Queen Mary University of London

Department of Electrical Engineering & Computer Science

ECS766P Data Mining – Assignment 3

Hasan Emre Erdemoglu

24 November 2020

Table of Contents:

Any code within this report could be provided upon request.

Part 1: Nov 3, 2020

Question 1.1:

Apriori algorithm is a Breadth-First Search algorithm, starting from set of frequent 1-itemsets; sequentially finding candidate frequent k-itemsets, where k is starting from 2, up to the cardinality of the items.

Join operation effectively gives combinations of candidate itemsets[1]; therefore, different ordered, yet same itemsets are pruned from the search tree. These candidates are further pruned by eliminating the non-frequent candidate k-itemsets. This happens either if the k-itemset does not have enough support or the subset of that subset is not frequent. The successful candidates are used to generate (k+1)-itemset candidates, which the process goes up until no more frequent itemsets are found or the cardinality is reached.

Effectively, this algorithm reduces the search space by removing redundancies and removing undesirable branches in the search process. Computing the support of every subset would include every possible permutation of the itemsets at each depth of the tree which is redundant. Additionally, the search will expand on every node at depth D even the parent of that node is not frequent. Hence Apriori is faster and more efficient than brute for approach.

Question 1.2:

Building up frequent k-itemsets are like building search trees. It starts with L_1 and the children is developed by using join operation within the parent itemset. At each join step, Apriori Algorithm joins two sets from L_{k-1}, merging the last item of the (k-1) itemset, if all but last item of the itemsets are the same. Therefore, the children of this merge operation become the superset of the parent. Going all the way from L_k back to L_1; sequentially each child is the superset of the parent and 1-itemset is the root parent. Therefore, every frequent k-itemset is a superset of an itemset in L_1. Note that the search can branch out with possible join combinations within the parent itemset.

In terms of frequentness, at each stage; the members of the parent are frequent and non-frequent children developed from parent itemsets are pruned. As the children are closed under the frequentness rule, children itemsets (if parent is at level k-1; children will be at level k) will also obey Apriori property[2], the items inside the (children) k-itemset will follow from the parent k-1 itemset with the new item added at level k.

Using induction; for a branch; making a superset from the parent will propagate to older ancestors; and as L_1 is the root of this search tree; every k-itemset will propagate to L_1. This will work for every branch; even though k-itemset do not subset themselves; they are build starting from the root L_1. Hence every k-itemset will become a superset of an itemset in L_1.

[1] K-itemset candidates are developed from (k-1)-itemsets.
[2] All non-empty subsets of a frequent itemset must be also frequent.

Question 1.3:

The following frequent 2-itemset is given:

$$L_2 = \{\{1,2\},\{1,5\},\{2,3\},\{3,4\},\{3,5\}\}$$

The candidates are generated using Join operation, which requires the sets to have all the same terms except for the last term. The candidates are shown below:

$$C_3 = \{\{1,2,5\},\{3,4,5\}\}$$

Note that {2,3,4} is not valid , as {3,4} does not fit to the definition.

Question 1.4:

S_1 states that popcorn and soda frequently imply a movie. S_2 states that popcorn frequently imply a movie. Note that {popcorn, soda} itemset is a superset of {popcorn}. Due to Apriori property it is expected that the support of {popcorn, soda} itemset is less than or equal to {soda} itemset.

The support of an association rule can be calculated using the following formula:

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

Figure 1. Support of an Association Rule

The question does not indicate the support count of the union of transactions {popcorn} and {popcorn, soda} with the itemset {movie}. However, in terms of sets, as {popcorn} is a subset of {popcorn, soda}; therefore, every association with {popcorn, soda} will also include {popcorn} association implicitly. Therefore, in terms of support, S_2 will have more support than S_1.

Subsets (parents) hold more generalized information than the supersets (children). Less items within an itemset; more support it can get from individual transactions. As itemsets get more populated; the change of seeing a transaction involving all the items within the itemset is reduced; therefore, there is less support. In association analysis, supersets also have their subsets within them, which also gives implicit information about association about the subsets as well.

Also, the slide 11 in Week 7 lecture notes this in “support of subsets” section.

Question 1.5:

Antecedent is empty set and consequent is “Kidney Beans”. An equivalent question to this question is “What is the support of “Kidney Beans” in 1-itemset?” Using cell 4 (or cell 5); it is seen that the support for “Kidney Beans” is 1.0. Note that the cell numbers are given with respect to running the entire tutorial script with a fresh notebook.

Question 1.6:

This is given in cell 5. Adjusting *min_support = 0.2;* the maximum length of frequent itemset becomes 6 with (Eggs, Nutmeg, Onion, Yogurt, Kidney Beans, Milk). Running Apriori on the data; 149 frequent itemsets are generated (From L_1 to L_6).

Question 1.7:

To answer this question a method called "calculate_kulczynski" is written. It takes a dataframe with frequent itemsets and two frozen sets for a strong association rule. The following code is written:

```
def calculate_kulczynski(itemset, antecendents, consequents):
    # We don't need to set a threshold as the question states that we would pick
    # strong rules by ourselves. (Otherwise we could set threshold; give one set of
    #antecendents and 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]
    #print(conf_fwd)
    conf_bck = bck_rule['confidence'].values[0]
    #print(conf_bck)

    kul = (conf_bck + conf_fwd) / 2
    #print(kul)

    return kul
```

Figure 2. Kulczynski Measure Implementation

Using the dataframe of frequent itemsets; association rules are calculated. Here, the threshold is set to zero as the question implies that the antecedents and consequents will be strong rules in both directions by default. Therefore, no rule checking code is implemented.

This code queries the rules in forward in backward directions. As association_rules of mlxtend calculates the confidence by itself; the confidence value of respective rules is directly used. Kulczynki measure is basically the average of confidence of antecedents given consequents and consequents given antecedents.

Two test examples are used, which is known from the tutorial data where the both forward and backward rules are strong, these are {Eggs} → {Onion} and {Eggs, Corn} → {Ice cream}. The following are the results:

```
# Do two tests:

# Test 1: (eggs -> onion) - Both ways are strong! 0.7
ants = frozenset(['Eggs'])
cons = frozenset(['Onion'])
print('Test 1: ' + str(calculate_kulczynski(frequent_itemsets, ants, cons)))

# Test 2: (eggs, corn -> ice cream) - Both ways are strong! 0.7
ants = frozenset(['Eggs', 'Corn'])
cons = frozenset(['Ice cream'])
print('Test 2: ' + str(calculate_kulczynski(frequent_itemsets, ants, cons)))
```

```
Test 1: 0.875
Test 2: 1.0
```

Figure 3. Testing Kulczynski Measure Method

If the confidence values of the association rules are inspected (using the tutorial outputs), the output of this method is the same with manual calculation. For example for test 1; the following are the results:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
15	(Eggs)	(Onion)	0.8	0.6	0.6	0.75	1.25	0.12	1.6
16	(Onion)	(Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	inf

Figure 4. Output of association_rules() method for Eggs and Onions

If Kulczynski measure is calculated manually via confidence column listed in Figure 4; the score is indeed the same with the function's output. Kulczynski measure is basically the average of confidence of antecedents given consequents and consequents given antecedents.

Question 1.8:

The code for calculating the imbalance ratio is given below. For consistency, the same test rules will be used for this part as well.

```
def calculate_imbalance_ratio(itemset, antecendents, consequents):
    """
    Takes a DataFrame of frequent itemsets and a strong association rule (Antacedents & Consequents).
    Returns imbalance ratio.
    """

    ab_set = frozenset.union(antecendents,consequents)

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

    # Actually calculating the N factor is not necessary
    # as it will be gone when calculating the ratio.
    support_a = a['support'].values[0] * len(itemset)
    #print(support_a)
    support_b = b['support'].values[0] * len(itemset)
    #print(support_b)
    support_ab = ab['support'].values[0] * len(itemset)
    #print(support_ab)

    return abs(support_a-support_b) / (support_a+support_b-support_ab)
```

Figure 5.Imbalance Ratio Implementation

This method; takes the two frozensets to constructs a union of the sets. Using the frequent itemsets; it gets the support value for each of the itemsets. Then it calculates the support counts and use the equation in Week 7 slide 38 to calculate the imbalance ratio. The formula is given below:

$$I_{\mathcal{A},\mathcal{B}} = \frac{|N_{\mathcal{A}} - N_{\mathcal{B}}|}{N_{\mathcal{A}} + N_{\mathcal{B}} - N_{\mathcal{A}\cup\mathcal{B}}}$$

```
# Do two tests: - Same as previous section:

# Test 1: (eggs -> onion) - Both ways are strong! 0.7
ants = frozenset(['Eggs'])
cons = frozenset(['Onion'])
print('Test 1: ' + str(calculate_imbalance_ratio(frequent_itemsets, ants, cons)))

# Test 2: (eggs, corn -> ice cream) - Both ways are strong! 0.7
ants = frozenset(['Eggs', 'Corn'])
cons = frozenset(['Ice cream'])
print('Test 2: ' + str(calculate_imbalance_ratio(frequent_itemsets, ants, cons)))
```

```
Test 1: 0.2500000000000001
Test 2: 0.0
```

Figure 6. Example Outputs on Imbalance Ratio

The outputs for the imbalance ratio are given above. There are 149 frequent itemsets in total. Eggs have a support of 0.8; Onions have a support of 0.6 and {Eggs, Onions} have a support of 0.6. When put on the equation the imbalance ratio is indeed 0.25. Support count can be calculated by support times the length of the frequent itemset.

For test 2; antecedent, consequent, and union classes have a support of 0.2. Therefore, imbalance ratio becomes 0.

Part 2: Nov 10, 2020

Question 2.1:

Contextual attributes explain the datum's (object's) context. Behavioral attributes explain the characteristics of the object within the context as described in Week 8, Slide 7. Here are two examples:

Example 1:

- **Contextual Attribute**: Time of Transaction
- **Behavioral Attribute**: Amount of Transaction

Explanation: Customers have a daily, weekly, and monthly trends. Given this context; the amount of transaction made at a time can be considered as a behavioral measure to identify whether a certain transaction is an outlier or not. For example, it is less likely to do a large transaction in the middle of the night.

Example 2:

- **Contextual Attribute**: Type of Transaction
- **Behavioral Attribute**: Date of the Transaction

Explanation: The customers use their cards for a variety of needs. These needs may follow seasonality. Such examples include school costs on September, tourism costs in holiday season or heating costs in winter period. If a certain type of transaction occurs when it is not expected; this could indicate an outlier.

Note that all these attributes can be worked together as well; they are not an exhaustive list.

Question 2.2:

As a brief answer, statistical methods can be used for the dataset. The exact application of the statistical methods is dependent on how the data is going to be used.

If the outliers are going to be detected within a context of an attribute; univariate approach can be used; which was already used in Lab Assignment #2. It detected attribute's outliers by fitting samples of that attribute to normal distribution and assigning samples that are 3 or more standard deviations away as outliers. Later it showed these plots in boxplot, as seen below. The rows with outliers are dropped later, as pre-processing step.

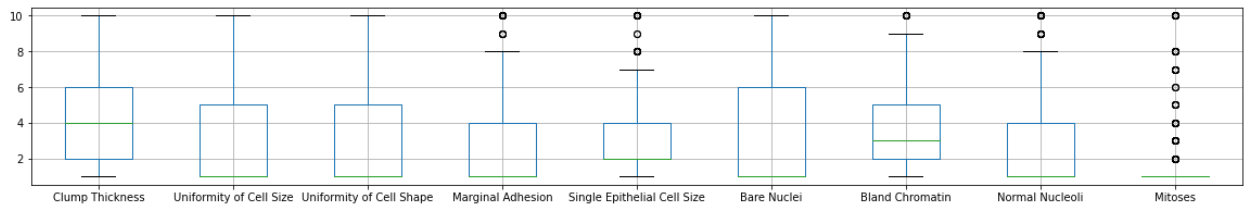

Figure 7. Box Plots of Outliers

If all attributes are going to be used to determine the outliers, this multivariate outlier's problem should be turned into a univariate problem by using either Mahalanobis distance or Chi-Squared statistic (under assumption of normal distribution). The rest follows from univariate outlier analysis.

Question 2.3:

For this question univariate statistical methods are used to extract the outlier. The outlier is visible by inspecting the data as well, it has the value "4.55", it is the fifth (May) entry of the dataset. After normalizing data; a range of 3 standard deviations are assumed normal; the values above or below three standard deviations are considered as outlier. The following code was written and the results are obtained:

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

# Construct the dataset:
q3data = np.array([22.93, 20.59, 25.65, 23.74, 25.24, 4.55, 23.45, 28.18, 23.52, 22.32, 26.73, 23.42])

# Data assumed to fit normal distribution - Use MLE to identify outlier values:
mu = np.mean(q3data)
sigma = np.std(q3data)

Z_q3 = (q3data - mu)/sigma
print('All values: ')
print(Z_q3)# 5th index is an outlier.
```

```
All values:
[ 0.07003718 -0.33629424  0.54235404  0.21069036  0.47115922 -3.12157455
  0.16033305  0.98167818  0.17248827 -0.03588682  0.72989162  0.15512368]
```

Figure 8. The code segment and answers to Question 2.3

With the value -3.12; fifth entry; corresponds to May; is more than 3 standard deviations away. Therefore, this month can be identified as an outlier.

Question 2.4:

The following code was written, relative frequencies are also given below:

```
import pandas as pd
import matplotlib.pyplot as plt

# Load CSV file, set the 'Date' values as the index of each row, and display the first rows of the dataframe
q4data = pd.read_csv('graduation_rate.csv', header='infer')
freq_hs_gpa = q4data['high school gpa'].value_counts()/len(q4data)
print('High School GPA Relative Frequencies:')
display(freq_hs_gpa)

# Show relative frequencies in bar plot
plt.bar(freq_hs_gpa.index, freq_hs_gpa, width = 0.05 * (4-2.8))
plt.xlabel('Grades')
plt.ylabel('Relative Frequencies')
```

```
High School GPA Relative Frequencies:
4.0    0.294
3.8    0.132
3.9    0.108
3.7    0.106
3.6    0.101
3.5    0.071
3.4    0.064
3.3    0.052
3.2    0.025
3.0    0.020
3.1    0.015
2.9    0.008
2.7    0.003
2.8    0.001
Name: high school gpa, dtype: float64
```

Figure 9. The code for calculating relative frequencies

For better visualization, the bar-chart is also given:

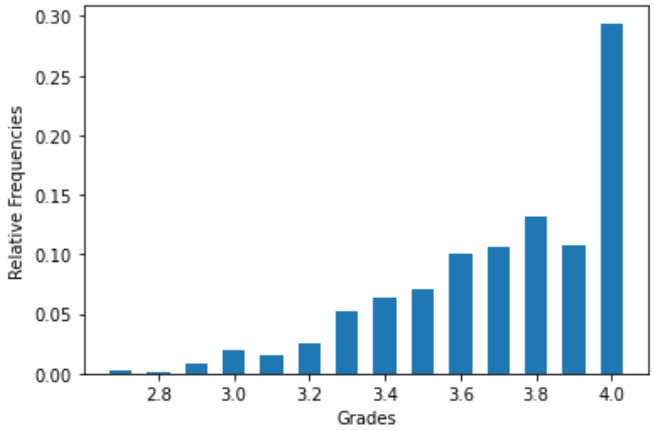

Figure 10. Bar Chart of Relative Frequencies of GPA

If two more entries are added with GPA's 2.8 and 3.6; the former (2.8) will be considered as an outlier as it is less frequently observed within the context of high school GPA population. 2.8 has a relative frequency of 0.001 whereas 3.6 has 0.101. A GPA of 2.8 deviates too much from the overall trend.

Question 2.5:

As input to the classifier, the percentage of changes in daily closing price of each stock was implemented in Section 1 of the notebook. This section follows from this part:

```
# These are code migrated from the assignment: - I want each question to work in clean workspace:
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

# Works for 5.1:
# 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()
```

Date	MSFT	F	BAC
1/4/2007	-0.167455	2.529960	0.637532
1/5/2007	-0.570278	-1.038961	-0.801185
1/8/2007	0.978411	1.443570	0.394438
1/9/2007	0.100231	0.776197	0.093543
1/10/2007	-1.001332	-0.770218	0.149536

Figure 11. Loading up the dataset; calculating percentage change in daily closing price

Then using the percentage changes in daily closing prices One-class SVM can be implemented; the implementation follows from Section 3 of the notebook tailored for the given dataset.

The code and the output are as follows:

```
# 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=(16,10)).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()
```

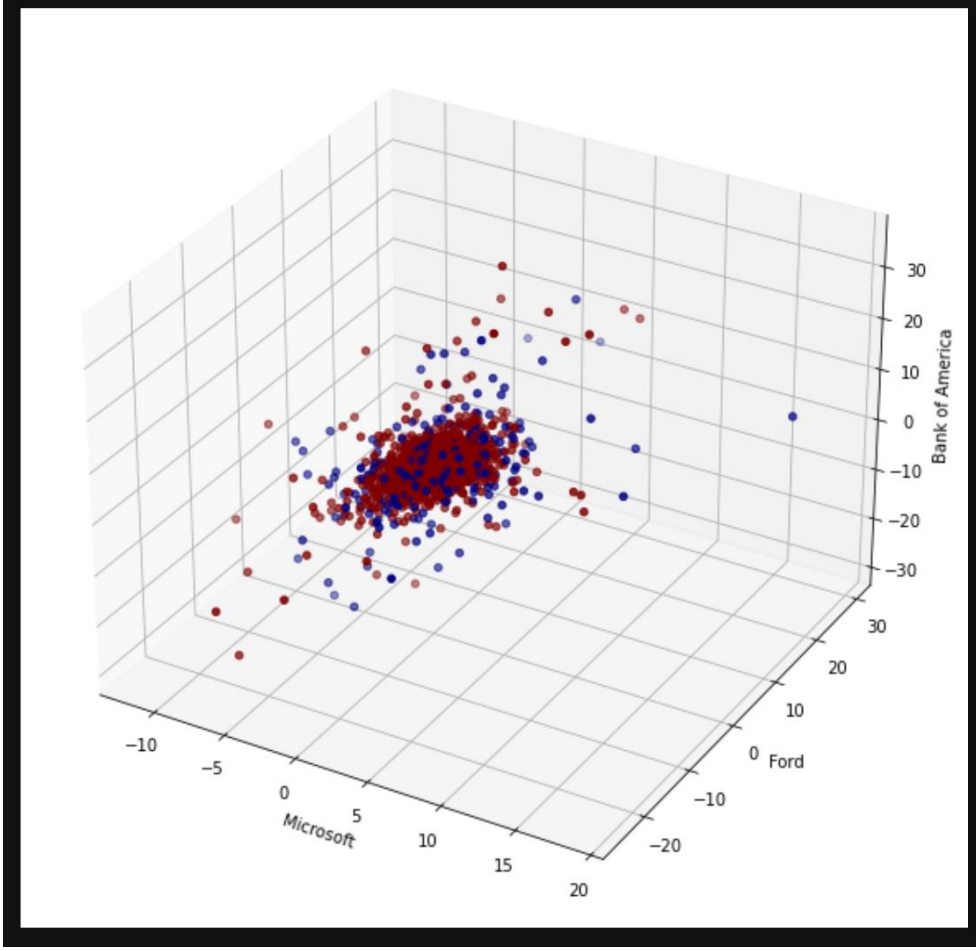

Figure 12. Scatter Plot of Stocks with Inlier and Outlier Labels shown with red and blue

The question also asks for the histogram and frequencies of estimated outlier and inlier labels. The following code segment and the results describe the results:

```
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=16)
plt.ylabel('Count', fontsize=16)

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)))
```

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

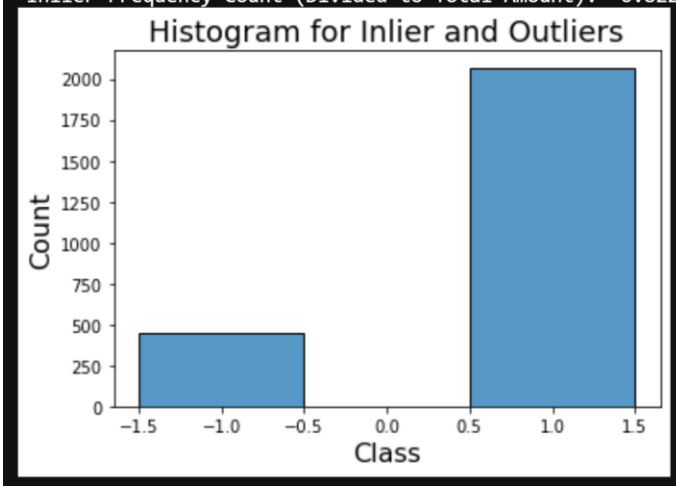

Figure 13. Histogram and Frequency of Inliers vs Outliers

For parametric methods multivariate Gaussian distribution is used. For proximity-based methods nearest neighbor is used in Sections 1 and 2. Both methods rely on distance metrics (Mahalanobis and Euclidean respectively). The results are slightly differing from each other due to different metrics used however the overall trend and the top outliers that are found are the same. These outlier labels/scores are continuous and based on the distances.

For one-class SVM approach; the classes are binary, i.e., outlier and inlier. The labeled data-points do not have an associated outlier score. In this dataset, 82% of the datapoints are labeled as inliers, whereas 18 % of the datapoints are considered as outliers. Figure 13 displays the counts and frequency of the outliers and the inliers with respect to the dataset.

For visualization see the figure below:

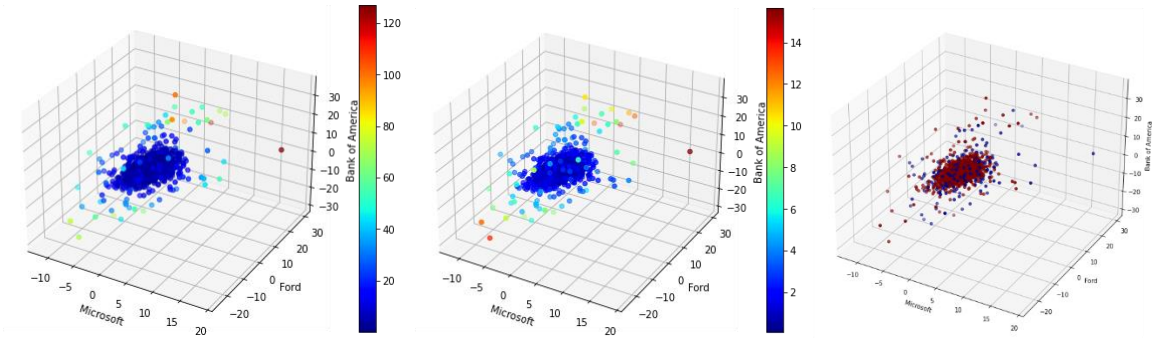

Figure 14. 3D Scatterplots: MV-Gaussian (Left), Nearest Neightbor (Middle), One-class SVM (Right)

Since MV-Gaussian and NN methods use distances to decide on the metrics they show very similar results, which deviate slightly due to the distance metric used. In both methods, inliers are clustered in the center; outliers are generally distributed in the peripheries.

SVM uses discrete labels inlier and outlier, it has a similar trend for the inliers as well. Red datapoints are inliers and blue data points outliers. It is seen that larger distances (green to red) in the first two methods translate to outlier labels in SVM method, whereas smaller distances (blue to green) translate to inliers.

As a difference Gaussian and NN methods give metrics about what can be considered as an outlier or not. SVM directly separates out inliers and outliers.

Question 2.6:

The data is prepared and preprocessed using the code below. Data and the label were split prior to normalization.

```
import pandas as pd
from pandas import read_csv
import numpy as np

# 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)
#print(y)
#print(X)
```

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

```
# 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
```

Figure 15. Data preparation

For normalization z-score normalization is used; by using the second cell given in the figure above. Then PCA is applied on z-normalized data. The code and the output can be found below:

```
# Apply PCA:
from sklearn.decomposition import PCA

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

projected = pca.transform(Z)
projected = pd.DataFrame(projected,columns=['pc1','pc2'],index=range(1,len(Z)+1))

display(projected)
```

	pc1	pc2
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

Figure 16. Code and Output for PCA Implementation

Finally, outlier detection is applied on pre-processed dataset using 2-nearest neighbors' approach. The results are also displayed on 2D scatter plot, where dimensions are the first two principal components respectively. The code for this section is below:

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

# Implement a 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)

# The 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=(10,6))
p = plt.scatter(projected['pc1'], projected['pc2'], c=outlier_score, cmap='jet')
plt.xlabel('PC1')
plt.ylabel('PC2')
fig.colorbar(p)
plt.show()
```

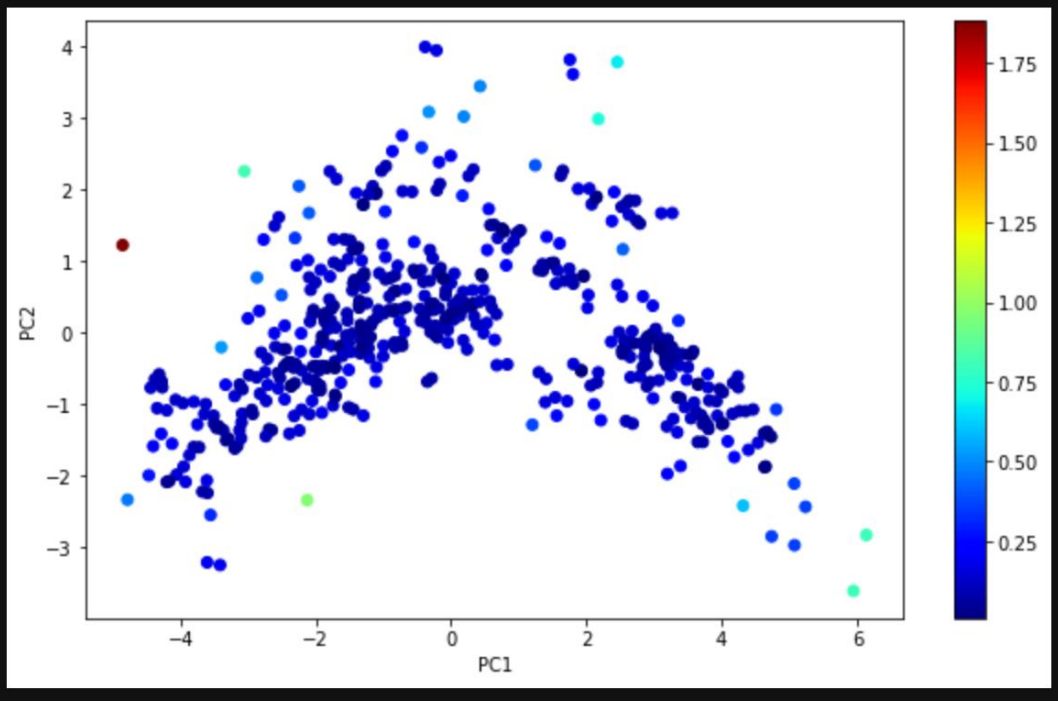

Figure 17. Implementation of 2-nearest neighbors & scatterplot visualization