Machine Learning Capstone Project

I. Definition

A. Project Overview

This project is derived from cancer classification and prediction by gene expression at the molecular level. With a background in molecular biology and research experience in cancer treatment study, topics related to genetic engineering and biotechnology have always attracted my attention.

Acute lymphocytic leukemia (ALL) is a rare cancer in adults, but it is the most common form of leukemia in children. Acute myeloid leukemia (AML) is one of the most common leukemias in adults. They are developed from different types of white blood cells and involve different treatment strategies. Better classification can provide useful information for medical staff to identify cancer.

The dataset used in this project is provided by Kaggle (https://www.kaggle.com/crawford/gene-expression/download). It original comes from a research study by Professor Golub. The study described a generic method for automatically determining the type of cancer between acute lymphocytic leukemia (ALL) and acute myeloid leukemia (AML). The published paper showed the possibility of cancer classification based only on the gene expressions without relevant biological knowledge.

The dataset contains gene expression measurements corresponding to ALL and AML patient samples from bone marrow and peripheral blood. According to the paper, the intensity values have been rescaled so that overall intensities for each microarray chip are equivalent.

B. Problem Statement

The goal of the project is to find a good machine learning model to classify leukemia patients into acute lymphocytic leukemia (ALL) and acute myeloid leukemia (AML) based on his own gene expression.

The dataset contains three files which has been converted to comma separated value files.

1. actual.csv: each patients cancer type
2. data_set_ALL_AML_independent.csv: test data
3. data_set_ALL_AML_train.csv: training data

The input data format is a table of 7129 gene expressions from 72 patients with ALL or AML. The expression of each gene is a numeric value measured from DNA microarray.

The expected output of the classification model is a single categorical value, showing the cancer type ALL or AML of each patient.

Since each patient is labeled with its own cancer type, each data point is composed of gene expression values as input and cancer type as output. This is a discrete classification problem and supervised learning should be the solution.

The project tried widely used learning algorithms such as naive Bayes, logistic regression, Support Vector Machines, and Gradient Boosting models. Since the small dataset only has 72 data points, deep learning may not work well. The neural network model is not included in the project.

Each patient has 7129 different gene values within the dataset, which means 7129 dimensions of the input data. For higher-dimensional data like this, classification algorithms may have difficulty distinguishing noise from important features. Therefore, PCA analysis is a pre-processing to reduce the number of features and select the top principal components to be used in the classification model.

In addition, the input data from the microarray may be non-linear or not representative of the actual gene expression. To counter this, two standardization methods, *StandardScaler* and *MinMaxScaler*, are tested to understand how they change the data distribution.

In this dataset, the ratio of the two types of cancer is about 2:1, which is not a very high imbalance. However, it may be worth trying to see if Hyperparameter Tuner can improve the model.

The machine learning modeling workflow to solve the problem is as follows.

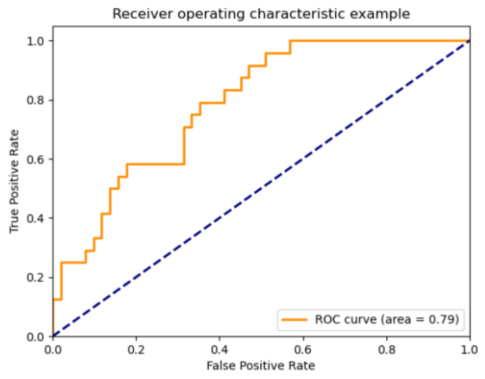
- Data Loading and Exploration
    - Data Loading
    - Exploratory Data Analysis
    - Data Cleaning and Pre-processing
- Feature Engineering and Data Transformation
    - Comparison of StandardScaler and MinMaxScaler
    - Standardization
    - Dimensionality reduction
- Training Model
    - Define and Create Estimator:
        - Benchmark Model
        - K-Means Clustering Model
        - K-Nearest Neighbors Model
        - Naive Bayes Model
        - Logistic Regression Model
        - Support Vector Machine Model
        - Gradient Boosting Model
    - Model tuning
    - Deploy the trained model
    - Evaluate the Performance
- Clean up Resources

C. Metrics

The supervised learning compares the actual output and predicted output of the model and use an error matrix with true positives, false positives, true negatives, false negatives to quantify its performance.

Since the imbalance of the dataset classes is not high and the AUC score is not sensitive to imbalance. The *roc_auc_score* and *accuracy_score* functions in the scikit-learn metrics library are used to calculate the AUC score and the accuracy score to measure the performance of a model and compare it with the benchmark.

The *roc_auc_score* function computes the area under the receiver operating characteristic (ROC) curve, which summarizes the curve information into one number.



The *accuracy_score* function computes the accuracy, which is the fraction of correct predictions.

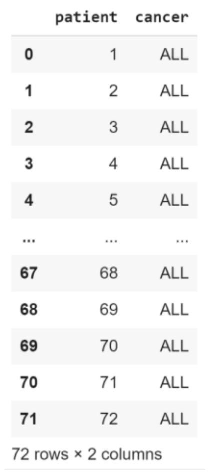
$$\texttt{accuracy}(y, \hat{y}) = \frac{1}{n_\text{samples}} \sum_{i=0}^{n_\text{samples}-1} 1(\hat{y}_i = y_i)$$

II. Analysis

A. Data Exploration

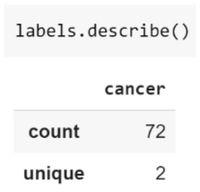
<u>Patients file:</u>

The first column is the patient id number; the second column is the cancer type. The total number of patients is 72; the cancer type is ALL or AML. The distribution of cancer types is 47 ALL patients and 25 AML patients.



	patient	cancer
0	1	ALL
1	2	ALL
2	3	ALL
3	4	ALL
4	5	ALL
...	...	...
67	68	ALL
68	69	ALL
69	70	ALL
70	71	ALL
71	72	ALL

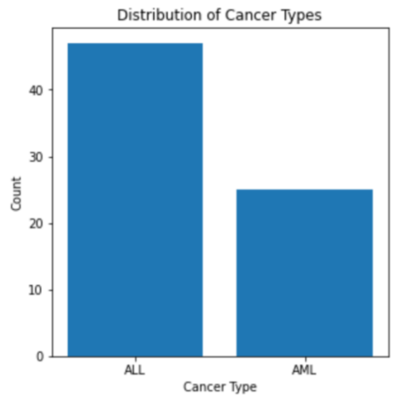
72 rows × 2 columns



```
labels.describe()
```

	cancer
count	72
unique	2

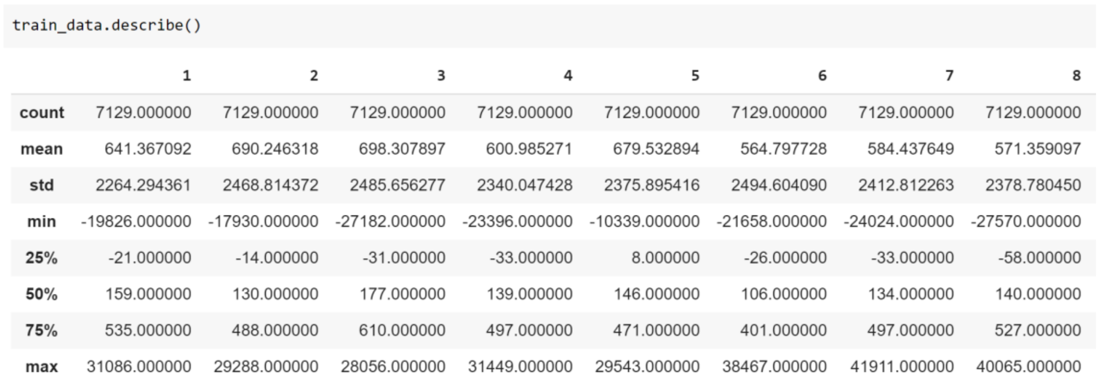
```
cancer
ALL    47
AML    25
dtype: int64
```



<u>Training data/test data file:</u>

The table has 7129 rows representing 7129 different genes. The first column is "gene description" and the second column is "gene accession number". The rest of columns represent the gene expression value of each patient. Each patient has two columns showing the value and category of each gene. The numeric value ranges from -30k to 30k. The category has three value A, P, and M.

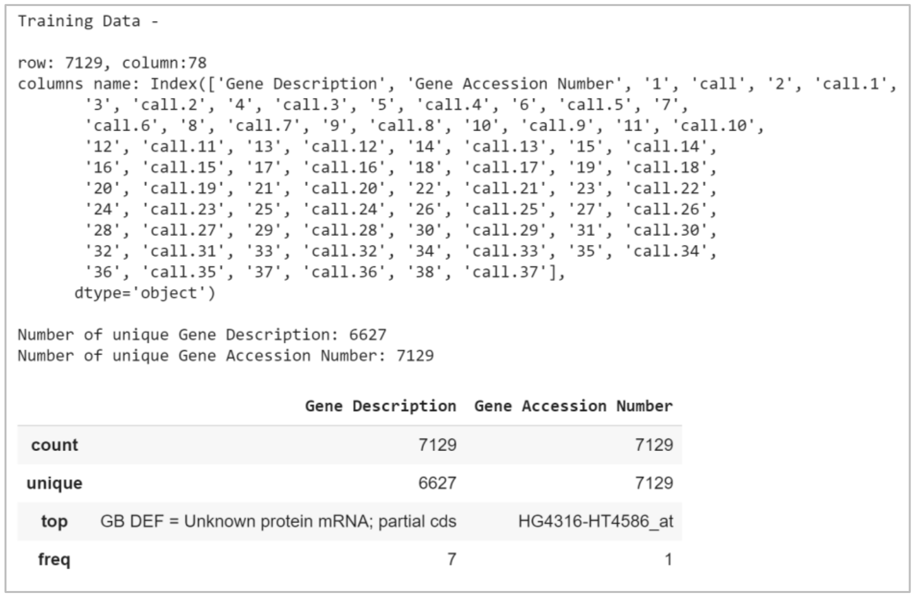
The gene accession number has a unique value, but the gene description does not. The training data contains patients 1 to 38. The test data includes patients 39 to 72.



```
train_data.describe()
```

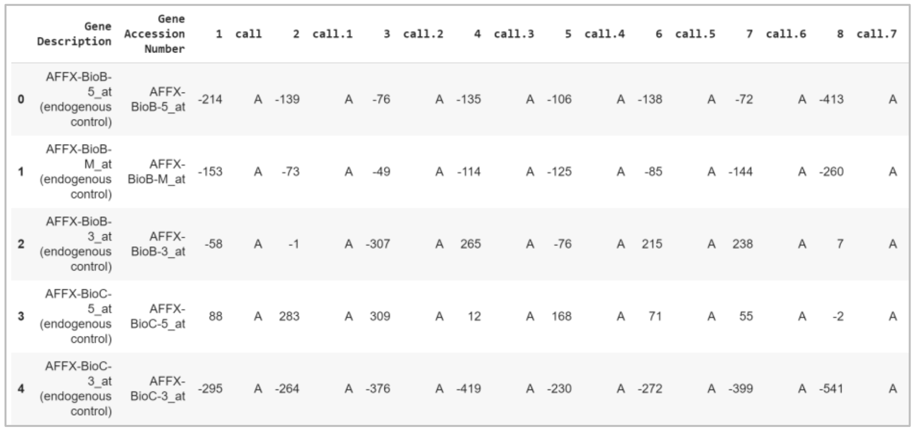
	1	2	3	4	5	6	7	8
count	7129.000000	7129.000000	7129.000000	7129.000000	7129.000000	7129.000000	7129.000000	7129.000000
mean	641.367092	690.246318	698.307897	600.985271	679.532894	564.797728	584.437649	571.359097
std	2264.294361	2468.814372	2485.656277	2340.047428	2375.895416	2494.604090	2412.812263	2378.780450
min	-19826.000000	-17930.000000	-27182.000000	-23396.000000	-10339.000000	-21658.000000	-24024.000000	-27570.000000
25%	-21.000000	-14.000000	-31.000000	-33.000000	8.000000	-26.000000	-33.000000	-58.000000
50%	159.000000	130.000000	177.000000	139.000000	146.000000	106.000000	134.000000	140.000000
75%	535.000000	488.000000	610.000000	497.000000	471.000000	401.000000	497.000000	527.000000
max	31086.000000	29288.000000	28056.000000	31449.000000	29543.000000	38467.000000	41911.000000	40065.000000

Training Data:



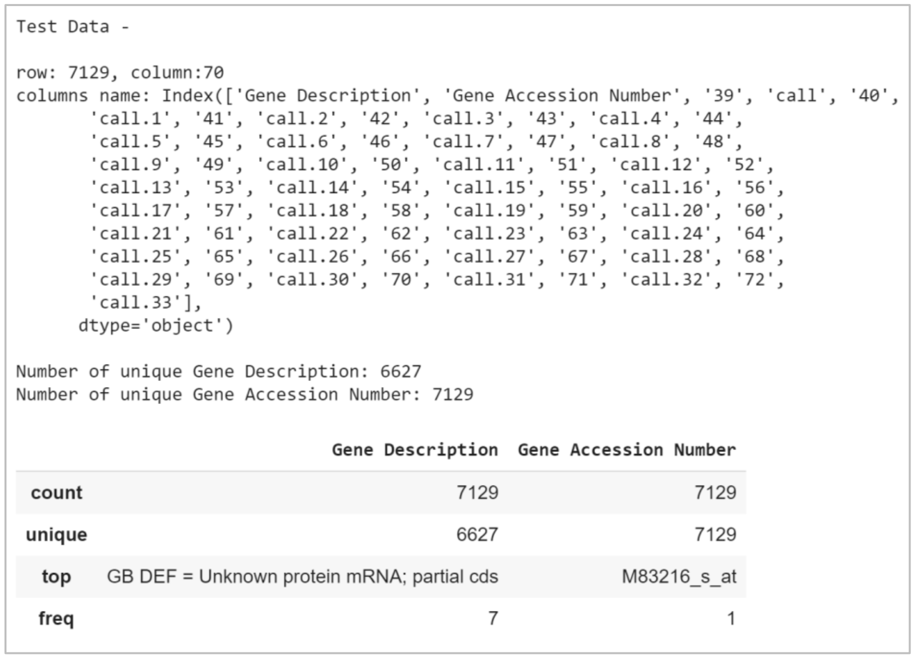
```
Training Data -

row: 7129, column:78
columns name: Index(['Gene Description', 'Gene Accession Number', '1', 'call', '2', 'call.1',
       '3', 'call.2', '4', 'call.3', '5', 'call.4', '6', 'call.5', '7',
       'call.6', '8', 'call.7', '9', 'call.8', '10', 'call.9', '11', 'call.10',
       '12', 'call.11', '13', 'call.12', '14', 'call.13', '15', 'call.14',
       '16', 'call.15', '17', 'call.16', '18', 'call.17', '19', 'call.18',
       '20', 'call.19', '21', 'call.20', '22', 'call.21', '23', 'call.22',
       '24', 'call.23', '25', 'call.24', '26', 'call.25', '27', 'call.26',
       '28', 'call.27', '29', 'call.28', '30', 'call.29', '31', 'call.30',
       '32', 'call.31', '33', 'call.32', '34', 'call.33', '35', 'call.34',
       '36', 'call.35', '37', 'call.36', '38', 'call.37'],
      dtype='object')

Number of unique Gene Description: 6627
Number of unique Gene Accession Number: 7129
```

	Gene Description	Gene Accession Number
count	7129	7129
unique	6627	7129
top	GB DEF = Unknown protein mRNA; partial cds	HG4316-HT4586_at
freq	7	1



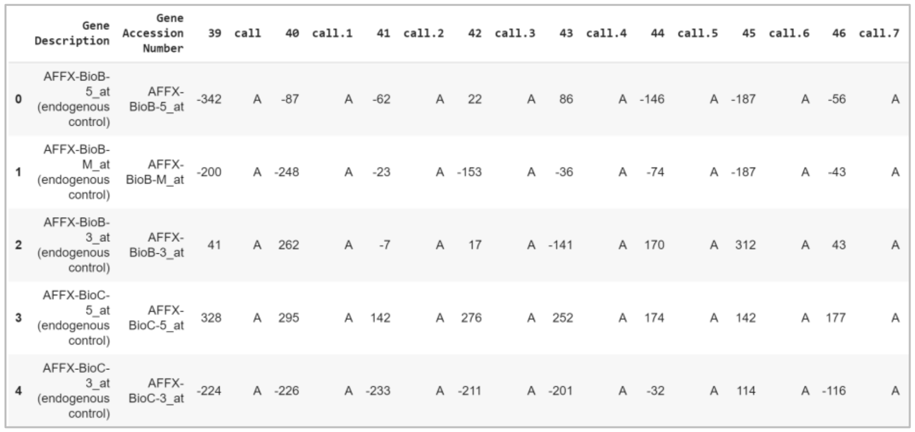
	Gene Description	Gene Accession Number	1	call	2	call.1	3	call.2	4	call.3	5	call.4	6	call.5	7	call.6	8	call.7
0	AFFX-BioB-5_at (endogenous control)	AFFX-BioB-5_at	-214	A	-139	A	-76	A	-135	A	-106	A	-138	A	-72	A	-413	A
1	AFFX-BioB-M_at (endogenous control)	AFFX-BioB-M_at	-153	A	-73	A	-49	A	-114	A	-125	A	-85	A	-144	A	-260	A
2	AFFX-BioB-3_at (endogenous control)	AFFX-BioB-3_at	-58	A	-1	A	-307	A	265	A	-76	A	215	A	238	A	7	A
3	AFFX-BioC-5_at (endogenous control)	AFFX-BioC-5_at	88	A	283	A	309	A	12	A	168	A	71	A	55	A	-2	A
4	AFFX-BioC-3_at (endogenous control)	AFFX-BioC-3_at	-295	A	-264	A	-376	A	-419	A	-230	A	-272	A	-399	A	-541	A

Test Data:



```
Test Data -

row: 7129, column:70
columns name: Index(['Gene Description', 'Gene Accession Number', '39', 'call', '40',
       'call.1', '41', 'call.2', '42', 'call.3', '43', 'call.4', '44',
       'call.5', '45', 'call.6', '46', 'call.7', '47', 'call.8', '48',
       'call.9', '49', 'call.10', '50', 'call.11', '51', 'call.12', '52',
       'call.13', '53', 'call.14', '54', 'call.15', '55', 'call.16', '56',
       'call.17', '57', 'call.18', '58', 'call.19', '59', 'call.20', '60',
       'call.21', '61', 'call.22', '62', 'call.23', '63', 'call.24', '64',
       'call.25', '65', 'call.26', '66', 'call.27', '67', 'call.28', '68',
       'call.29', '69', 'call.30', '70', 'call.31', '71', 'call.32', '72',
       'call.33'],
      dtype='object')

Number of unique Gene Description: 6627
Number of unique Gene Accession Number: 7129
```

	Gene Description	Gene Accession Number
count	7129	7129
unique	6627	7129
top	GB DEF = Unknown protein mRNA; partial cds	M83216_s_at
freq	7	1

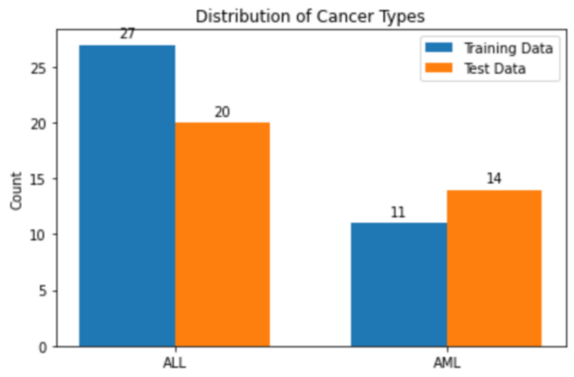


	Gene Description	Gene Accession Number	39	call	40	call.1	41	call.2	42	call.3	43	call.4	44	call.5	45	call.6	46	call.7
0	AFFX-BioB-5_at (endogenous control)	AFFX-BioB-5_at	-342	A	-87	A	-62	A	22	A	86	A	-146	A	-187	A	-56	A
1	AFFX-BioB-M_at (endogenous control)	AFFX-BioB-M_at	-200	A	-248	A	-23	A	-153	A	-36	A	-74	A	-187	A	-43	A
2	AFFX-BioB-3_at (endogenous control)	AFFX-BioB-3_at	41	A	262	A	-7	A	17	A	-141	A	170	A	312	A	43	A
3	AFFX-BioC-5_at (endogenous control)	AFFX-BioC-5_at	328	A	295	A	142	A	276	A	252	A	174	A	142	A	177	A
4	AFFX-BioC-3_at (endogenous control)	AFFX-BioC-3_at	-224	A	-226	A	-233	A	-211	A	-201	A	-32	A	114	A	-116	A

B. Exploratory Visualization

Distribution of cancer types

In the training data, 27 cases are ALL cancer, and 11 cases are AML cancer. The ratio is about 2.5 to 1. In the test data, 20 cases are ALL cancer, and 14 cases are AML cancer. The ratio is about 1.5 to 1. It indicates that the dataset is not highly imbalance. However, it may be attributed to the small number of patients.



Distribution of sample before and after standardizing

In order to compare two different standardization methods, *StandardScaler* and *MinMaxScaler,* in the scikit-learn preprocessing library, the histogram and density plots are used for visual comparison here. The histogram shows the frequency distribution of the dataset. The density shows the distribution of individual features.

Due to the high dimensional dataset, a small portion of the gene data are randomly selected as sample to explore visualization.

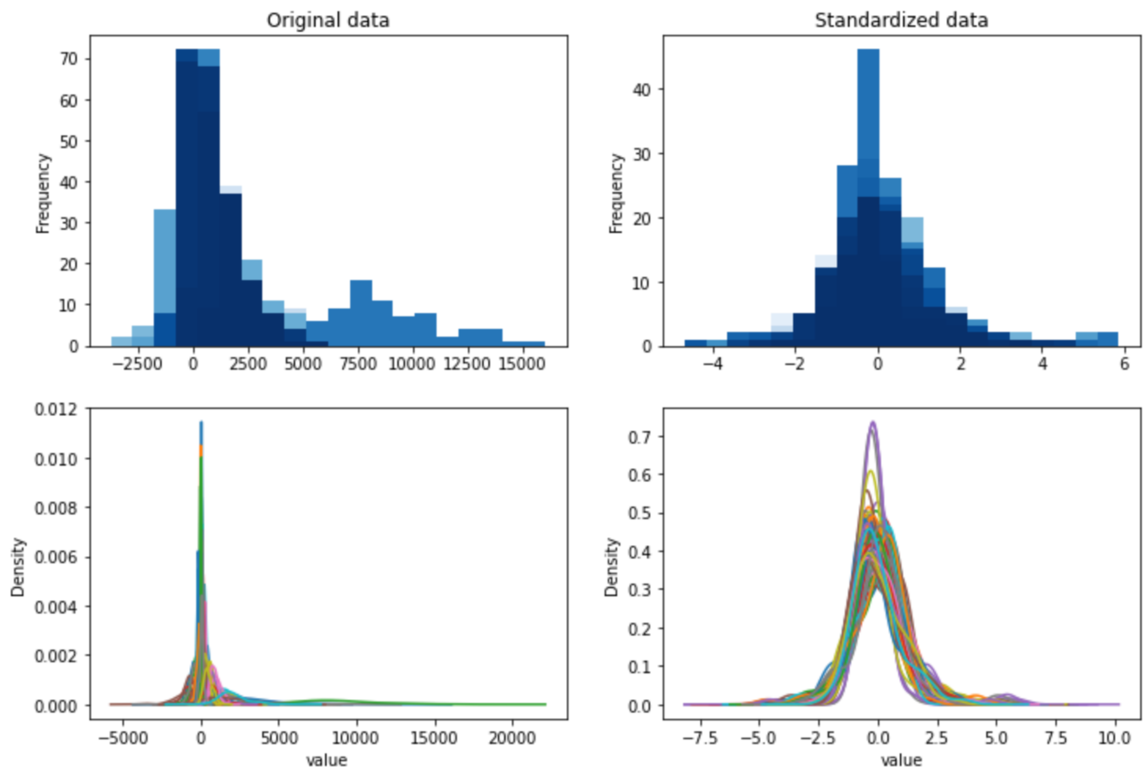
StandardScaler standardize features by removing the mean and scaling to unit variance. The standard score of a sample x is calculated as: z = (x - u) / s, where u is the mean of the training samples, and s is the standard deviation of the training samples.

MinMaxScaler transform features by scaling each feature to a given range. The default range is between zero and one.

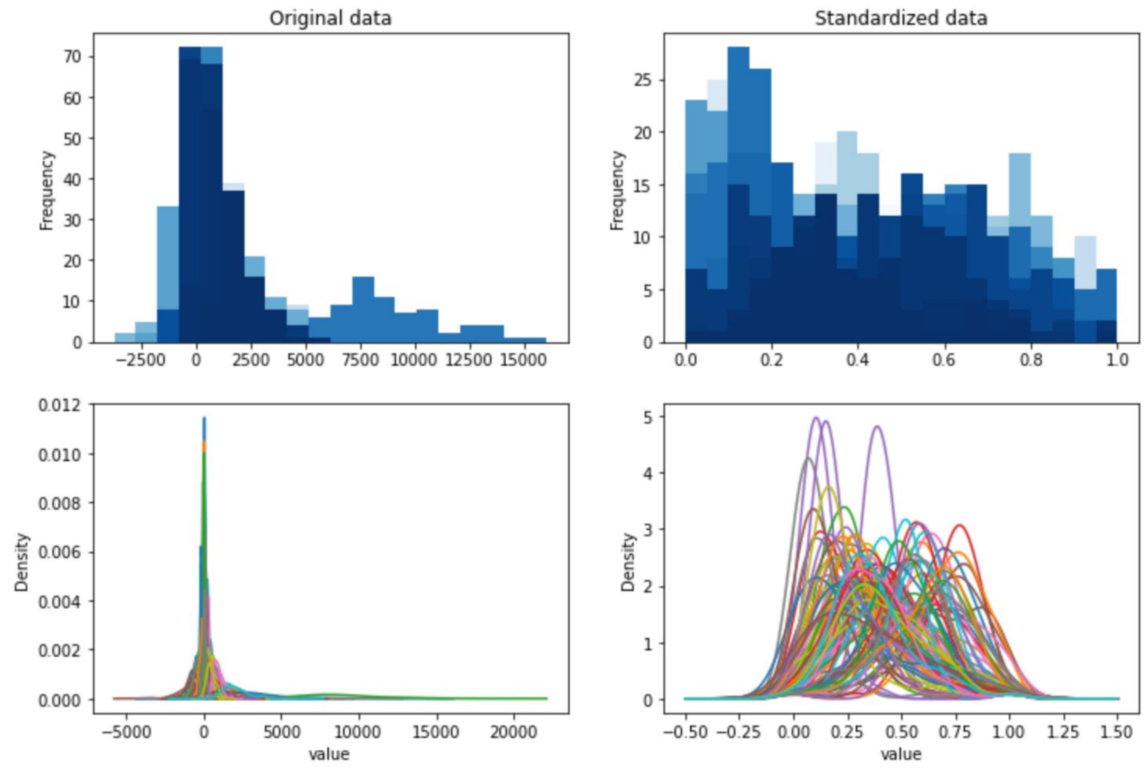
The figures below show the difference between *StandardScaler* and *MinMaxScaler* before and after standardization. The standardized sample obtained by *StandardScaler* method have a concentrated bell-shape distribution on frequency and density plot. However, the standardized sample obtained by *MinMaxScaler* method are spread on both frequency and density plots.

Therefore, the *StandardScaler* method is used to standardize all dataset including training and test data.

Distribution of Random Sample - StandardScaler



Distribution of Random Sample - MinMaxScaler



C. Algorithms and Techniques

This project focused on supervised learning, which is a machine learning modeling that can learn pattens from a large number of input and output data point pairs. Supervised learning algorithms analyze the training data and generate an inference function that can predict the output of unseen examples.

The most widely used learning algorithms are linear regression, logistic regression, naive Bayes, Support Vector Machines, and k-nearest neighbor algorithm. Since I was not very familiar with these algorithms, I read the user guide of scikit-learn and tried to implement them by following its examples.

K-Nearest Neighbor algorithm: `KNeighborsClassifier` method in the scikit-learn neighbors module was used to implement learning based on the nearest neighbors of each query point. This algorithm is the most common used technique in classification.

Naive Bayes: `GaussianNB` method in the scikit-learn naive_bayes module was used to implement the Gaussian Naive Bayes algorithm for classification with assumption of conditional independence between every pair of features given the value of the class variable. The likelihood of the features is assumed to be Gaussian.

Logistic Regression: `LogisticRegression` method in the scikit-learn linear_model module was used to implement regularized logistic regression using the liblinear library. The liblinear solver supports both L1 and L2 regularization.

Support Vector Machines: `SVC` method in the scikit-learn support vector machines module was used to perform C-Support Vector Classification based on libsvm.

Gradient Boosting algorithm: `GradientBoostingClassifier` method in the scikit-learn ensemble module was used to build an additive model in a forward stage-wise fashion. `n_estimators` and `learning_rate` are two most important parameters of this estimators. There is a trade-off between `n_estimators` and `learning_rate`. According to the user guide, large number of `n_estimators` usually results in better performance and `learning_rate` controls overfitting via shrinkage.

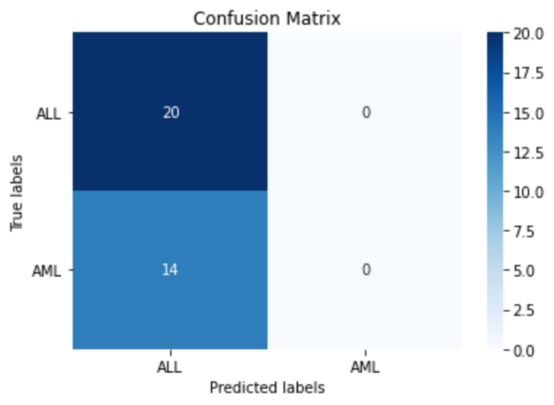
K-Means Clustering: `KMeans` method in the scikit-learn cluster module was used here to setup a benchmark model from an unsupervised learning algorithm. The parameter `n_clusters` is set to 2 to divide the input dataset into two disjoint groups. This algorithm relies on finding clusters in n-dimensional feature space, so it has difficulty distinguishing noise from important features in higher dimensions. Thus, a dimensionality reduction algorithm Principal component analysis (PCA) is running prior to k-means clustering to counter this problem and speed up the computations.

D. Benchmark

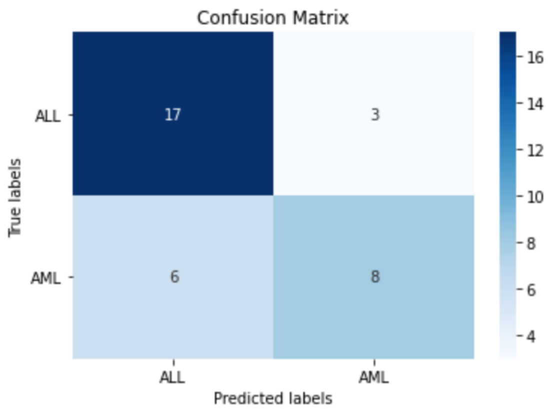
The dataset has labeled the cancer type of each patient. In the test data, there are 20 ALL patients and 14 AML patients. If the binary classification predicts all patients have ALL type cancer, the AUC score is 0.5 and the accuracy score is 0.59. This is a simple benchmark for modeling.

In addition, an unsupervised clustering approach K-Means algorithm is tried as a benchmark model, because supervised learning models should have better performance than unsupervised learning models. For the K-means clustering model, the number of clusters is set to 2, and the prediction has the AUC score 0.71 and the accuracy score 0.74.

```
Baseline Model Performance -
AUC: 0.5
Accuracy: 0.59
```



```
K-Means Clustering Model Performance -
AUC: 0.71
Accuracy: 0.74
```



III. Methodology

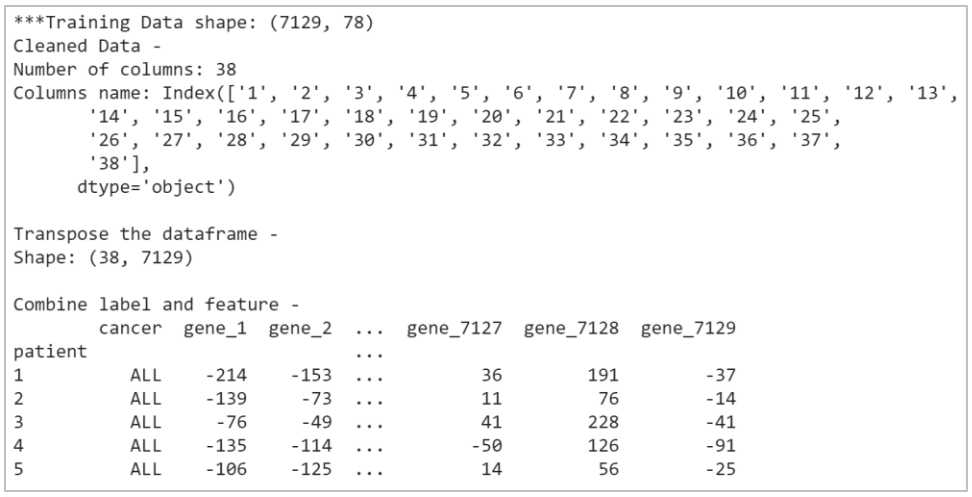
A. Data Preprocessing

Data Cleaning:

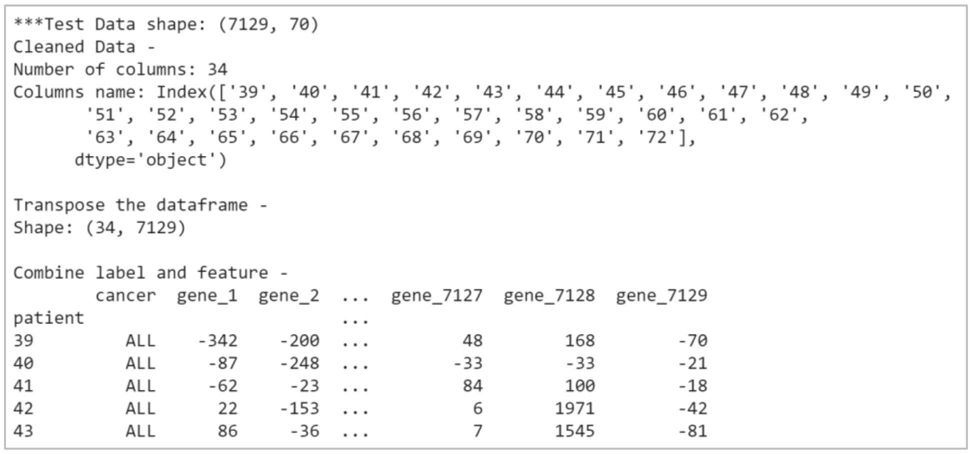
```python
def cleanData(data, patient_data, dataset):
  # Remove columns that contain 'call'
  col_drop = [col for col in data.columns if 'call' in col or 'Gene' in col]
  data_clean = data.drop(col_drop, axis=1, inplace=False)

  print('Cleaned Data - ')
  print('Number of columns:', len(data_clean.columns))
  print('Columns name:', data_clean.columns)

  # Transpose index and columns
  print('\nTranspose the dataframe - ')
  data_df = data_clean.T
  print('Shape:', data_df.shape)

  # Rename columns
  genes_total = len(data['Gene Accession Number'].unique())
  data_df.columns = ['gene_' + str(i) for i in range(1, genes_total+1)]

  # Rename index and combine label and features
  print('\nCombine label and feature - ')
  patient_data.index = patient_data['patient']
  if dataset == 'train':
    data_df.index = patient_data['patient'][:38]
    data_df = pd.concat([patient_data['cancer'][:38], data_df], axis=1)
  else:
    data_df.index = patient_data['patient'][38:]
    data_df = pd.concat([patient_data['cancer'][38:], data_df], axis=1)

  return data_df
```

The `cleanData` function above performs data cleaning steps as following:

1. Delete all "Call" columns in the training data and test data:
   Since the distribution of three type are similar for each gene. It is not helpful in model training.
2. Remove "Gene description" columns
3. Transpose the training and test data frame:
   Each row is a patient case data point, and each column is a gene expression value.
4. Rename the "Gene Accession Number" column to simplify the gene number name.
5. Combine label and features:
   Label is cancer type, ALL or AML. Features are gene expression values.



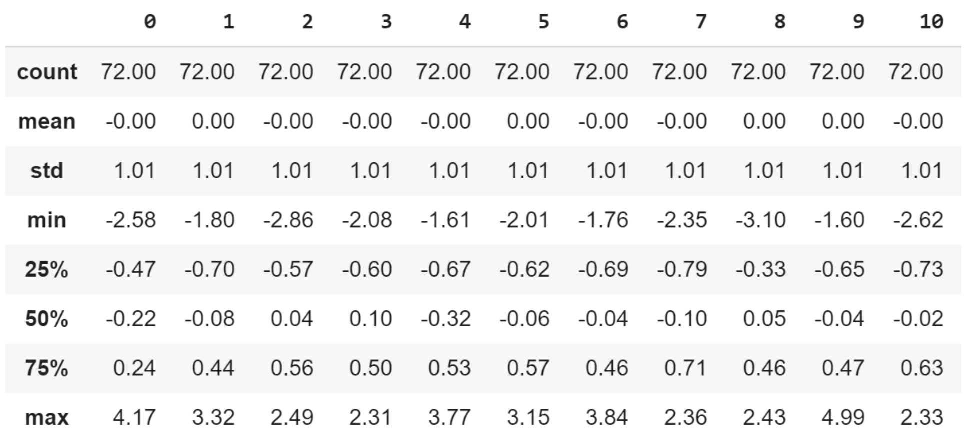
```
***Training Data shape: (7129, 78)
Cleaned Data -
Number of columns: 38
Columns name: Index(['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13',
       '14', '15', '16', '17', '18', '19', '20', '21', '22', '23', '24', '25',
       '26', '27', '28', '29', '30', '31', '32', '33', '34', '35', '36', '37',
       '38'],
      dtype='object')

Transpose the dataframe -
Shape: (38, 7129)

Combine label and feature -
        cancer  gene_1  gene_2  ...  gene_7127  gene_7128  gene_7129
patient                         ...
1          ALL    -214    -153  ...         36        191        -37
2          ALL    -139     -73  ...         11         76        -14
3          ALL     -76     -49  ...         41        228        -41
4          ALL    -135    -114  ...        -50        126        -91
5          ALL    -106    -125  ...         14         56        -25
```



```
***Test Data shape: (7129, 70)
Cleaned Data -
Number of columns: 34
Columns name: Index(['39', '40', '41', '42', '43', '44', '45', '46', '47', '48', '49', '50',
       '51', '52', '53', '54', '55', '56', '57', '58', '59', '60', '61', '62',
       '63', '64', '65', '66', '67', '68', '69', '70', '71', '72'],
      dtype='object')

Transpose the dataframe -
Shape: (34, 7129)

Combine label and feature -
        cancer  gene_1  gene_2  ...  gene_7127  gene_7128  gene_7129
patient                         ...
39         ALL    -342    -200  ...         48        168        -70
40         ALL     -87    -248  ...        -33        -33        -21
41         ALL     -62     -23  ...         84        100        -18
42         ALL      22    -153  ...          6       1971        -42
43         ALL      86     -36  ...          7       1545        -81
```

Standardization

As mentioned in the comparison in the previous section, the `StandardScaler` method in the scikit-learn library was used to standardize numerical features by removing the mean and scaling to unit variance. The description of the scaling data is shown below.



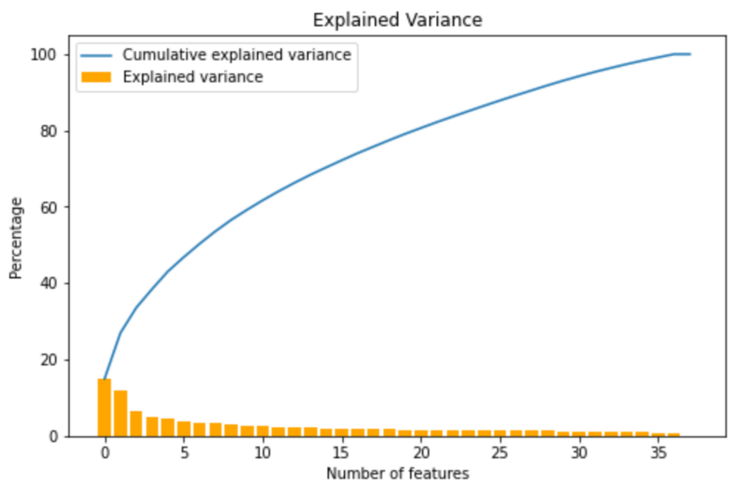
	0	1	2	3	4	5	6	7	8	9	10
count	72.00	72.00	72.00	72.00	72.00	72.00	72.00	72.00	72.00	72.00	72.00
mean	-0.00	0.00	-0.00	-0.00	-0.00	0.00	-0.00	-0.00	0.00	0.00	-0.00
std	1.01	1.01	1.01	1.01	1.01	1.01	1.01	1.01	1.01	1.01	1.01
min	-2.58	-1.80	-2.86	-2.08	-1.61	-2.01	-1.76	-2.35	-3.10	-1.60	-2.62
25%	-0.47	-0.70	-0.57	-0.60	-0.67	-0.62	-0.69	-0.79	-0.33	-0.65	-0.73
50%	-0.22	-0.08	0.04	0.10	-0.32	-0.06	-0.04	-0.10	0.05	-0.04	-0.02
75%	0.24	0.44	0.56	0.50	0.53	0.57	0.46	0.71	0.46	0.47	0.63
max	4.17	3.32	2.49	2.31	3.77	3.15	3.84	2.36	2.43	4.99	2.33

Dimensionality Reduction

The principal component analysis (PCA) algorithm in the scikit-learn library was used in the project to reduce the dimensionality of the input data set.

In the first run, the number of components parameter (`n_components`) was kept as the default value, which is the minimum of the number of samples and the number of features. Then, in order to determine how many principal components constitute at least 90% of the data variance, the `explained_variance_ratio_` attribute (percentage of variance explained by each of the selected components) was used to calculated the cumulative explained variance and draw the following figure.

```
28 principal components captures at least 90% of the total variance in the dataset.
```

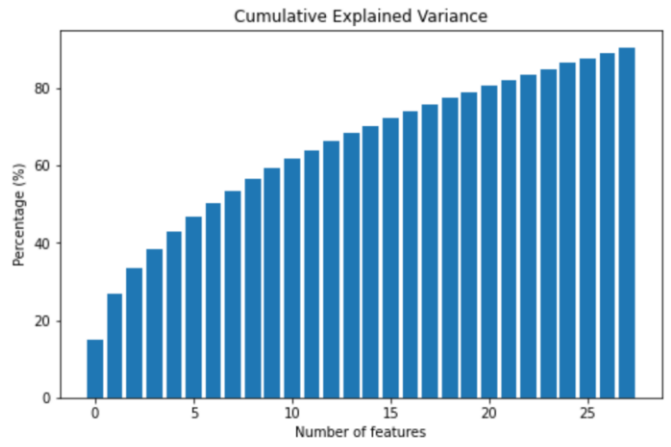


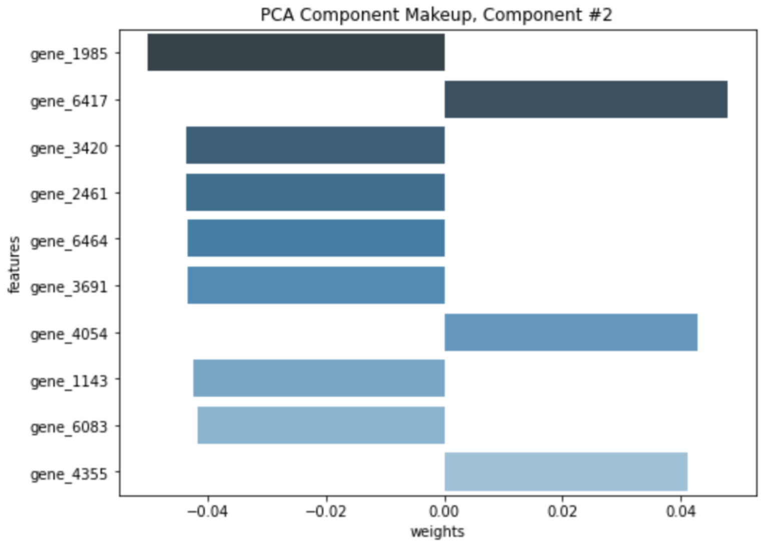
In the second run, the `n_components` parameter was set to 28 to capture 90% of the data variance, and the transformed training and test data sets were saved for model training. Looking at the percentage of variance explained by each selected component, the first three principal components account for approximately 34% of the total variance.

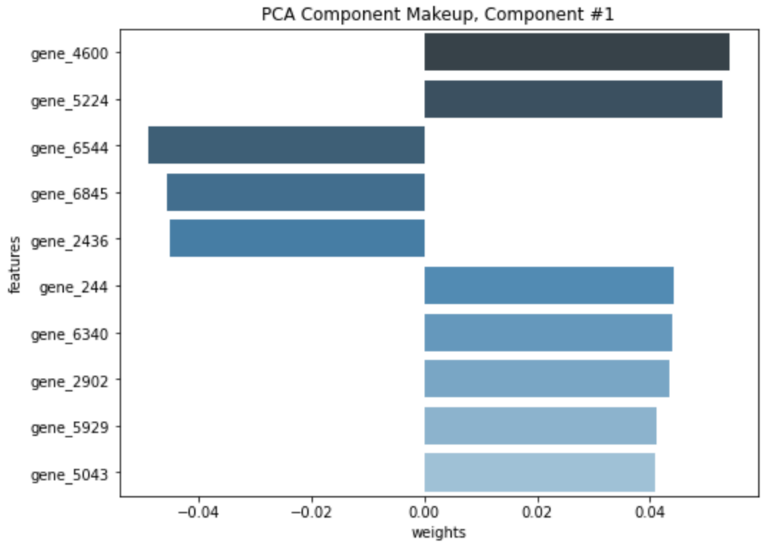
Component Makeup and 3D visualization

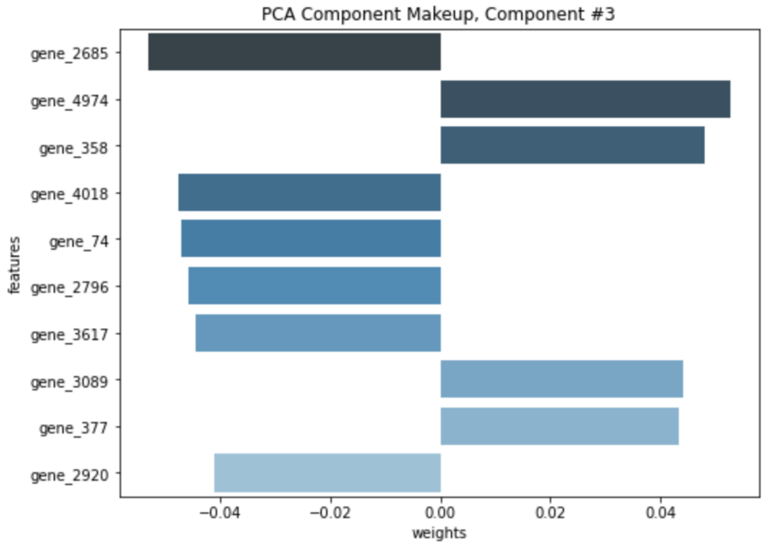
The `display_component` function matches the weightings of the original features contained in the components and displays the composition of each PCA component. The first three components account for 15%, 12%, and 6.6%of the total variance respectively, and the top 10 of their contained gene expression features are shown below. Since there are a total of 7129 genes in the dataset and the weightings difference is not significant, it is difficult to tell any story from the makeup of each component.

```
Percentage of variance explained by each of the selected components:
[14.99 11.98  6.6   4.88  4.63  3.72  3.49  3.29  2.99  2.65  2.51  2.36
  2.21  2.09  1.94  1.89  1.85  1.71  1.71  1.64  1.54  1.53  1.46  1.43
  1.42  1.38  1.35  1.31]
```

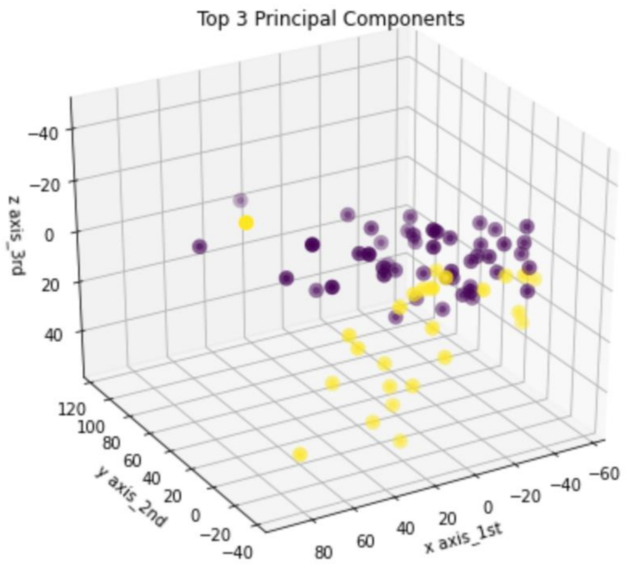








In order to explore the distribution of the dataset, the same PCA algorithm was used to reduce the entire gene expression dataset including training and test data into three principle components and the transformed data was plotted on three-dimensional coordinates. In the figure below, it is clear to tell that there are two clusters. Based on this finding, I believe that binary classification should be able to distinguish cancer types from the gene expression data.



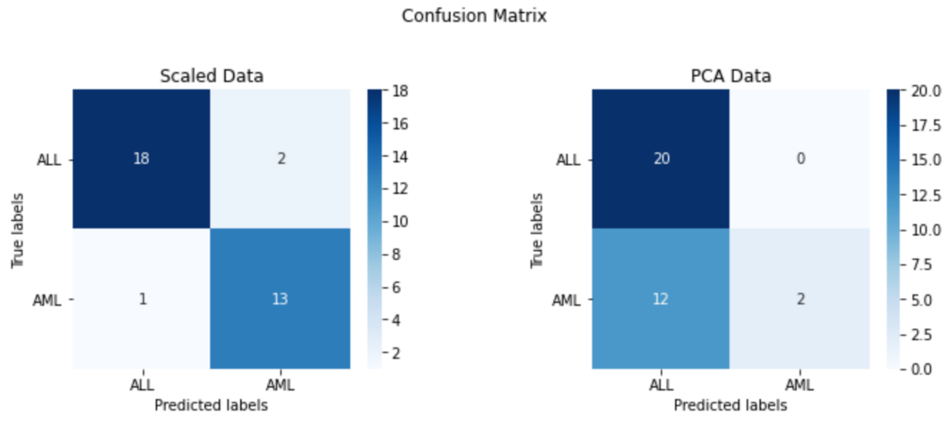
B. Implementation

In order to compare and find a better supervised learning algorithm, the algorithm modules in the scikit-learn library were used and their default settings were tried in the initial run to define and create each estimator.

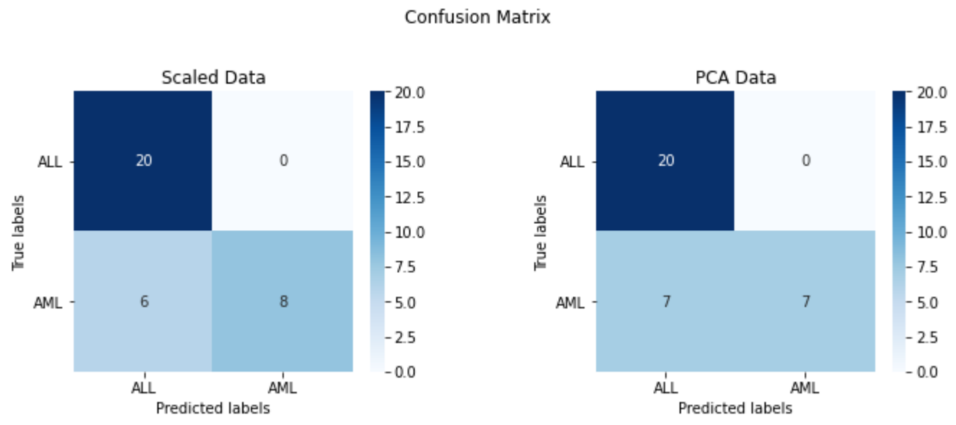
In the first attempt, I found that the models trained by the PCA transformed data (28 principle components) did not perform well in classifying cancer types. The AUC score ranges from 0.57 to 0.75, and the accuracy score ranges from 0.65 to 0.79. The logistic regression model has the best performance among all models. The AUC score is 0.75 and the accuracy score is 0.79.

Then, I tried to train the estimators using the standardized data with all gene expression features. Surprisingly, it generated better models for most algorithms except the SVC model. The Naive Bayes model and the Gradient Boosting model performed best on the AUC score of 0.91 and the accuracy score of 0.91.

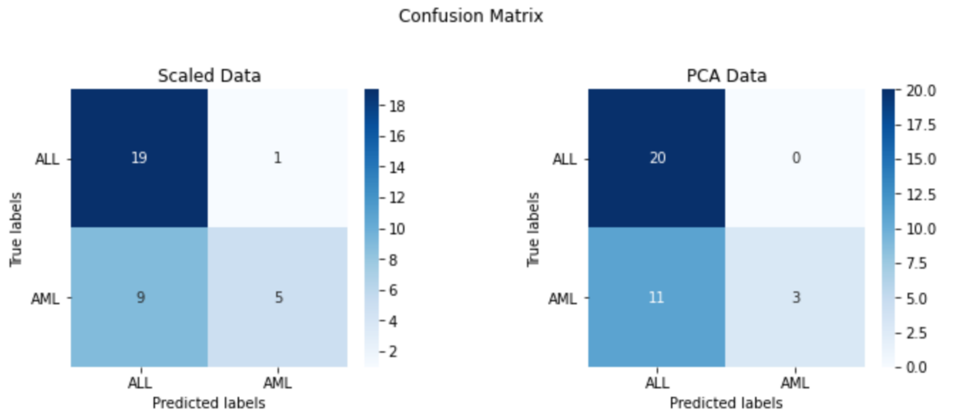
```
Gaussian Naive Bayes Classifier Model Performance -

Scaled Data -
AUC: 0.91
Accuracy: 0.91

PCA Data -
AUC: 0.57
Accuracy: 0.65
```



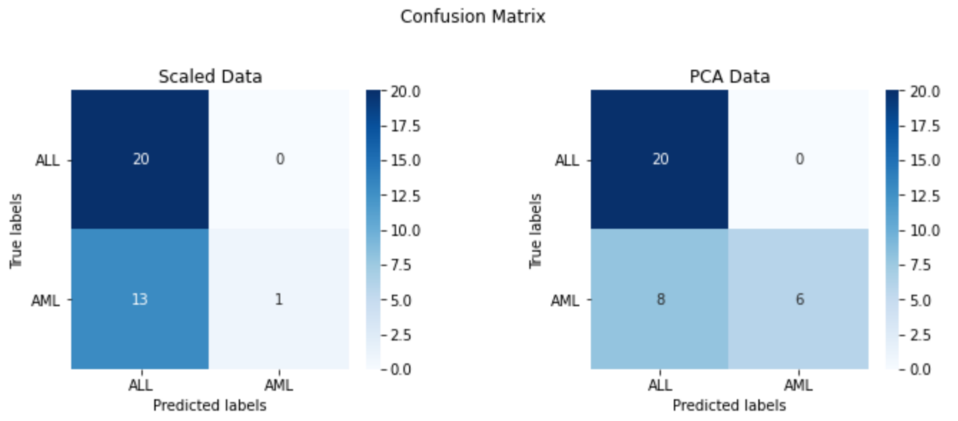
```
Logistic Regression Classifier Model Performance -

Scaled Data -
AUC: 0.79
Accuracy: 0.82

PCA Data -
AUC: 0.75
Accuracy: 0.79
```



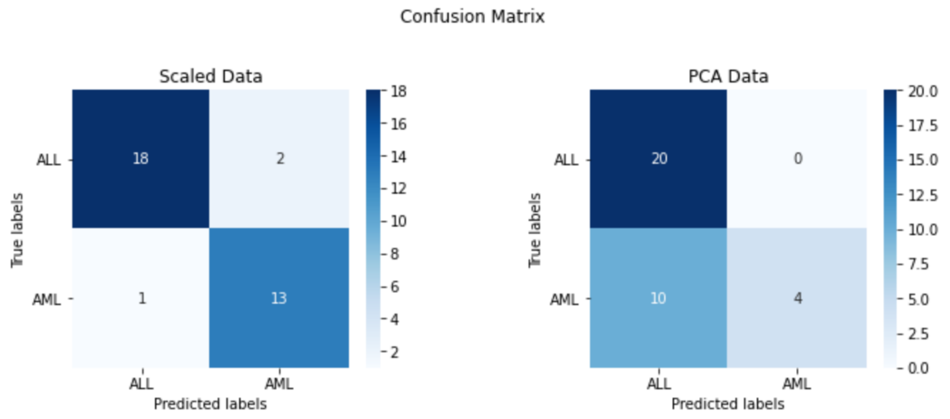
```
K-Nearest Neighbors Classifier Model Performance -

Scaled Data -
AUC: 0.65
Accuracy: 0.71

PCA Data -
AUC: 0.61
Accuracy: 0.68
```



```
C-Support Vector Classifier Model Performance -

Scaled Data -
AUC: 0.54
Accuracy: 0.62

PCA Data -
AUC: 0.71
Accuracy: 0.76
```



```
Gradient Boosting Classifier Model Performance -

Scaled Data -
AUC: 0.91
Accuracy: 0.91

PCA Data -
AUC: 0.64
Accuracy: 0.71
```



C. Refinement

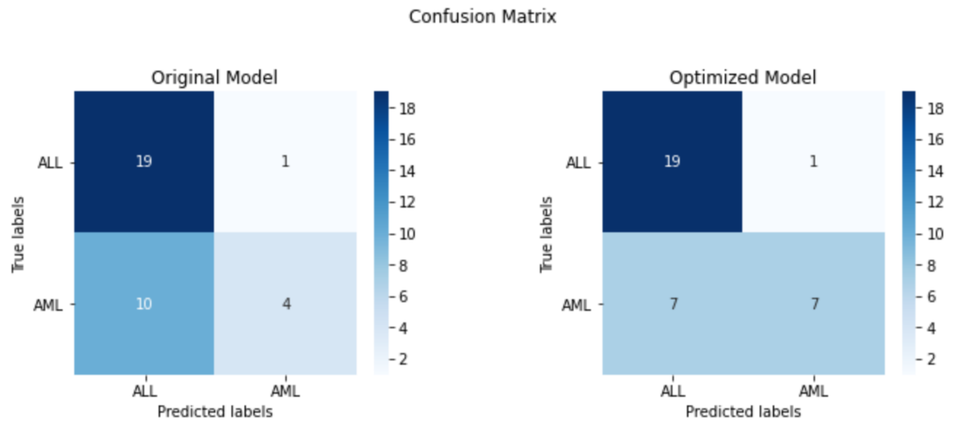
In order to improve the performance of these models by tuning the hyperparameters. The `GridSearchCV` method in the scikit-learn model_selection module was used to search over different parameter values of the estimator and find the best combination of parameters. This method optimizes the parameters of the estimator by grid-search of cross-validated over a parameter grid and selects the parameters on the basis of maximizing the score.

Since the standardized dataset with all features had better performance on model training in the previous section, it was used to tune the hyperparameters of these models. After adjusting the hyperparameters, the performance of the K-Nearest Neighbors model, Logistic Regression model, and Support Vector Machine model was improved by at least 10%.

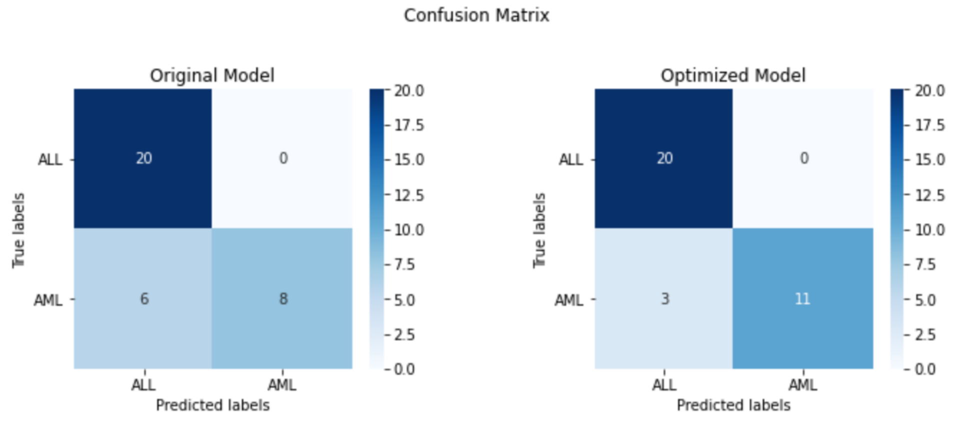
The Support Vector Machine model has the greatest improvement with the AUC score increased from 0.71 to 0.89, and the accuracy score increased from 0.76 to 0.91. It shows that a good combination of parameters plays an important role in training a good model.

The default setting of Gradient Boosting algorithm with the standardized dataset already has 91% accuracy, so tuning the hyperparameters did not make any improvement in the performance of this model.

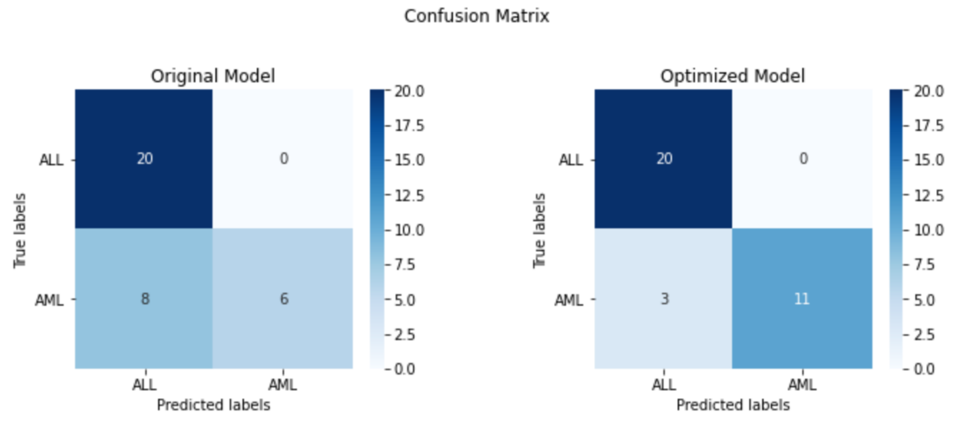
```
K-Nearest Neighbors Classifier Model Performance -

Original Model -
AUC: 0.62
Accuracy: 0.68

Optimized Model -
AUC: 0.72
Accuracy: 0.76
```



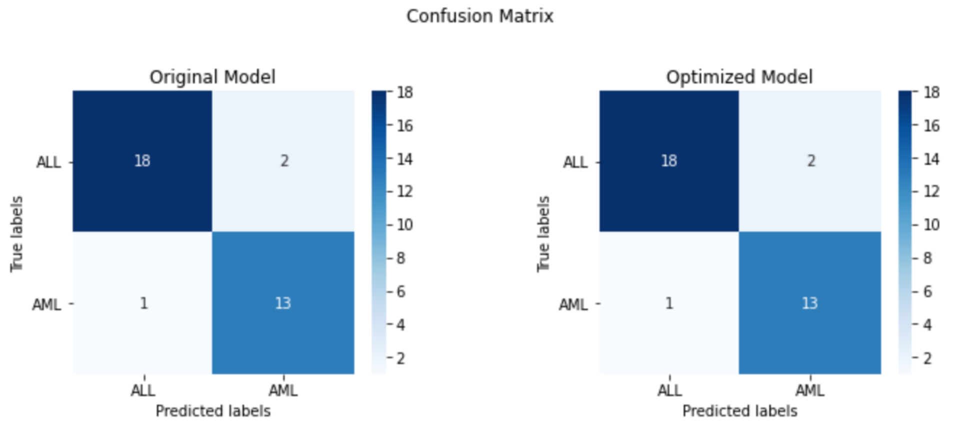
```
Logistic Regression Classifier Model Performance -

Original Model -
AUC: 0.79
Accuracy: 0.82

Optimized Model -
AUC: 0.89
Accuracy: 0.91
```



```
C-Support Vector Classifier Model Performance -

Original Model -
AUC: 0.71
Accuracy: 0.76

Optimized Model -
AUC: 0.89
Accuracy: 0.91
```



```
Gradient Boosting Classifier (Python Package) Model Performance -

Original Model -
AUC: 0.91
Accuracy: 0.91

Optimized Model -
AUC: 0.91
Accuracy: 0.91
```



IV. Results

Having created estimators based on different algorithms, trained by scaled training data or PCA transformed training data, and turned their hyperparameters, the Naive Bayes model and the Gradient Boosting model showed best performance in classifying the type of cancer among all the trained models.

The estimators were trained by the scaled training data with 7129 features and the default parameter settings of `GaussianNB` method and `GradientBoostingClassifier` method in the scikit-learn libraries as shown below.

Default Parameters of Naive Bayes model:

- `priors`=None
- `var_smoothing`=1e-09

Default Parameters of Gradient Boosting model:

- `loss=` "deviance": refers to deviance (logistic regression) for classification with probabilistic outputs.
- `learning_rate`=0.1: shrinks the contribution of each tree by learning_rate.
- n_estimatorsint=100: the number of boosting stages to perform.
- `max_depthint`=3: maximum depth of the individual regression estimators

The AUC scores of both are 0.91 and the accuracy rates of both are 91%. They predicted 18 corrected ALL cancer cases and 13 corrected AML cancer cases out of total 34 cases.

Compared with the baseline model with 0.5 AUC score and 59% accuracy rate and the K-Means model with 0.71 AUC score and 74% accuracy rate, the Naive Bayes model or the Gradient Boosting model are better classifiers in the cancer type prediction base on the patient's 7129 gene expression.

V. Conclusion

This dataset provided by Kaggle is a small dataset with high dimensionality. Only 38 training data points might not be enough to train an effective model. I have reviewed other notebooks on Kaggle and their results have similar with 91%-92% accuracy rate on prediction the cancer type.

However, trying the different algorithms in this project gave me the opportunity to study these supervised learning algorithms. It is a good start for my first project.