Subarna Chowdhury Soma
SJSU ID: 014549587

1. **[a] [10 points]**

The wine data set at https://archive.ics.uci.edu/ml/datasets/wine

has 13 features. Develop in Python and apply your own version of the PCA algorithm to this data set, to visualize how PCA helps with dimensionality reduction.

Explain how many Principal Components you will choose and why.

What percent of the variance in the data do the selected Principal Components cover?

For the implementation, you may use any objects, modules, and functions in NumPy, SciPy and other python libraries to do various operations such as to compute the eigen values, vectors or perform any other math / linear algebra operation, but not use the PCA function available in SciKit-Learn directly.

Answer:

File: hw4_PCA_wine_data_fromScratch_vs_Library

Principal Components Selection:

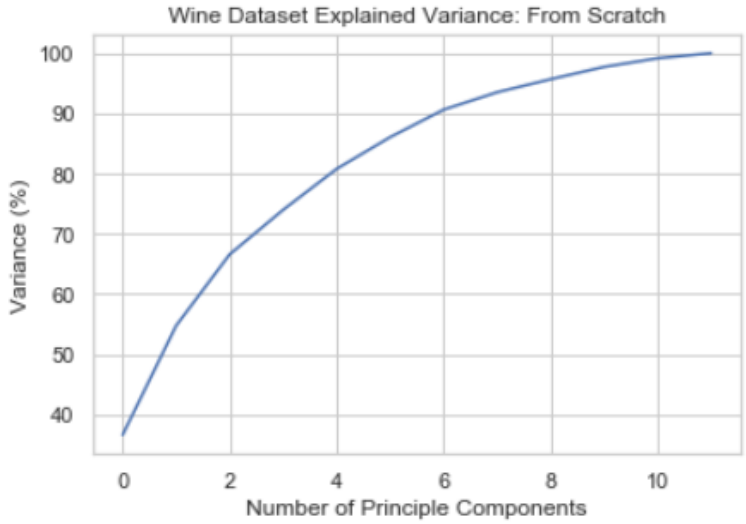
Number of Principle component is selected based on the percentage of variance, means information the PCs carry in the new transformed dataset. After Eigenvalue and Eigenvector calculation, I have calculated explained variance ratio and plotted against the number of principal components to decide the number of principal component number to choose. Sum of all individual principle components is called total variance. So the fraction of explained variance by a principal component is the ratio between the variance of that principal component and the total variance and can be presented as below for each eigenvector:

$|\lambda j| \ / \ \Sigma|\lambda j|$

Following are my plots from code:

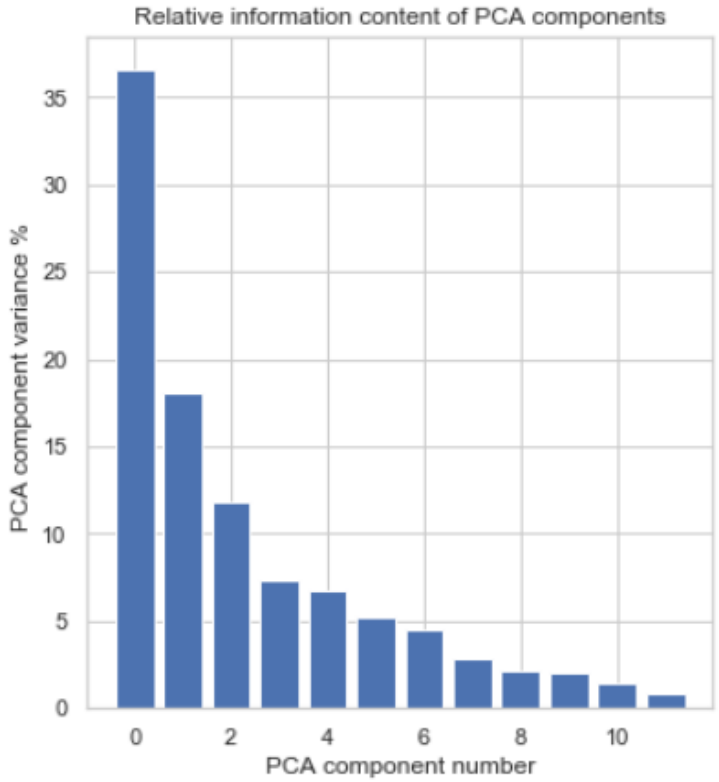
In [28]:

```
#Plotting the Cumulative Summation of the Explained Variance
plt.figure()
plt.plot(np.cumsum(explained_variance))
plt.xlabel('Number of Principle Components')
plt.ylabel('Variance (%)') #for each component
plt.title('Wine Dataset Explained Variance: From Scratch')
plt.show()
```



In [31]:

```
#pca vs variance
fig = plt.figure(figsize=(12,6))
fig.add_subplot(1,2,1)
plt.bar(np.arange(12), explained_variance)
plt.title('Relative information content of PCA components')
plt.xlabel("PCA component number")
plt.ylabel("PCA component variance % ")
plt.show()
```

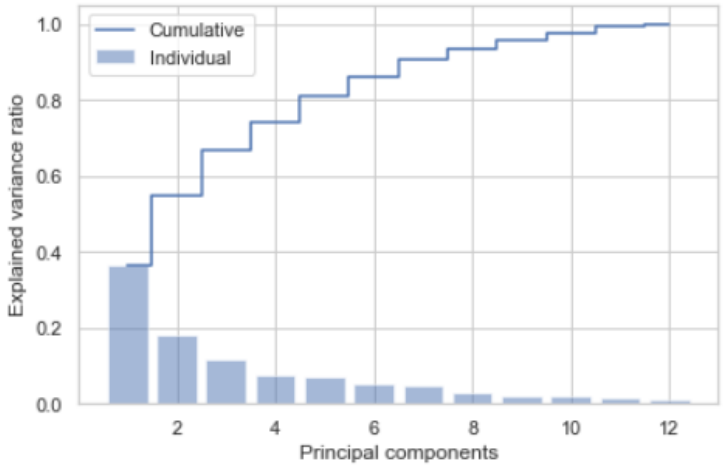


In [32]:

```
tot = sum(np.abs(eigen_vals))
var_exp = [(i / tot) for i in sorted(np.abs(eigen_vals), reverse=True)]
cum_var_exp = np.cumsum(var_exp)

plt.bar(range(1, eigen_vals.size + 1), var_exp, alpha=0.5, align='center',
        label='Individual')
plt.step(range(1, eigen_vals.size + 1), cum_var_exp, where='mid',
         label='Cumulative')
plt.ylabel('Explained variance ratio')
plt.xlabel('Principal components')
plt.legend(loc='best')
plt.tight_layout()

plt.show()
```



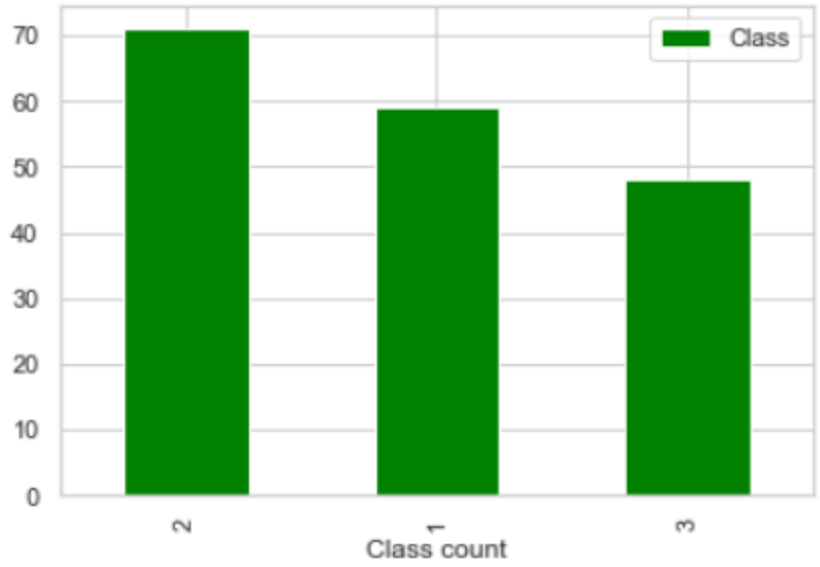
Here plots are presented in descending order to decide the number of eigenvectors to keep. The resulting plots indicate that first principle component alone can carries 40% of the variance. From the cumulative sum calculation and plot it is clear that almost 60% of the variance of data is combindly represented by the first two principal components. So I can put a threshold here and choose *two principal components*. So my selected *principal components will cover 60% variance of data*. It will be also easier to plot the data in two dimensional scatter plot. In reality, the number of principal components can be selected deciding upon the trade-off between computational efficiency and performance.

Output and Observation:

The main wine dataset has 13 features with three different classes:

In [8]:

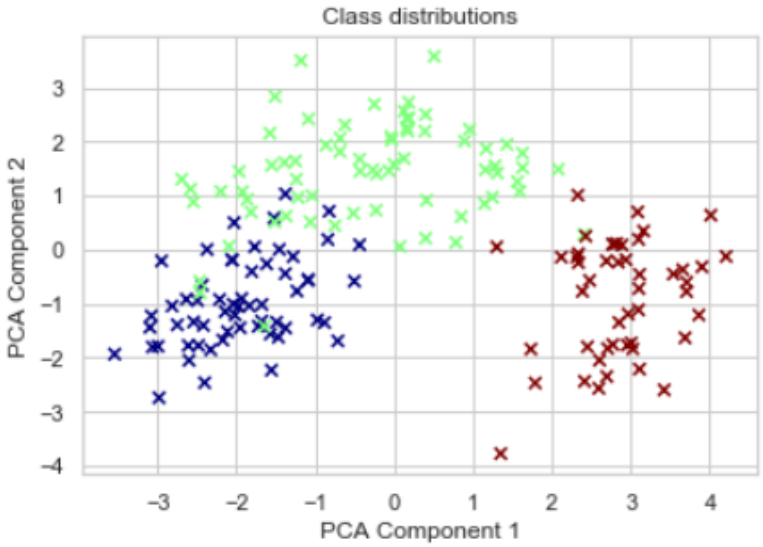
```
#Plot for class
data["Class"].value_counts().plot.bar(color='Green')
plt.xlabel("Class count")
plt.legend()
plt.show()
```



After dimensionality reduction using PCA, new transformed data can be represented as two dimensions. Here is transformed class distribution of data:

In [33]:

```
#Class distributions
fig.add_subplot(1,2,2)
plt.scatter(pca1, pca2, c=y_train, marker='x', cmap='jet')
plt.title('Class distributions')
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.show()
```



At the bottom of my notebook, I have verified my PCA implementation from scratch by comparing the result with sklearn-PCA implementation and found that results are matching exactly.

1. Explained variance from my implementation:

In [24]:

```
#explained_variance calculation
total = sum(eigen_vals)
print('Sum of eigen values:', total)
explained_variance = [(i / total)*100 for i in sorted(eigen_vals, reverse=True)]
print('Explained variance:', explained_variance)
```

```
Sum of eigen values: 12.067796610169497
Explained variance: [36.632969340019486, 18.12747309931419, 11.829746542581228, 7.398949352643272, 6.7984467263487405, 5.266
291586598307, 4.565687389134465, 2.889379524708695, 2.1627943406002745, 2.0538734701139814, 1.4125374898181164, 0.8618511381
192573]
```

Explained variance calculation using sklearn-PCA:

```
In [43]:
# Preprocessing: Feature Scaling
sc = StandardScaler()
X_train_PCA = sc.fit_transform(X_train)
#X_test = sc.transform(X_test)

# Applying PCA
from sklearn.decomposition import PCA
pca = PCA() # n_components = 2
X_train_PCA = pca.fit_transform(X_train_PCA)

explained_variance = pca.explained_variance_ratio_
print('Explained variance:',explained_variance * 100)
#print('pca.n_components', pca.n_components_)
```

```
Explained variance: [36.63296934 18.1274731  11.82974654  7.39894935  6.79844673  5.26629159
  4.56568739  2.88937952  2.16279434  2.05387347  1.41253749  0.86185114]
```

2. Cumulative sum from my implementation:

```
In [25]:
#Cumulative sum
print('Cumulative sum',np.cumsum(explained_variance))
```

```
Cumulative sum [ 36.63296934  54.76044244  66.59018898  73.98913833  80.78758506
  86.05387665  90.61956404  93.50894356  95.6717379   97.72561137
  99.13814886 100.        ]
```

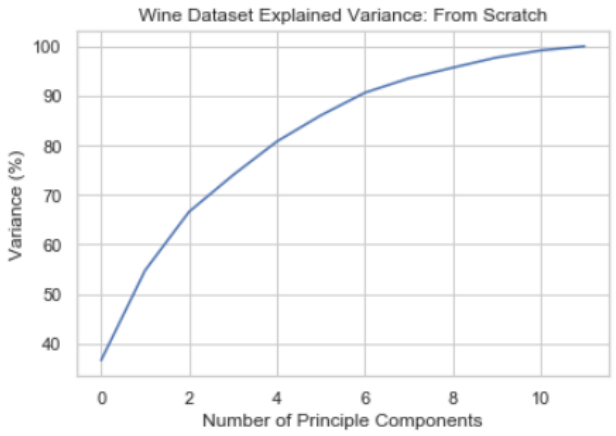
Cumulative sum using sklearn-PCA:

```
In [42]:
#Cumulative sum
print('Cumulative sum',np.cumsum(pca.explained_variance_ratio_ *100))
```

```
Cumulative sum [ 36.63296934  54.76044244  66.59018898  73.98913833  80.78758506
  86.05387665  90.61956404  93.50894356  95.6717379   97.72561137
  99.13814886 100.        ]
```

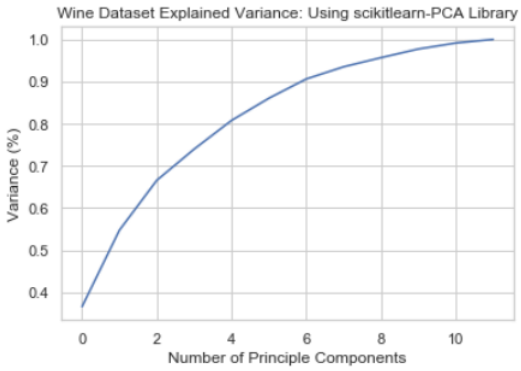
3. Plotting Explained Variance vs. no of principal components from my implementation:

```
In [28]:
#Plotting Explained Variance vs. no of principal components
plt.figure()
plt.plot(np.cumsum(explained_variance))
plt.xlabel('Number of Principle Components')
plt.ylabel('Variance (%)') #for each component
plt.title('Wine Dataset Explained Variance: From Scratch')
plt.show()
```



Plotting Explained Variance vs. no of principal components using sklearn-PCA:

```
In [41]:
#Plotting Explained Variance vs. no of principal components
plt.figure()
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of Principle Components')
plt.ylabel('Variance (%)') #for each component
plt.title('Wine Dataset Explained Variance: Using scikitlearn-PCA Library')
plt.show()
```



[b] [10 points]

For each of the numerical features of the above dataset, apply one of the techniques you learnt in the class for data reduction. Use as many applicable techniques as possible, such as saving only the parameters of their linear or log

linear models (if applicable), partitioning into equal width or equal depth bins, applicable techniques of sampling, etc.

Answer:

a. **Partitioning data into equal frequency and equal width:**

File:

hw4_WineDataReduction_EqualWidth_EqualFreq_Binning_Histogram;

hw4_WineDataReduction_Histogram_byR.R [for combined histogram visualization after partition]

I have implemented both equal frequency and equal width binning methods for partitioning data using each column of wine dataset for data reduction. Now for each bin, we can store the average value instead of storing the entire data and therefore data will be reduced.

b. **Clustering:** Partition data based on similarity clusters the data and hence reduces dimension.

File: hw4_WineDataReduction_Clustering

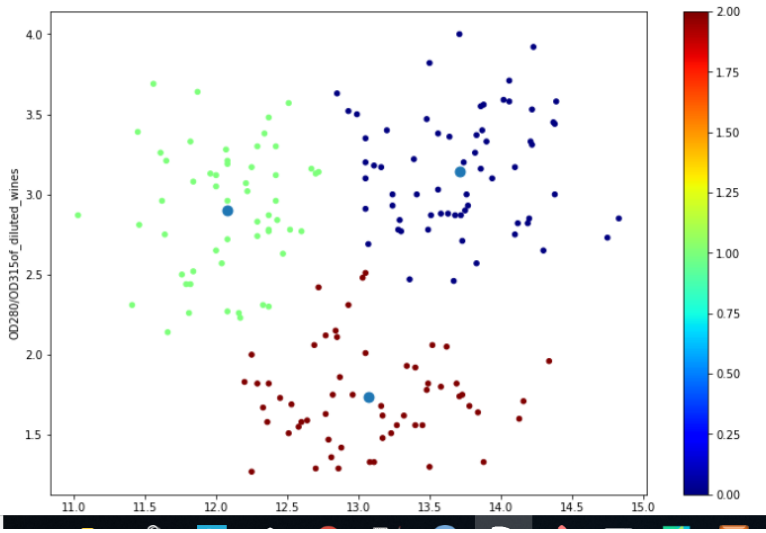
Output and Observation: Since we know the true Class, this K means clustering algorithm will be supervised learning algorithm. Here I will calculate the centroid of each cluster, say considering two columns can be done on all the columns. Later, instead of storing the whole data, we can only store the centroid of each cluster to reduce the dimension of numerical columns.

```
In [11]: #recalculating centroid by taking mean of the clusters for better result

kmeans = KMeans(n_clusters=3, init = 'random', max_iter = 100, random_state = 5).fit(data.iloc[:,[12,1]])
centroids = pd.DataFrame(kmeans.cluster_centers_, columns = list(data.iloc[:,[12,1]].columns.values))

fig, ax = plt.subplots(1, 1)
data.plot.scatter(x = 'Alcohol', y = 'OD280/OD315of_diluted_wines', c= kmeans.labels_, figsize=(12,8), colormap='jet', ax=ax,
centroids.plot.scatter(x = 'Alcohol', y = 'OD280/OD315of_diluted_wines', ax = ax,  s = 80, mark_right=False)

Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x11c16830>
```



c. Sampling:

File: hw4_WineDataReduction_StrataSampling

Here I have used sampling technique to reduce data. In sampling method we choose a representative of data to represent the whole data set by randomly selecting smaller set. In this way data reduction is also possible by storing/choosing that representative instead large dataset. I have used stratified method. First I have identified the most correlated attribute with ‘target variable ‘class’. Then divided that attribute into strata to represent significant classes of that column. Same techniques can be performed on all the columns.

2. **[a] 10 points**

Assume we have only two features in our dataset. The transposes of the feature vectors comprise of the first 10 consecutive pairs of primes: [2 3], [5 7], …, [67 71]. For k=2, show step-by-step (either manually or programmatically) iterations of k-means clustering when the centroids are initialized to (i) [2 3] and [5 7] and to (ii) [2 3] and [67 71]. Compare and comment on the results in both cases.

Answer:

File: hw4_Kmean_step_by_step.py

I have written a code in python to implement k-means clustering. The algorithm terminates when there are no changes in the clusters in successive iterations. Euclidean distance is used as the distance between two points. The step by step assignment of points into clusters and computation of centroids are shown below when the centroids are initialized to [2 3] and [5 7]:

```
***Iteration 1 ***
('cluster %s', 0)
Point: x = 2, y = 3

('cluster %s', 1)
Point: x = 5, y = 7

Point: x = 11, y = 13

Point: x = 17, y = 19

Point: x = 23, y = 29

Point: x = 31, y = 37

Point: x = 41, y = 43

Point: x = 47, y = 53

Point: x = 59, y = 61

Point: x = 67, y = 71

Centroids:
Point: x = 2, y = 3

Point: x = 33, y = 37

***Iteration 2 ***
('cluster %s', 0)
Point: x = 2, y = 3

Point: x = 5, y = 7

Point: x = 11, y = 13

Point: x = 17, y = 19

('cluster %s', 1)
Point: x = 23, y = 29

Point: x = 31, y = 37

Point: x = 41, y = 43

Point: x = 47, y = 53

Point: x = 59, y = 61

Point: x = 67, y = 71

Centroids:
Point: x = 8, y = 10

Point: x = 44, y = 49
```

```
***Iteration 3 ***
('cluster %s', 0)
Point: x = 2, y = 3

Point: x = 5, y = 7

Point: x = 11, y = 13

Point: x = 17, y = 19

Point: x = 23, y = 29

('cluster %s', 1)
Point: x = 31, y = 37

Point: x = 41, y = 43

Point: x = 47, y = 53

Point: x = 59, y = 61

Point: x = 67, y = 71

Centroids:
Point: x = 11, y = 14

Point: x = 49, y = 53

***Iteration 4 ***
('cluster %s', 0)
Point: x = 2, y = 3

Point: x = 5, y = 7

Point: x = 11, y = 13

Point: x = 17, y = 19

Point: x = 23, y = 29

('cluster %s', 1)
Point: x = 31, y = 37

Point: x = 41, y = 43

Point: x = 47, y = 53

Point: x = 59, y = 61

Point: x = 67, y = 71

Centroids:
Point: x = 11, y = 14

Point: x = 49, y = 53
```

The step by step assignment of points into clusters and computation of centroids are shown below when the centroids are initialized to [2 3] and [67 71]:

```
***Iteration 1 ***
('cluster %s', 0)
Point: x = 2, y = 3

Point: x = 5, y = 7

Point: x = 11, y = 13

Point: x = 17, y = 19

Point: x = 23, y = 29

Point: x = 31, y = 37

('cluster %s', 1)
Point: x = 41, y = 43

Point: x = 47, y = 53

Point: x = 59, y = 61

Point: x = 67, y = 71

Centroids:
Point: x = 14, y = 18

Point: x = 53, y = 57

***Iteration 2 ***
('cluster %s', 0)
Point: x = 2, y = 3

Point: x = 5, y = 7

Point: x = 11, y = 13

Point: x = 17, y = 19

Point: x = 23, y = 29

Point: x = 31, y = 37

('cluster %s', 1)
Point: x = 41, y = 43

Point: x = 47, y = 53

Point: x = 59, y = 61

Point: x = 67, y = 71

Centroids:
Point: x = 14, y = 18

Point: x = 53, y = 57
```

Here we can observe that when we assign two close points as centroids ((2, 3) and (5, 7)), it takes more iterations for termination of the algorithm, but when we assign two distant points ((2, 3) and (67, 71)), it takes very few iterations. This is expected, because when we assign two distant points, the initial assignment of the points are not changed much in successive iterations. On the other case, both initial clusters are very much different from the final clusters, therefore they need more iterations to reach termination.

I have also written two functions to compute the inter-cluster distance (here I have computed the average linkage distance between two clusters) and the intra-cluster distances (here I have implemented the average diameter distance which is the average of the distances between all pairs of points) of the clusters. The inter-cluster distances for two different choices of centroids is very close: 53.9 for (2,3) and (5,7) and 54.9 for (2, 3) and (67, 71) as the initial centroids. So, in terms of inter-cluster distance, the 2nd choice is marginally better.

For the first choice of centroids, the intra-cluster distances of the two clusters are 16.78 and 25.01, where the intra-cluster distances of the two clusters are 21.5 and 21.1 for the 2nd choice. So, we can observe that the first cluster is formed better for the first choice, but the 2nd cluster is worse than the 2nd cluster for the 2nd choice. Therefore, the 2nd choice of centroids forms both clusters with similar intra-cluster distances for both clusters with marginally better inter-cluster distances.

[b] 10 points

K-means algorithm is applied to the above wines dataset in the tutorial available at https://www.kaggle.com/xvivancos/clustering-wines-with-k-means

However, K-means algorithm assumes that the mean is representative of the cluster. In real-world though, most often, the "most" vociferous, "most" influential, "most" wealthy, or some other "most" ____ person gets elected to represent the people of a constituency. Explain using visualization tools (like in the tutorial) and words, the difference the clustering algorithm makes, if based on the "mode" instead of the "mean" on the above wines dataset. Based on the cluster analysis, can the most _____ wine represent a cluster of wines? Explain.

Answer:

File: hw4_WineData_Kmean_Kmode_byR.R

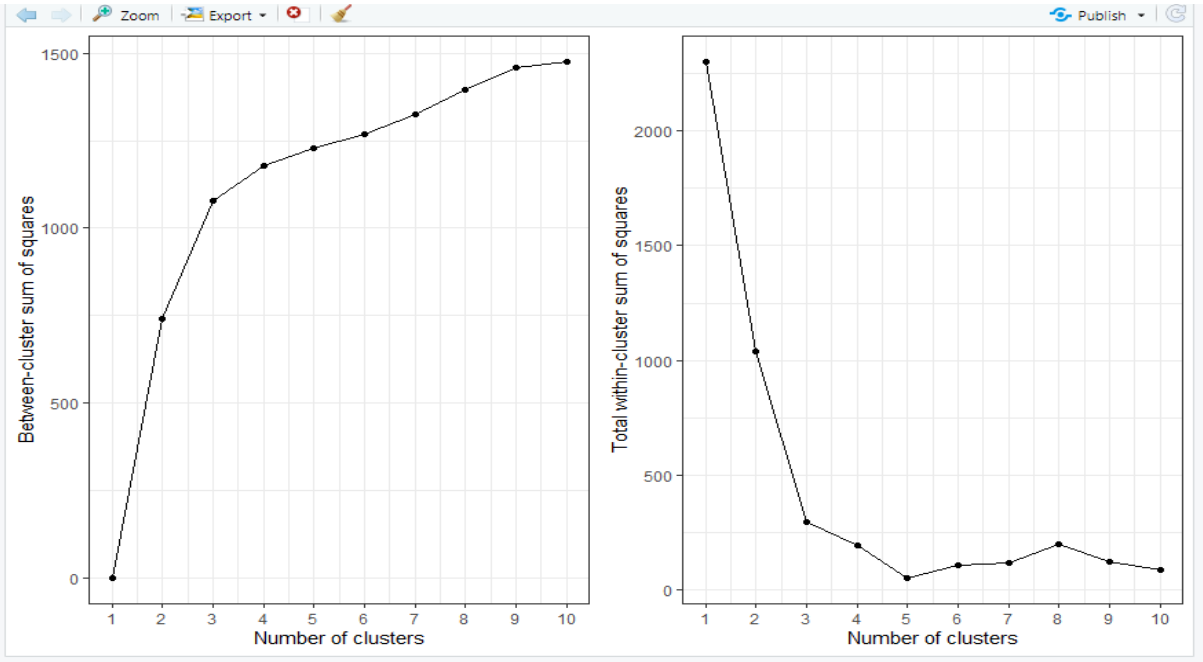
Observation: In the code I have first implemented K-mean algorithm on wine data set following the article. Then I have implemented K-mode algorithm which considers

'mode/most' instead of 'mean'. Section wise I have compared the output and observations and at the bottom I have done a purity check on the algorithms to compare their performance on wine data set.

1. K value selection:

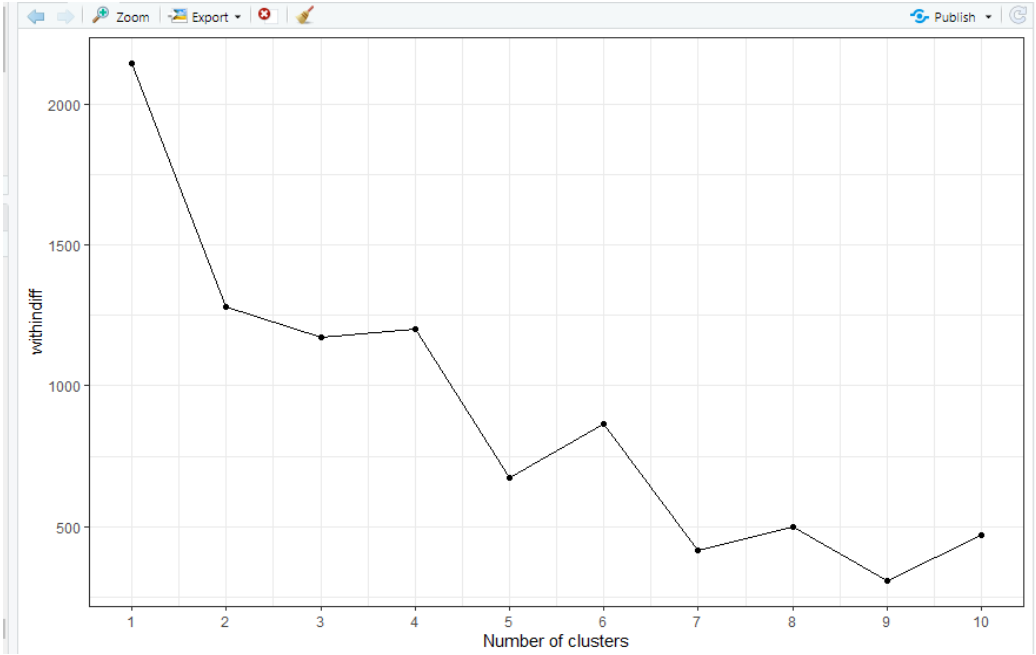
I have plotted numbers of K's against appropriate metrics for K-mean and K-mode to decide the optimal number of K. K is chosen in such a way that adding another cluster doesn't give better partition of data considering heterogeneous cluster and homogeneity inside the clusters. Analyzing my case, I have chosen that 3 will be appropriate value for k.

K-mean: For K-mean to decide upon the number of clusters, I have plotted betweenss and total withinss against number of clusters. betweenss is the between-cluster sum of squares, which is expected to be as high as possible for heterogeneous clusters and tot.withinss is defined as total within-cluster sum of squares, which is expected to be as lower as possible for homogeneity within clusters.



K-mode: For K-mode the library provides a measure called 'withindiff' which is the within-cluster simple-matching distance for each cluster. I have plotted it against number

of cluster and it is desired to be as lower as possible homogeneity within clusters. Here the graph is not very smooth as it for K-mean. But after K=3 there is a very small change in value so I chose K=3 for K-mode also.



2.Output: Output from both the algorithms is given below along with the plots showing the clustering result.

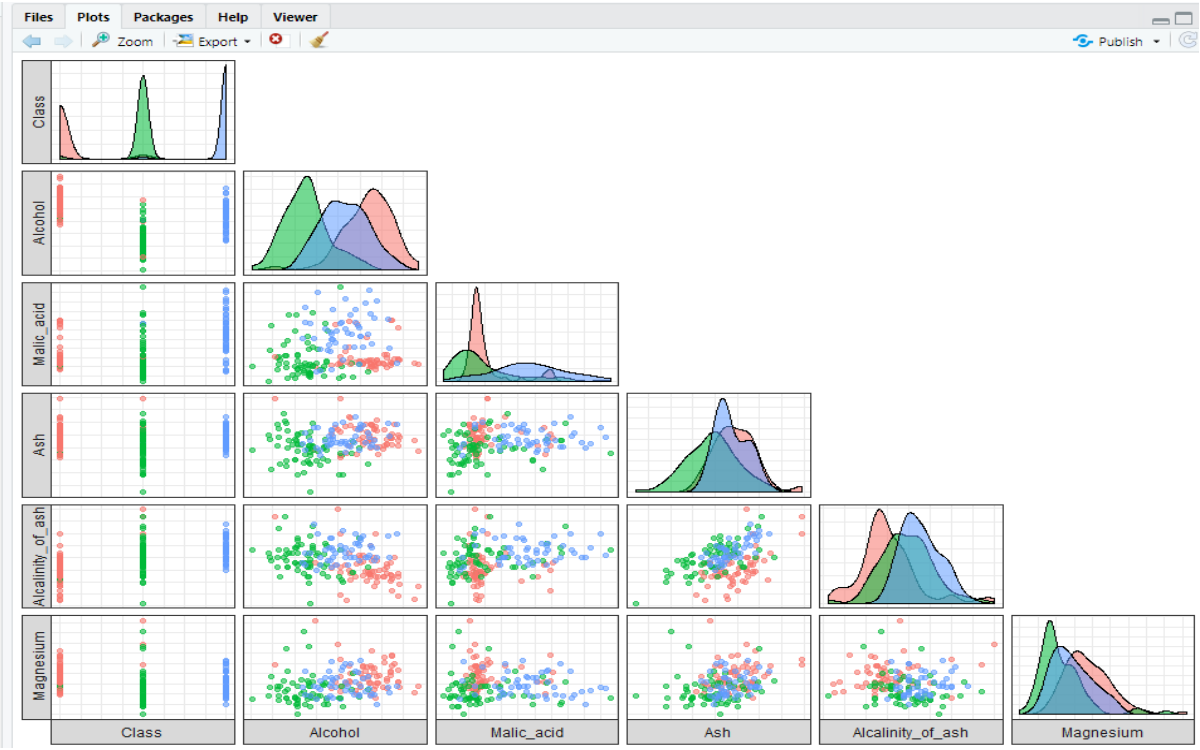
K-mean:

```
> # Cluster to which each point is allocated
> wines_kmean3 $cluster
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [59] 1 2 2 2 2 2 2 2 2 2 2 2 2 1 2 1 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[117] 2 2 2 2 2 1 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
[175] 3 3 3 3
>
> # Cluster centers
> wines_kmean3$centers
      Class    Alcohol Malic_acid        Ash Alcalinity_of_ash   Magnesium Total_phenols   Flavanoids
1 -1.12592139  0.8564436 -0.2987303  0.4172380       -0.55956839  0.57822065     0.9088777  0.992026486
2  0.04178649 -0.9023230 -0.3822988 -0.5533599        0.08253905 -0.48231331    -0.1049780 -0.001624458
3  1.34366784  0.1860184  0.9024258  0.2485092        0.58206156 -0.05049296    -0.9857762 -1.232717398
  Nonflavanoid_phenols Proanthocyanins Color_intensity        Hue OD280.OD315of_diluted_wines
1          -0.56681851     0.609361150       0.1789516  0.4888223                   0.7969904
2          -0.00662514    -0.007993843      -0.8708267  0.4175188                   0.2202887
3           0.71482528    -0.747498955       0.9857177 -1.1879477                  -1.2978785
>
> # Cluster size
> wines_kmean3 $size
[1] 61 68 49
>
> # Between-cluster sum of squares
> wines_kmean3 $betweenss
[1] 1079.191
>
> # Within-cluster sum of squares
> wines_kmean3 $withinss
[1] 357.4263 566.1694 298.2133
>
> # Total within-cluster sum of squares
> wines_kmean3 $tot.withinss
[1] 1221.809
>
> # Total sum of squares
> wines_kmean3 $totss
[1] 2301
```

```
>
> # Mean values of each cluster
> aggregate(wines, by=list(wines_kmean3$cluster), mean)
  Group.1    Class  Alcohol Malic_acid      Ash Alcalinity_of_ash Magnesium Total_phenols Flavanoids
1       1 1.065574 13.69590   2.002623 2.480984          17.62623 108.00000      2.863934  3.0201639
2       2 1.970588 12.26809   1.909265 2.214706          19.77059  92.85294      2.229412  2.0276471
3       3 2.979592 13.15163   3.344490 2.434694          21.43878  99.02041      1.678163  0.7979592
  Nonflavanoid_phenols Proanthocyanins Color_intensity       Hue OD280.OD315of_diluted_wines
1            0.2913115        1.939672        5.472951 1.0691803                    3.177541
2            0.3610294        1.586324        3.039265 1.0528824                    2.768088
3            0.4508163        1.163061        7.343265 0.6859184                    1.690204
>
```

```
> #checking
> table(data_$Class,wines_kmean3$cluster)

    1  2  3
  1 57  2  0
  2  4 66  1
  3  0  0 48
> # Execution of k-mode with k=3
```



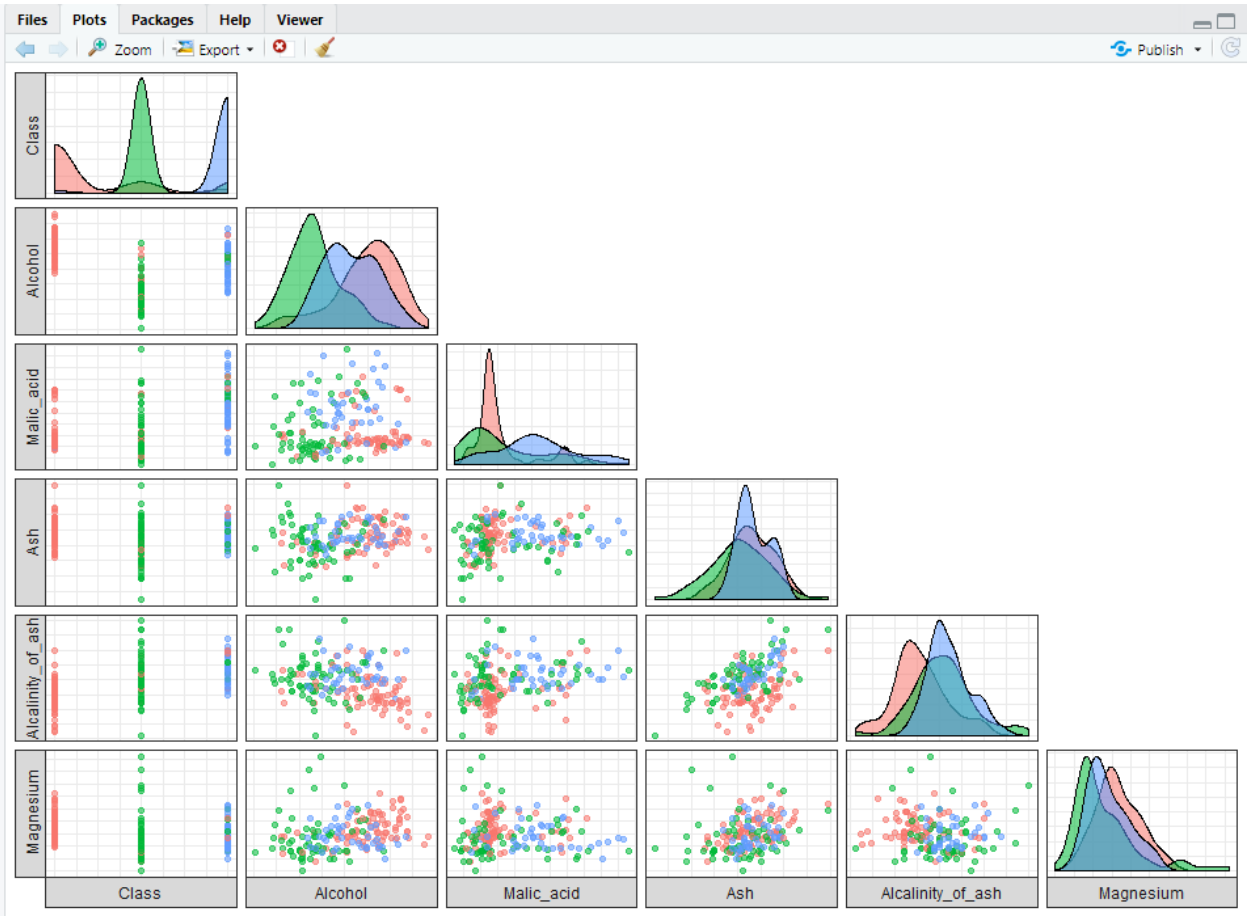
K-mode:

```
>
> # Cluster to which each point is allocated
> wines_kmode3 $cluster
  [1] 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [54] 1 1 1 1 1 1 2 1 2 1 2 2 2 2 1 2 2 2 2 1 2 1 1 2 2 2 2 2 2 2 1 2 2 2 2 1 2 2 2 1 2 2 1 2 2 2 2
[107] 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 3 3 1 3 3 3 3 3 3 3 3 3 3 3 2 3 2 3 3 3 3 3 1 3 3 3 3 3
[160] 3 3 3 3 3 3 3 2 3 3 1 3 3 3 3 2 2 2 3 1
>
> # Cluster modes
> wines_kmode3$modes
       Class     Alcohol Malic_acid        Ash Alcalinity_of_ash   Magnesium Total_phenols    Flavanoids
1 -1.21052889  0.06082829 -0.5427655 -0.3153590        -1.0465271  0.08810981     1.1262866  0.6514738740
2  0.07973544 -1.13400823 -0.7396958 -1.4088766        -0.1482061 -0.96212770    -0.1519728  0.0007311716
3  1.36999976 -0.92460389  0.6209140 -0.1695567         0.1512342 -0.82209603    -1.3024063 -1.4509256261
  Nonflavanoid_phenols Proanthocyanins Color_intensity        Hue OD280.OD315of_diluted_wines
1           -0.7380592      -0.4208878     -0.54268108  0.3611585                   0.5469294
2            0.5475632      -0.2287007     -1.06030491 -0.1200912                  -1.4812670
3            1.3510772      -1.1372216     -0.02505726 -1.6950903                  -1.8052150
>
> # Cluster size
> wines_kmode3 $size
cluster
 1  2  3
75 63 40
>
> # Between-cluster sum of squares
> wines_kmode3 $betweenss
NULL
>
> # Within-cluster sum of squares
> wines_kmode3 $withindiff
[1] 848 700 435
```

```
> # Iterations
> wines_kmode3 $iterations
[1] 3
>
> # Mean values of each cluster
> aggregate(wines, by=list(wines_kmode3$cluster), mode)
  Group.1   Class Alcohol Malic_acid     Ash Alcalinity_of_ash Magnesium Total_phenols Flavanoids
1       1 numeric numeric    numeric numeric           numeric   numeric       numeric    numeric
2       2 numeric numeric    numeric numeric           numeric   numeric       numeric    numeric
3       3 numeric numeric    numeric numeric           numeric   numeric       numeric    numeric
  Nonflavanoid_phenols Proanthocyanins Color_intensity     Hue OD280.OD315of_diluted_wines
1              numeric         numeric         numeric numeric                      numeric
2              numeric         numeric         numeric numeric                      numeric
3              numeric         numeric         numeric numeric                      numeric
>
> # Clustering
```

```
>
> #checking
> table(data_$class,wines_kmode3$cluster)

    1  2  3
  1 58  0  1
  2 13 58  0
  3  4  5 39
>
```



3. Cluster quality comparison:

Following table is to measure the overall performance of the K-mean and K-mode clustering on wine data set. In wine data, we have three clusters where number count for each cluster is consecutively 59, 71 and 48.

```
> #checking
> table(data_$class,wines_kmean3$cluster)

    1  2  3
1  57  2  0
2   4 66  1
3   0  0 48
> # Execution of k-mode with k=3
```

```
>
> #checking
> table(data_$class,wines_kmode3$cluster)

    1  2  3
1  58  0  1
2  13 58  0
3   4  5 39
>
```

From K-mean algorithm output it is found that it can classify 57 cluster-1 items correctly whereas K-mode can classify 58 items correctly. Hence performance and purity of cluster-1 using K-mode is better than K-mean algorithm. But for the other two clusters K-mean does a way better classification compared to K-mode. Cluster-2 purity using K-mode is .81 but K-mean gives purity score 0.85. Again K-mean algorithm classifies cluster-3 elements perfectly and therefore the purity score is 1 but K-mode gets purity score .81 by classifying 39 items correctly.

Clusters	**Wine Data Cluster Count**	**K-mean: Cluster Purity**	**K-mode: Cluster Purity**
Cluster 1	**59**	Cluster 1: Purity = 1/59(max(57,2,0))= **.96**	Cluster 1: Purity = 1/59(max(58,0,1))= **.98**
Cluster 2	**71**	Cluster 2: Purity = 1/71(max(4,66,1))= **.85**	Cluster 2: Purity = 1/71(max(13,58,0))= **.81**
Cluster 3	**48**	Cluster 3: Purity = 1/48(max(0,0,48))= **1**	Cluster 3: Purity = 1/48(max(4,5,39))= **.81**

From the above analysis and discussion it is clear that we can use K-mode classification by considering ‘most’ rather than using ‘mean’. But for wine data set K-mode classification does not perform well compared K-mean. Therefore choosing ‘most’ _ wine to represent a cluster might give some misleading information by misclassifying some members compared to K-mean.

3. [a] 10 points

In the class, we discussed about the data becoming sparse when we add dimensions (curse of dimensionality). Which means, we need more training samples when there are more dimensions, to avoid overfitting. Suppose an infant needs only a sample of 20% of the population to learn to recognize if it is a rose or a sunflower just by the smell. What percent of the population will the infant need for each feature, if a 2nd feature, the shape was also used to do the classification?

How about a 3rd feature, the color was added? Explain your answers.

[Hint: Think square for two features and a cube for three features in the feature space]

Answer: In general, it is obvious that adding more input features or dimension to a fixed number of training samples may increase overfitting. It happens most of the time because more features are either irrelevant or redundant and make the model complex in order to fit the data. Keeping practical concerns of memory and processor time aside, adding more training samples to avoid overfitting if we need to increase feature.

For the above mentioned problem, training set row numbers are fixed, say 100. So an infant can classify rose and/or sunflower by looking at the 20% sample of the total population just by considering ‘smell’ feature. Though adding 2nd feature ‘shape’ will make feature space to square size but will definitely give better classification result. Now if we add a 3rd feature,’color’, the feature space will become three dimensional cube. It may also happen now identifying rose or sunflower becomes a linearly separable problem in 3d space. So the more features we add, the higher the likelihood that infant can successfully identify rose or, sunflower perfectly.

From the above discussion it might seem that adding more features until we get the perfect classification is the best way to train a classifier model, but the reality is that by

adding more features we start to lose generality and get worse performance on unseen data due to overfitting model.

Now let's see the effect of adding features on a fixed sized training sample. For example, consider that in the 1D case (when we have only one feature to consider), 100 training sample rows covered the complete 1D feature space. Say the interval is 10. Therefore the sample density is 100/10 = 10 sample per interval. After adding 'shape' feature, the resulting feature space is 2D. In the 2D case, we still have 100 training samples which covers the 2D space with an area of 10*10= 100 unit squares. Therefore in the 2D space, the sample density is 100/25 =4 samples/interval. Finally, in the 3D feature space, the 100 training samples will cover a feature space of volume 10*10*10=1000 unit cubes. Now for 3D case, the sample density will be 100/1000= 0.1 samples/interval. So the density of the training model decrease exponentially when we increase the feature or dimensionality of the training dataset.

Now to explain the above scenario in a different manner, let's assume that infant is only using 'smell' feature whose value ranges from 0 to 1. Also assume that, this feature is unique for rose and/or sunflower. Now as an infant need only 20% sample of the population to learn to recognize rose/sunflower, so our training data will cover 20% of this range. After adding 2nd feature in a 2D feature space, we now need to obtain **45%** of the complete training population to cover 20% of the 2D feature space in each dimension because .45 *.45 = 0.2. So the infant will need 45% of the population for each feature after adding 2nd feature. After adding 3rd feature 'color', the situation gets worse because to cover 20% of the 3D feature space, the infant will need to obtain [0.58 *0 .58 * 0.58 = 0.2] **58%** of the population in each dimension. So the amount of training sample data needed to cover 20% of the feature space increases exponentially with the increase of features.

[b] 5 points

Can PCA help to reduce the computation time required to build a Machine Learning model? What about when all eigen values are more or less equal to each other? Explain.

Answer:

PCA is one kind of dimensionality reduction algorithm. Usually dimensionality reduction algorithms are used to reduce the computation time required to build a machine learning

model. Because reducing the dimension of data will take less time to train a model. So definitely PCA helps to reduce the required computation time to build a machine learning model.

Suppose for an nxn dimension matrix M eigenvalues are all 1 or all 0, means more or less similar. If M has n eigenvalues with all 1's, then M is equal to the identity matrix because $M - ??I = 0$ If M has n eigenvalues that are all 0's, then M is equal to zero matrix. Moreover if eigenvalues are more or less equal to each other that we will not get n different eigenvectors. Without having n different eigenvectors we do not have a basis to compute n different principal components which hampers the whole concept of PCA algorithm.

[c] 5 points

Explain how k-NN algorithm can be used for imputing missing values in case of (i) categorical and (ii) continuous variables.

Answer:

K-Nearest Neighbour (k-NN) is a non-parametric approach to estimate the class of data point. For any given point X, we find the k nearest neighbours to x in the given data based on different distance metric (euclidean) and assign the most common class among the majority k neighbours to X. k-NN is widely used to impute missing values. In this method, k neighbors with non-missing values are chosen based on some distance metric and their average/mean/weighted mean etc. is used as an imputation estimate to fill the missing value. So we will train the model and classify using the non missing values and will classify the missing values using the model. Different distance measures are being used for categorical and continuous variables.

(i) Categorical variables: Generally hamming distance is used for categorical variables. This method considers all the categorical attributes and count one if the value is not the same between two points and does this for each attribute. The hamming distance is then equal to the number of attributes for which the value is different. In the existing library packages there is a parameter called cat_nm where we have to define the function for aggregation the k-NN in the case of categorical variables.

(ii) Continuous variables: Usually k-NN uses euclidean distance to find k-NN nearest neighbours. Other commonly used distance measures are Manhattan and cosine.

We need to carefully select the value of k for k-NN imputation.

4. [a] 5 points

What is the rationale behind having multiple algorithms for doing the same kind of prediction or classification? Why not have just one algorithm?

Answer:

The main reason behind having different algorithms for doing the same kind of prediction or classification is data type. Every data is different in its own way. So different characteristics of data demands different algorithm. For example, some algorithms can only handle linear data type and some can handle both. Some algorithms perform well with even big volume of data and some are meant for small dataset.

Other important reasons are ‘performance’ and ‘accuracy’. When we have multiple algorithms to apply on the same data set, we can easily compare their performance and accuracy and can choose the most appropriate algorithm for that dataset. To draw a conclusion, I have implemented 4 classification algorithms on IRIS data set. From the output its clear that why it is benefited to have multiple options for classification algorithm. Same goes for prediction also.

File: hw4_IRIS_data_with_multiple_ClassificationAlgo

Observation: I have applied SVM, Decision Tree and Random forest to classify IRIS data set to measure performance and accuracy:

n [5]:
```
data  =  pd.read_csv("C:/Users/subar/Downloads/CMPE-255 Sec 99 - Data Mining/Home Works/HW3_Submission/IRIS.csv")

X  = data[['sepal_length','sepal_width','petal_length','petal_width']]
y= data['species']
```

n [6]:
```
#Split the Data into Training and Testing sets with test size as #30%
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3, shuffle=True)
```

[12]:
```
classifiers=[]
#SVM
model1 = svm.SVC()
classifiers.append(model1)

#Decision Tree
model2 = tree.DecisionTreeClassifier()
classifiers.append(model2)

#Random Forest
model3 = RandomForestClassifier()
classifiers.append(model3)
```

Here computational performance of all three algorithms are almost the same as IRIS is a small dataset but accuracy is different. From the below result it is clear that SVM and Decision tree classifiers are outperforming Random Forest [accuracy-93%] with 97% accuracy.

```
Accuracy of SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='rbf', max_iter=-1, probability=False, random_state=None,
    shrinking=True, tol=0.001, verbose=False) is:
97.77777777777777
Confusion Matrix:
[[15  0  0]
 [ 0 16  1]
 [ 0  0 13]]
**************************************
Accuracy of DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best') is:
97.77777777777777
Confusion Matrix:
[[15  0  0]
 [ 0 16  1]
 [ 0  0 13]]
**************************************
Accuracy of RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False) is:
93.33333333333333
Confusion Matrix:
[[15  0  0]
 [ 0 16  1]
 [ 0  2 11]]
**************************************
```

Hence we need multiple algorithms for same type of prediction or classification so that we can compare and decide.

[b] 5 points

Suppose we obtain the following weight vectors when we run logistic regression:

- **w1 = (389572, −973456, 43567, 874651, −9875)**
- **w2 = (2.956, −0.187, 1.49, 3.449, −1)**

For which of these weight vectors is small changes between test instances likely to make large changes in classification? Which of these models do you think generalizes better and why?

Answer: Weight vectors with large values will make a large impact on classification with slight changes. So here **w1 = (389572, −973456, 43567, 874651, −9875),** is likely to make large changes in classification if we make small changes there. This is because

too much importance is given in each feature that small change in data will cause probability to vary a lot mostly because of overfitting.

Model with small weights will generalize better. So **w2 = (2.956, −0.187, 1.49, 3.449, −1)** will generalize more that the weight vector with large value. Because model with small weight values are mostly free from the risk of overfitting. But model with large weight values has a tendency of overfitting because too much confidence is given to the features that their small change will vary the probability a lot. It is also obvious from mathematical representation of the logistic regression given below [Screenshot is from our class lecture slide]. Here sigmoid function is directly proportional to the weight function. So the larger the weight value, the larger the chance of varying probability with small change.

- $f(X) = w_0 + \sum_{i=1}^{n} w_i x_i$ .....(2)
- $\text{logistic}(z) = \frac{e^z}{1+e^z}$ .....(3)
- $P(Y=1|X) = \frac{e^{w_0 + \sum w_i x_i}}{1 + e^{w_0 + \sum w_i x_i}}$ ...(4)

That's why we always try to achieve smaller weight by maximizing or minimizing objective function.

[c] 5 points

What is the one characteristic among the features of data that makes dimensionality reduction possible and necessary?

Answer:

Correlation among the features of data makes dimensionality reduction possible and necessary. As correlation is normalized covariance value, so in other words we can say that covariance is the required feature. When comparing different features of data, correlation is the unit measure of change between two attribute change with respect to each other and covariance measures change of how the variables are changing with respect to each other. This concept is very important for dimensionality reduction.

Because in dimensionality reduction goal is to reduce the dimension without compromising information. To make this happen we need to know the relation among the features. That’s why correlation/covariance is important to measure how strong the relationship is between the two features before reducing them.