Assignment4_Solutions

May 18, 2021

```
In [87]: import numpy as np
         import math
         import pandas as pd

         from scipy.stats import poisson, sem, poisson, ttest_ind, shapiro, mannwhitneyu
         import scipy.stats as stats # for 'f_oneway'
         from scipy.cluster.hierarchy import cophenet
         from scipy.spatial.distance import pdist

         from sklearn import linear_model
         from sklearn.cluster import AgglomerativeClustering
         from sklearn.decomposition import PCA
         from sklearn.model_selection import train_test_split
         from sklearn.linear_model import LogisticRegression
         from sklearn.metrics import brier_score_loss

         import statsmodels.formula.api as sm # get ANOVA table as R like output
         from statsmodels.formula.api import ols # Ordinary Least Squares (OLS) model

         import matplotlib.pyplot as plt
         from IPython.display import display, HTML
         from scipy.cluster.hierarchy import dendrogram, linkage
         from mpl_toolkits.mplot3d import Axes3D
```

0.1 Biomedical Data Science & AI

24/25

0.2 Assignment 4

Group members: Fabrice Beaumont, Fatemeh Salehi, Genivika Mann, Helia Salimi, Jonah

---

0.2.1 Exercise 1 - ANOVA *F*-test and Hierarchical Clustering

7/8 points

Load the leukemia dataset (`leukemia.csv`). It contains gene expression data of 1397 genes from 38 tumor mRNA samples. The expression data is organized in a matrix where rows correspond to genes and columns to samples. The tumor class of the columns is given in the file `golub.cl`.

```
In [88]: leukemia_db = pd.read_csv('https://raw.githubusercontent.com/D34dP0oL/4216_Biomedical_
         leukemia_db.head(4)

Out[88]:                                 V1       V2  ...      V37      V38
         gene_name                                    ...
         AFFX-HUMISGF3A/M97935_3_at  0.45695 -0.09654  ...  0.90774  0.46509
         AFFX-HUMTFRR/M11507_5_at   -0.56223  0.05358  ...  0.44808  1.19275
         AFFX-M27830_M_at            2.40116  1.83222  ...  1.87913  2.49036
         AFFX-HUMGAPDH/M33197_3_st   0.10806  0.08245  ... -0.11911  0.48378

         [4 rows x 38 columns]

In [89]: leukemia_db.shape

Out[89]: (1397, 38)

In [90]: tumor_class_db = pd.read_csv('https://raw.githubusercontent.com/D34dP0oL/4216_Biomedic
         tumor_class_db.head(4)

Out[90]:    x
         1  0
         2  0
         3  0
         4  0

In [91]: ### Lets check, that exactly two types of Leukemia (0 and 1) are identified
         tumor_class_db.values.T

Out[91]: array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])

In [92]: tumor_class_db.shape

Out[92]: (38, 1)
```

1.1. ANOVA *F*-test

1.1.a. What are the assumptions of the ANOVA *F*-test? To conduct an ANOVA *F*-test one assumes that: - **Homoscedasticity** - The variances in the studied groups of samples is the same - **Independant samples** - The samples are randomly and independantly selected for each group. - **Normal distribution** - All groups of samples are normally distributed. - A linear model can sufficiently fit the group means (**no heteroscedastic noise**). - The residuals are normally distributed.

Sidenote to the foreign words: - A vector of random variables is **homoscedastic**, if all its random variables have the same finite variance. - A vector of random variables is "heteroscedastic", if the variability of the random disturbance is different across elements of the vector

(Greek: *hetero* "different", *homo* "equal", *skedasis* "dispersion")

1 point

1.1.b. For each gene in the dataset, perform the ANOVA *F*-test (assumptions are already met) to see whether the gene is significantly differentially expressed between the two types of Leukemia.

```
In [93]: ### For every gene we conduct an ANOVA F-test with the same null hypothesis:
         ### Null hypothesis: "The group means of type 0 of leukemia and type 1 of leukemia ar
         print("The genes that are significantly differentially expressed between the two type
         ### Store all signifiant genes in a dataframe
         signif_genes = pd.DataFrame(columns=['gene_name', 'p-value', 'F-value'])

         ### From the 'tumor_class_db' get the indices of the two types of leukemia
         type_zero_indices = tumor_class_db.index[tumor_class_db['x'] == 0].tolist()
         type_one_indices = tumor_class_db.index[tumor_class_db['x'] == 1].tolist()
         ### Since the 'tumor_class_db' counts the first row as '1' and not '0', we need to sh
         type_zero_indices = [x - 1 for x in type_zero_indices]
         type_one_indices = [x - 1 for x in type_one_indices]

         ### Iterate through all genes
         for gene in leukemia_db.index:
             ### Differentiate all columns/ samples that belong to the two different types of
             gene_expressions_type_zero = leukemia_db.loc[gene].iloc[type_zero_indices].tolist
             gene_expressions_type_one = leukemia_db.loc[gene].iloc[type_one_indices].tolist()

             ### Stats 'f_oneway' functions takes the two groups as input and returns ANOVA F-
             fvalue, pvalue = stats.f_oneway(gene_expressions_type_zero, gene_expressions_type
             ### A p-value < 0.05 is significant. Print the name of the gene if that is the ca
             if pvalue < 0.05:
                 signif_genes = signif_genes.append({'gene_name' : gene, 'p-value' : pvalue, 'F

         signif_genes
```

```
The genes that are significantly differentially expressed between the two types of Leukemia are
```

```
Out[93]:                     gene_name   p-value    F-value
         0    AFFX-HUMTFRR/M11507_5_at  0.000072  20.086635
         1                 AB000449_at  0.000850  13.247115
         2                 AB000468_at  0.039889   4.545975
         3             AC000064_cds1_at  0.022949   5.644936
         4                 AF008937_at  0.011464   7.099773
         ..                        ...       ...        ...
         480                 U40279_at  0.002999  10.133038
         481                 X17093_at  0.025641   5.420034
         482                 Z30643_at  0.044146   4.350522
         483                 U04241_at  0.044844   4.320449
         484                 X51345_at  0.037074   4.688270

         [485 rows x 3 columns]
```

1 point

Note: don't forget about the Family-wise error rate. You could use Bonferroni Holm correction.

```
In [94]: nr_all_genes = len(leukemia_db)
         nr_signif_genes = len(signif_genes)
         signif_percentage = nr_signif_genes / nr_all_genes
         print(f"Out of the {nr_all_genes} genes, {nr_signif_genes} are significantly differen
         print(f"That is about {round(signif_percentage*100, 2)}%")
```

```
Out of the 1397 genes, 485 are significantly differentially expressed between the two types of
That is about 34.72%
```

1.1.c. Due to our analysis, we now know which genes are significantly differentially expressed between groups. These will be the best features to use in order to get good cluster separation. Subset only the rows which represent the top 100 most significant genes.

```
In [95]: ### The genes are more significant, the lower their p-value is.
         ### Thus return the 100 genes with the lowest p-value:
         most_signif_100_genes = signif_genes.nsmallest(100, 'p-value')
         most_signif_100_genes
```

```
Out[95]:            gene_name       p-value    F-value
         155         M63138_at  2.366804e-08  50.530583
         243     U50136_rna1_at  2.519519e-08  50.235874
         478         M31523_at  2.713676e-08  49.887317
         124         M16038_at  4.802350e-08  47.255323
         173         M92287_at  6.046048e-08  46.217096
         ..                ...           ...        ...
         137         M28209_at  2.294286e-04  16.754857
         169         M83221_at  2.310119e-04  16.735696
         219         U30521_at  2.363345e-04  16.672289
         422  HG627-HT5097_s_at  2.443697e-04  16.579380
         371         Z35102_at  2.634902e-04  16.370725

         [100 rows x 3 columns]
```

1 point

1.2. Plot 2 dendrograms using the 100 selected genes:

```
In [96]: ### See this awesome website for further information:
         ### https://joernhees.de/blog/2015/08/26/scipy-hierarchical-clustering-and-dendrogram
```

1.2.a. One for a single linkage approach and another one for ward approach.

```
In [97]: most_signif_100_genes_db = leukemia_db.loc[most_signif_100_genes['gene_name']]
         most_signif_100_genes_db.head(4)
```

```
Out[97]:                      V1       V2       V3  ...      V36      V37      V38
         gene_name                                   ...
         M63138_at       0.98318  1.39165  1.46391  ...  2.60321  2.31917  1.50779
         U50136_rna1_at  0.77407  0.69785  0.85670  ...  1.65866  1.43275  1.51216
         M31523_at       0.92234  0.54933  0.10862  ... -1.20703 -0.79439 -0.40721
```

```
M16038_at      -0.26342  0.22701 -1.39460  ...  1.34766  1.38402  0.54227

[4 rows x 38 columns]
```

```
In [98]: ### The 'linkage' function returns a matrix which rows denote an cluster merging-itera
         ### Every row stores the indixes of the merged clusters, their distance and the numbe

         ### Singel linkage approach
         dendogram_single = linkage(most_signif_100_genes_db, 'single')
         ### Ward approach
         dendogram_ward = linkage(most_signif_100_genes_db, 'ward')
```

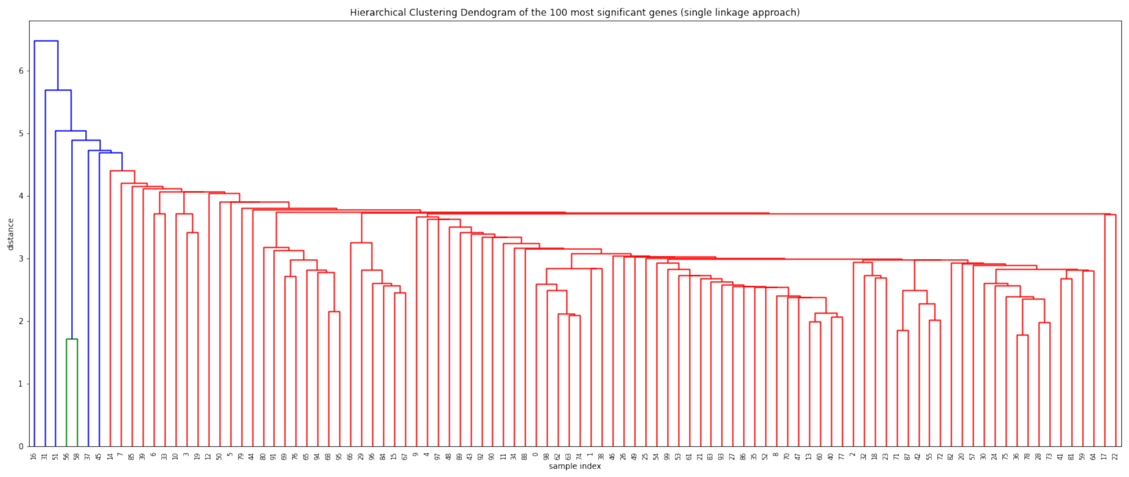
most_signif_100_genes_db.transpose()

```
In [99]: def plot_dendogram(data, name=None, x_label=None, y_label=None, truncate_nr=None):
             plt.figure(figsize=(25, 10))
             plt.title('Hierarchical Clustering Dendrogram')
             plt.xlabel('sample index')
             plt.ylabel('distance')
             if name is not None:
                 plt.title(name)
             if x_label is not None:
                 plt.xlabel(x_label)
             if y_label is not None:
                 plt.ylabel(y_label)

             dendrogram(
                 data,
                 leaf_rotation=90.,  # rotates the x axis labels
                 leaf_font_size=8.,  # font size for the x axis labels
             )
             if truncate_nr is not None:
                 dendrogram(
                     data,
                     truncate_mode='lastp',  # show only the last p merged clusters
                     p=truncate_nr,  # show only the last p merged clusters
                     show_leaf_counts=False,  # otherwise numbers in brackets are counts
                     leaf_rotation=90.,
                     leaf_font_size=12.,
                     show_contracted=True,  # to get a distribution impression in truncated bra
                 )

             plt.show()
```

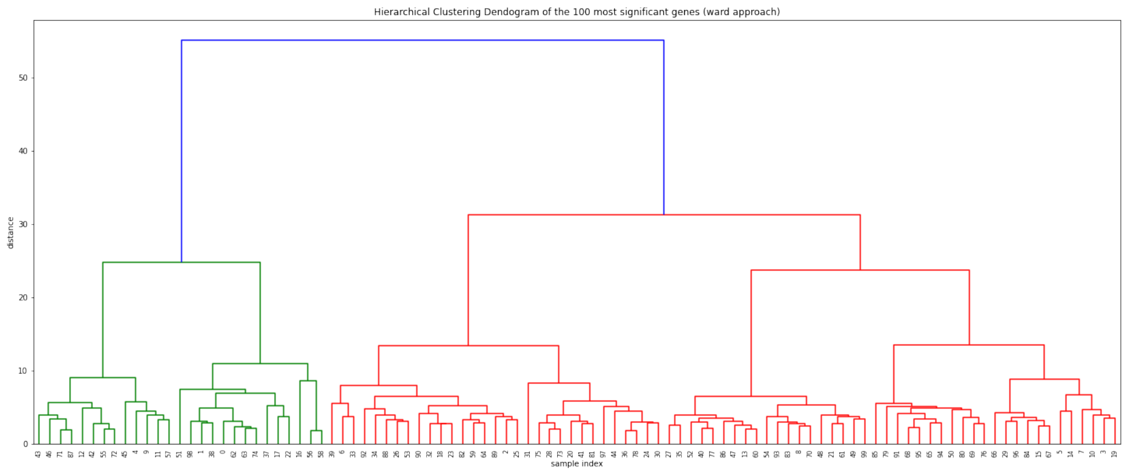
0.5 point

We want to cluster the tumors and not the genes.

```
In [100]: plot_dendogram(dendogram_single, "Hierarchical Clustering Dendogram of the 100 most s
```



```
In [101]: plot_dendogram(dendogram_ward, "Hierarchical Clustering Dendogram of the 100 most si
```



1.2.b. Which method would you recommend based on the dendrograms for a clustering? Why? The dendogram created using the **ward approach is better**, since the hierarchical tree is more equally distributed/merged. With the single linkage approach, a lot of clusters are successively growing by merging with a single gene.

The Single linkage method and Ward criterion method vary on the basis of the metric used to define the proximity between two clusters.We would prefer to obtain clusters which are more well separated from each other and where each cluster must contain data points which are similar to other data points within their cluster. Based on the dendrograms, the **Ward method** for clustering has created clusters which are more well separated as shown in dendrogram.

Also, since we know that there are two leukemia classes and the Ward method yields two rather distinct clusters, this reflects the leukemia classification.

1 point

1.2.c. Familiarize yourself with Cophenetic correlation coefficient and calculate the cophenetic correlation distance for both single linkage as well as ward. The Cophenetic correlation coefficien indicates how well the dendograms preserve the actual distances of the data. The closer the coefficient is to one, the better the preservation.

```
In [102]: c_single, coph_dists_single = cophenet(dendogram_single, pdist(most_signif_100_genes_
          c_ward, coph_dists_ward = cophenet(dendogram_ward, pdist(most_signif_100_genes_db))
          print(f"The Cophenetic correlation coefficient for the ward approach is          ~{
          print(f"The Cophenetic correlation coefficient for the single linkage approach is ~{
```

```
The Cophenetic correlation coefficient for the ward approach is           ~0.71         (0.712
The Cophenetic correlation coefficient for the single linkage approach is ~0.29         (0.292
```

1 point

1.2.d. Based on the cophenetic correlation distance, which clustering method performed better?

```
In [103]: if c_ward < c_single:
              name = "single linkage"
          else:
              name = "ward"

          print(f"The {name} approach is better, since its Cophenetic correlation coefficient
```

```
The ward approach is better, since its Cophenetic correlation coefficient (preservation of the
```

1 point

1.3. Apply two Agglomerative Clustering.

1.3.a. One using single linkage and one using ward method.

most_signif_100_genes_db.transpose()

```
In [104]: ### Single linkage approach
          agglomerative_single = AgglomerativeClustering(linkage='single').fit(most_signif_100_
          ### Ward approach
          agglomerative_ward = AgglomerativeClustering(linkage='ward').fit(most_signif_100_gene
```

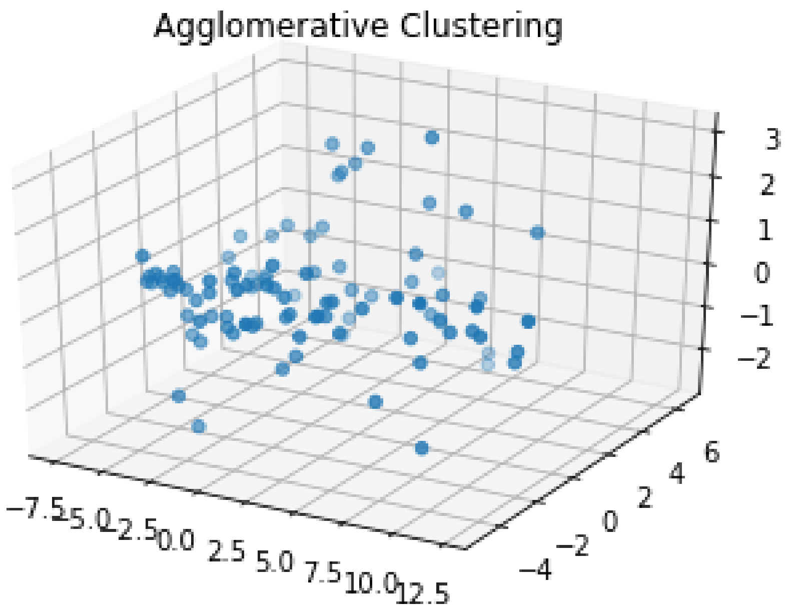
Visualizing this data is a bit tricky, since the feature vectors have 38 dimensions (number of samples). To still get an idea, that the agglomerative clustering was reasonable, we define a function that reduces the data to three dimensions and then display the agglomerative clustering as a heat map on this representation.

0.5 point

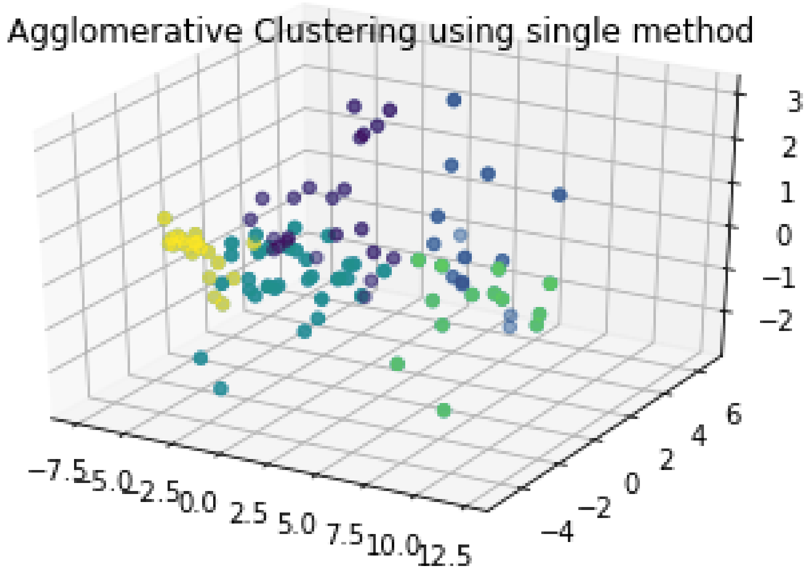
```
In [105]: def pca_scatter_plt(dataframe, colours=None, name=None):
              X = dataframe.values
              # Project data down to 3D
              pca = PCA(n_components=3)
              X_three_d = pca.fit_transform(X)

              # Now plot in 3D
              fig = plt.figure()
```

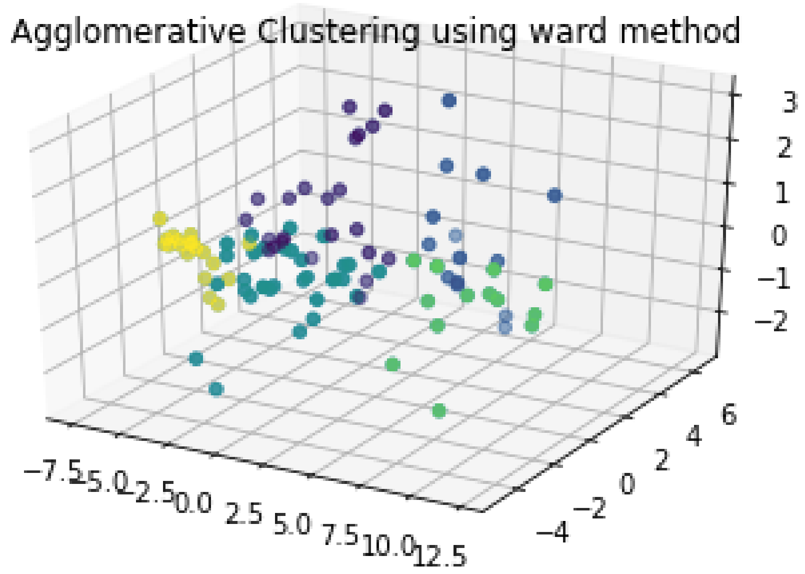
```
            ax = fig.add_subplot(111, projection='3d')

            plt.title(f"Agglomerative Clustering")
            if name is not None:
                plt.title(f"Agglomerative Clustering using {name} method")

            x = X_three_d[:,0]
            y = X_three_d[:,1]
            z = X_three_d[:,2]

            img = ax.scatter(x, y, z)
            if colours is not None:
                img = ax.scatter(x, y, z, c=colours)
            plt.show()
```

```
In [106]: pca_scatter_plt(most_signif_100_genes_db)
```



```
In [107]: for linkage in ['single', 'ward']:
              agglo_clustering = AgglomerativeClustering(affinity='euclidean', compute_full_tre
                      connectivity=None, linkage='ward', memory=None, n_clusters=5).fit(most_si

              pca_scatter_plt(most_signif_100_genes_db, agglo_clustering.labels_, linkage)
```





0.2.2 Exercise 2 - PCA 8/8 points

Using the same leukemia dataset generate the feature matrix (transposed leukemia dataset) and the class labels (`golub.cl.csv`).

```
In [108]: feature_matrix = leukemia_db.T
          feature_matrix.shape

Out[108]: (38, 1397)
```

2.1. Perform a PCA on the feature matrix and answer the following questions:

2.1.a. How many PC's do you need to explain at least 95% of the variance? 1 points

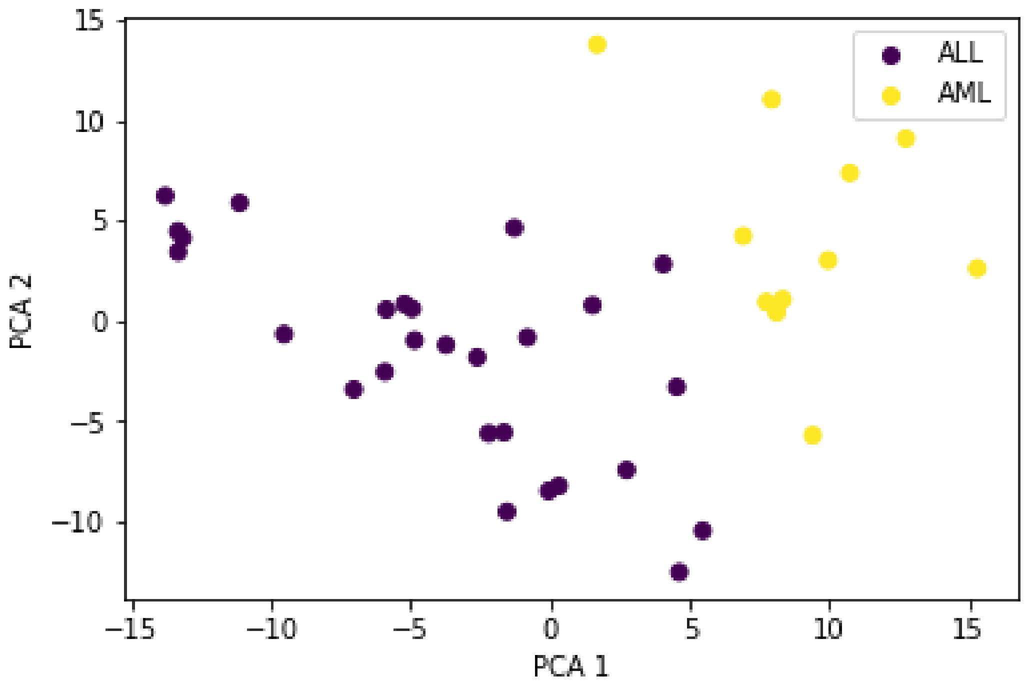
```
In [109]: ### Create PCA model and fit model to feature_matix
          pca = PCA()
          gene_pca_result = pca.fit_transform(feature_matrix)
          gene_pca_result

Out[109]: array([[-1.66097653e+00, -5.55344514e+00, -4.08752470e+00, ...,
                   1.20513583e+00, -3.42846960e-01,  4.87727279e-15],
                 [-1.28644441e+00,  4.63851571e+00,  3.39181451e+00, ...,
                   5.72539341e-01,  4.30725929e-03,  4.87727279e-15],
                 [-1.55187216e+00, -9.50775135e+00,  7.04167677e+00, ...,
                  -2.32465774e+00,  1.45084711e+00,  4.87727279e-15],
                 ...,
                 [ 1.07419931e+01,  7.37587603e+00,  6.36773890e-01, ...,
                   8.76250582e-01, -9.43719417e-02,  4.87727279e-15],
                 [ 1.27538160e+01,  9.09981623e+00,  1.31807025e+00, ...,
                  -2.17416840e-01, -2.62355679e+00,  4.87727279e-15],
                 [ 9.96351945e+00,  3.02486006e+00, -2.81442388e-01, ...,
                   1.18180814e+00,  5.45101470e-01,  4.87727279e-15]])

In [110]: ### Using 'pca.explained_variance_ratio' to obtain the percentage of variance explai
          sum = 0
          component_index = 0

          while sum < 0.95:
              tmp_ratio = pca.explained_variance_ratio_[component_index]
              print(f'Principal Component {component_index + 1} explains\t{round(tmp_ratio*100

              sum += tmp_ratio
              component_index = component_index + 1

          print( f'\nAnswer: We need {component_index} PC\'s to explain at least 95% of the va

Principal Component 1 explains        17.56% variance
Principal Component 2 explains        10.21% variance
Principal Component 3 explains        7.51% variance
Principal Component 4 explains        5.53% variance
Principal Component 5 explains        4.73% variance
Principal Component 6 explains        4.04% variance
Principal Component 7 explains        3.75% variance
Principal Component 8 explains        3.53% variance
Principal Component 9 explains        3.29% variance
Principal Component 10 explains        2.74% variance
Principal Component 11 explains        2.52% variance
Principal Component 12 explains        2.23% variance
Principal Component 13 explains        2.11% variance
Principal Component 14 explains        2.03% variance
Principal Component 15 explains        1.87% variance
Principal Component 16 explains        1.84% variance
Principal Component 17 explains        1.78% variance
Principal Component 18 explains        1.58% variance
Principal Component 19 explains        1.55% variance
Principal Component 20 explains        1.51% variance
Principal Component 21 explains        1.45% variance
Principal Component 22 explains        1.35% variance
Principal Component 23 explains        1.3% variance
Principal Component 24 explains        1.26% variance
Principal Component 25 explains        1.22% variance
Principal Component 26 explains        1.18% variance
Principal Component 27 explains        1.13% variance
Principal Component 28 explains        1.11% variance
Principal Component 29 explains        1.09% variance
Principal Component 30 explains        1.04% variance
Principal Component 31 explains        0.99% variance

Answer: We need 31 PC's to explain at least 95% of the variance in data.
```

```
In [111]: ### Now that we have the needed amount,
          ### we can compute the explained ration right away:
          pca = PCA(n_components=31)
          pcs = pca.fit_transform(feature_matrix)

          print(f'{round(np.sum(pca.explained_variance_ratio_)*100,2)}%')
```

```
95.02%
```

2.1.b. Make a scatterplot of the projections on the first two PC's with the colouring corresponding to the class labels. 2 points

```
In [112]: result_df = pd.DataFrame(np.hstack([pcs[:,:2],tumor_class_db]), columns = ['pc 1', '

          scatter = plt.scatter(result_df['pc 1'], result_df['pc 2'], c=result_df['label'])
          plt.xlabel('PCA 1')
          plt.ylabel('PCA 2')
          plt.legend(handles=scatter.legend_elements()[0], labels=['ALL', 'AML'])
          plt.show()
```

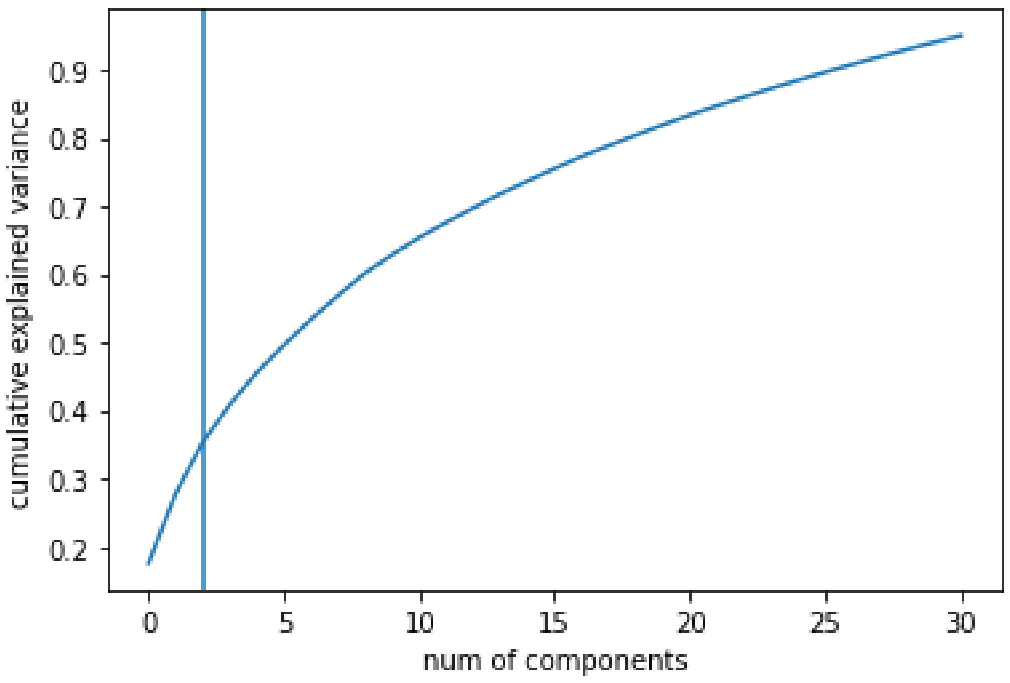


2.1.c. Based on the scatterplot answer the following questions: 2.1.c.1) Given the plot, do you think PCA might be a good choice? Why? 2 points

Yes, since as seen in the plot the first two principal components split the data into two distinct regions and is therefore a good tool in this case. - The large number of features of the dataset are reduced to just two Principal Components which are able to differentiate between the two classes. The dataset contained numerous features as compared to datapoints hence it is reasonable to apply a dimensionality reduction technique such as PCA. - All irrelevant information has been supressed and reduced to principal components which are not linearly correlated. - The first two PCA's are able to separate the data points of the two classes properly in the plot. - We are also able to plot the data points and view a visual representation because we have reduced the dimensions of the data.

2.1.c.2) Do you think n=2 components are a good choice? Why?

```
In [113]: plt.plot(np.cumsum(pca.explained_variance_ratio_))
          plt.xlabel('num of components')
          plt.ylabel('cumulative explained variance')
          plt.axvline(x=2)

Out[113]: <matplotlib.lines.Line2D at 0x7f670a8a0310>
```



In the plot we can see that the slope of the function on how many components explain how much variance is still very steep at more than two principal components. We would recommend to use at *least 7 components* in this case because at around 7-8 components the slope starts to decrease.

Also, since the two components are only able to explain around 27% of variance as indicated by the explained variance ratio it would be better to use more components. However these **two components** are able to **separate the two classes well** hence they are a **good choice for dimensionality reduction**.

2.2. Inform yourself regarding decorrelation of features in a dataset.

2.2.a. Identify the correlated features in the dataset. 1 points

```
In [114]: ### Calculating pairwise feature correlation in matrix
          correlated_features = set()
          corr_matrix = feature_matrix.corr().abs()
          for i in range(len(corr_matrix.columns)):
              for j in range(i):
                  if(corr_matrix.iloc[i,j]>0.8):
                      correlated_features.add(corr_matrix.columns[i])

          print(correlated_features)
```

```
{'X89101_s_at', 'HG4535-HT4940_s_at', 'Z80783_at', 'X83492_at', 'D14657_at', 'U50939_at', 'Z15
```

2.2.b. Decorrelate the correlated datasets. 1 points

```
In [115]: ### METHOD 1 - dropping correlated features of dataset
          decorrelated_dataset = feature_matrix.drop(labels=correlated_features, axis=1)
          decorrelated_dataset.head(4)

          ### METHOD 2 - It is also possible to prerform PCA
          ### as we have done in exercise 2.1.a to obtain PCAs which are not correlated

Out[115]: gene_name  AFFX-HUMISGF3A/M97935_3_at  ...  Z17240_at
          V1                            0.45695  ...   -0.35920
          V2                           -0.09654  ...   -0.43633
          V3                            0.90325  ...    0.34031
          V4                           -0.07194  ...   -0.90930

          [4 rows x 1320 columns]
```

2.2.c. What is the purpose of carrying out decorrelation of features in a dataset? 1 points

- Correlated features convey **redundant information** to models prepared for the dataset hence they must be decorrelated.
- Removal of correlated features also helps **reduce the dimension** of the dataset
- Linear models such as linear regression and logistic regression do not perform well when correlation is present in features.

0.2.3 Exercise 3 - Logistic Regression 6/9 points

3.1. Using the reduced dataset from exercise 2.1, carry out the following tasks:

3.1.a Generate a logistic regression model on the first 5 PCs of the reduced dataset using 80% of the total samples.

```
In [116]: ### Preparing dataset with first 5 PCAs
          column_names = ['pca_%i' % i for i in range(5)]
          gene_pca_dataframe = pd.DataFrame(gene_pca_result[:, 0:5], columns = column_names, i

          ### Preparing class label
          y = tumor_class_db

          ### Splitting into training and testing data
          X_train, X_test, y_train, y_test = train_test_split(gene_pca_dataframe, y, test_size
```

2 points

```
In [117]: ### Generating logistic regression model
          logistic_regression_model = LogisticRegression().fit(X_train, y_train['x'])
```

3.1.b. Predict the labels for the remaining 20% of the samples and calculate your model's accuracy

```
In [118]: ### Predicting labels for test data
          y_predicted = logistic_regression_model.predict(X_test)
          y_predicted

Out[118]: array([0, 0, 0, 1, 0, 1, 0, 0])
```

2 points

```
In [119]: ### Accuracy of model
          mean_accuracy = logistic_regression_model.score(X_test, y_test)
          print(f'Logistic Regression model has a mean accuracy of {mean_accuracy}')

Logistic Regression model has a mean accuracy of 1.0
```

3.2. Inform yourself about Brier's Score. How can it be used to evaluate the performance of your model? Show by implementation. Brier's Score calculates mean squared error between predicted probabilities and expected values. It summarises the magnitude of error in the probability forecasts. The score is between 0.0 and 1.0, with 0.0 being the best score.

```
In [120]: ### Prediction probabilities
          prediction_probabilities = logistic_regression_model.predict_proba(X_test)
          ### Now 'prediction_probabilities' stores the predicted probabilities for the
          ### negative label in the first and the positive in the second column

          brier_score_loss_result = brier_score_loss(y_test['x'].to_numpy(), prediction_probabi
          print(f"The Brier's score is almost {round(brier_score_loss_result,2)}, and thus pre

The Brier's score is almost 0.0, and thus pretty good. (2.1682642883096415e-05)
```

2 points

3.3. Assess the significance of your variables using the likelihood ratio test.

0 points