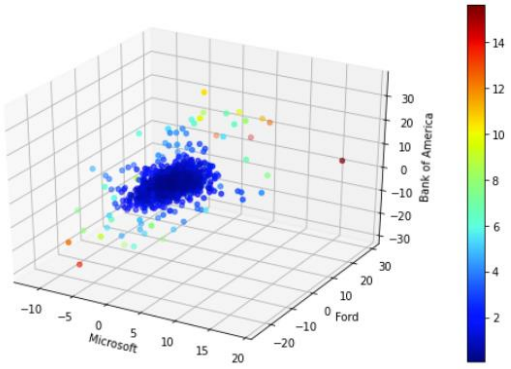


For parametric approaches, a multivariate Gaussian distribution is used. For proximity-based methods, the nearest neighbour is used in Sections 1 and 2. Both strategies employ distance measures. Because of the multiple measurements used, the results are slightly different, but the top outliers and general trend are the same. These outlier classifications are continuous and based on distances. In a one-class SVM technique, the classes are binary, i.e. inlier and outlier. For the data points displayed, there is no outlier score. In this dataset, 82 percent of the data points are classified as inliers, while 18 percent are classified as outliers. The outliers and inliers in the dataset, as well as their frequency, are depicted in the histogram above.

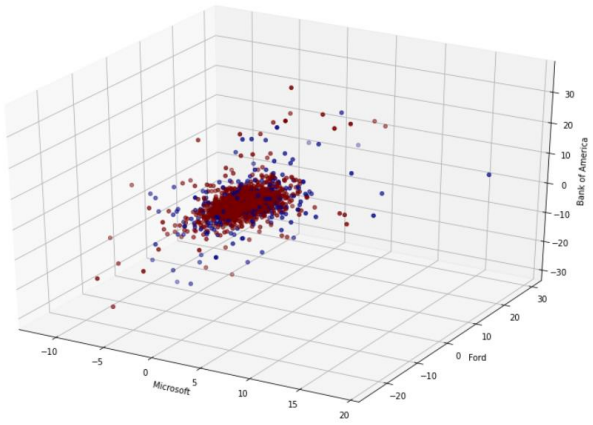
3-D Scatterplots:



Multivariate Gaussian



Nearest Neighbor



Support Vector Machine (on the left)

Despite the fact that both the Multi Variate Gaussian and Nearest Neighbour approaches use distance from the determined metrics, the results are nearly identical, with minor differences due to the distance metrics used. In both approaches, outliers are scattered around the perimeter, while inliers are concentrated in the centre.

For inliers and outliers, the Support Vector Machine uses discrete classification, and the inliers follow a similar pattern. Red data points are inliers, while blue data points are outliers. More extensive distances correspond to outlier names in the Support Vector Machine method in the first two approaches, but lower reaches fit inliers.

On the other hand, Gaussian and NN approaches provide metrics for determining whether or not a data point is an outlier. In SVM, inliers and outliers are distinguished.

Q5:

#Code

```
from pandas import read_csv

# Loading the dataset
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.csv'
df = read_csv(url, header=None)

# Extracting the values from the dataframe
data = df.values

# Split dataset into input and output elements
X, y = data[:, :-1], data[:, -1]

# Summarize the shape of the dataset
print(X.shape, y.shape)

# Normalize the data:
mu = np.mean(X, axis=0) # normalize over samples/rows
sigma = np.std(X, axis=0) # normalize over samples/rows

Z = (X-mu)/sigma
# Apply PCA:
from sklearn.decomposition import PCA

numComponents = 2
pca = PCA(n_components=numComponents)
pca.fit(Z)

project_h = pca.transform(Z)
project_h = pd.DataFrame(projected,columns=['pc_1','pc_2'],index=range(1,len(Z)+1))

display(project_h)

from sklearn.neighbors import NearestNeighbors
import numpy as np
from scipy.spatial import distance
import matplotlib.pyplot as plt

# k-nearest neighbour approach using k=2 neighbours
knn = 2
nbrs = NearestNeighbors(n_neighbors=knn, metric=distance.euclidean).fit(projected)
distances, indices = nbrs.kneighbors(projected)

# outlier score is set as the distance between the point and its k-th nearest neighbour
outlier_score = distances[:,knn-1]

# Plot scatterplot of outlier scores
fig = plt.figure(figsize=(10,5))
p = plt.scatter(projected['pc_1'], projected['pc_2'], c=outlier_score, cmap='jet')
```

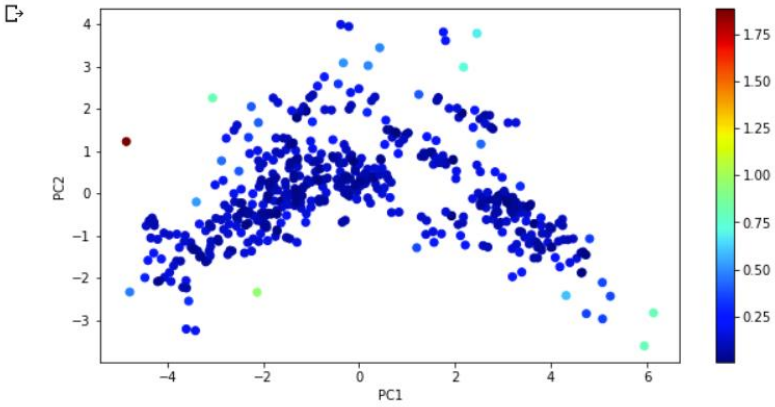
```
plt.xlabel('PC1')
plt.ylabel('PC2')
fig.colorbar(p)
plt.show()
```

Output:

```
(506, 13) (506,)
```

	pc_1	pc_2
1	-2.098297	0.773113
2	-1.457252	0.591985
3	-2.074598	0.599639
4	-2.611504	-0.006871
5	-2.458185	0.097712
...	...	...
502	-0.314968	0.724285
503	-0.110513	0.759308
504	-0.312360	1.155246
505	-0.270519	1.041362
506	-0.125803	0.761978

506 rows × 2 columns



References :

https://www.geeksforgeeks.org/types-of-outliers-in-data-mining/