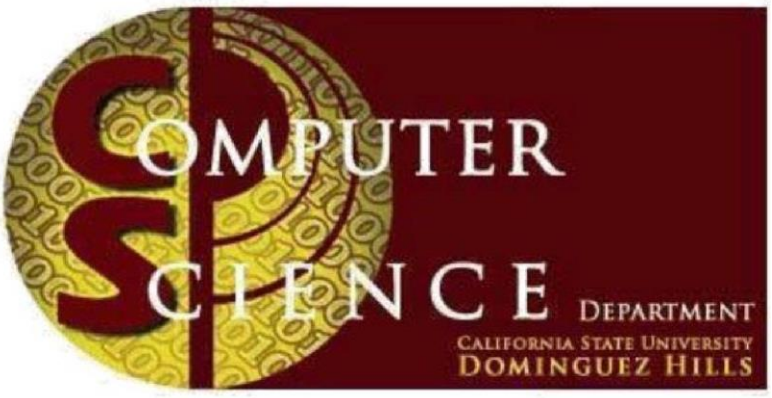


Fall 2023

Master Project

Computer Science Department

California State University, Dominguez Hills

TCGA CANCER GENOMICS PROJECT

A Project
Presented
to the Faculty of

_____

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

_____

by
Sai Keertana Padmanabham
Fall 2023

PROJECT: TCGA CANCER GENOMICS PROJECT

AUTHOR: SAI KEERTANA PADMANABHAM

APPROVED:

_____

Dr. Jianchao Han, Ph.D
Project Committee Chair

_____

Dr. Mohsen Beheshti, Ph.D
Committee Member

_____

Dr. Ryan Urbanowicz, Ph.D
Committee Member

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my esteemed committee for all their help and advice with my master's project.

I would first and foremost like to thank my project committee chair, Dr. Jianchao Han, Ph.D., from the bottom of my heart. His insightful feedback, constant backing, and significant machine learning expertise have greatly influenced the direction of my research. I was able to successfully complete my project because of Dr. Han's knowledgeable advice, and his helpful critique was helpful in improving my work.

I am also extremely grateful to Dr. Xiuzhen Huang, Ph.D., for providing me with the chance to collaborate on medical projects with Cedars Sinai Company. I am thankful of Dr. Ryan Urbanowicz, Ph.D., whose knowledge and insightful advice enhanced my understanding and methodology for this study effort.

I also thank Dr. Mohsen Beheshti, Ph.D., a debt of gratitude for his invaluable support during my academic career.

I also want to express my gratitude to all the staff and professors who have helped me succeed academically and have had a positive impact on my learning process. Their commitment and assistance have created a caring and engaging learning environment.

TABLE OF CONTENT

ACKNOWLEDGEMENT iv

TABLE OF CONTENTS v

LIST OF FIGURES v

ABSTRACT viii

CHAPTER 1: INTRODUCTION 1

CHAPTER 2: RELATED WORK 2

CHAPTER 3: SYSTEM ARCHITECTURE 5

3.1: MODULES 5

CHAPTER 4: SYSTEM IMPLEMENTATION. 13

4.1: SOFTWARE REQUIREMENTS. 13

4.2: LIBRARIES USED. 13

4.3: METHODOLOGIES. 14

CHAPTER 5: EXPERIMENTS RESULTS AND ANALYSIS. 18

CHAPTER 6: CONCLUSION AND FUTURE WORK 36

REFERENCES 38

APPENDIX 41

LIST OF FIGURES

FIG.3.1: SYSTEM ARCHITECTURE. 5

FIG.3.2: LABELS DATASET… 6

FIG.3.3: CANCER GENOMICS DATASET… 7

FIG.5.1: LOADING DATA DATASET 20

FIG.5.2: LOADING LABELS DATASET. 20

FIG.5.3: DATA TYPES OF DATA SET. 21

FIG.5.4: DATA TYPES OF LABEL SET. 21

FIG.5.5: REMOVING THE UNWANTED COLUMNS FROM DATASET. 21

FIG.5.6: MISSING VALUES IN THE DATASET. 22

FIG.5.7: DESCRIPTION OF THE DATASETS. 22

FIG.5.8: CORRELATION MATRIX FOR GENE EXPRESSION DATA. 23

FIG.5.9: HEATMAP OF LABELS DATASET. 24

FIG.5.10: CORRELATION MATRIX HEATMAP 24

FIG.5.11: NORMALIZING THE DATA. 25

FIG.5.12: APPLYING PCA TO RETAIN 95% OF VARIANCE. 25

FIG.5.13: PERCENTAGE VARIANCE OF PCA DATA. 26

FIG.5.14: APPLYING PCA FOR 2 COMPONENTS. 26

FIG.5.15.1, 5.15.2: SCATTER PLOT FOR PCA FOR 2 & 3 COMPONENTS. 27

FIG.5.16.1: K-MEANS CLUSTERING. 27

FIG.5.16.2: SILHOUETTE ANALYSIS FOR 5 CLUSTERS. 27

FIG.5.17.1: RF CLASSIFICATION REPORT. 28

FIG.5.17.2: HEATMAP FOR RANDOM FOREST 28

FIG.5.17.3: RF CONFUSION MATRIX. 28

FIG.5.17.4: RF CLASS PREDICTION ERROR… 28

FIG.5.18.1: SVM CLASSIFICATION REPORT. 29

FIG.5.18.2: HEATMAP FOR SVM 29

FIG.5.18.3: SVM CONFUSION MATRIX. 30

FIG.5.18.4: SVM CLASS PREDICTION ERROR… 30

FIG.5.19.1: GB CLASSIFICATION REPORT. 31

FIG.5.19.2: HEATMAP FOR GB 31

FIG.5.19.3: GB CONFUSION MATRIX. 31

FIG.5.19.4: GB CLASS PREDICTION ERROR… 31

FIG.5.20.1: NN CLASSIFICATION REPORT. 32

FIG.5.20.2: HEATMAP FOR NN 32

FIG.5.20.3: NN CONFUSION MATRIX. 32

FIG.5.20.4: NN CLASS PREDICTION ERROR… 32

FIG.5.21.1: XGB CLASSIFICATION REPORT. 33

FIG.5.21.2: HEATMAP FOR XGB 33

FIG.5.21.3: XGB CONFUSION MATRIX. 34

FIG.5.21.4: XGB CLASS PREDICTION ERROR… 34

FIG.5.22: COMPARISON OF SUPERVISED LEARNING CLASSIFIERS. 35

ABSTRACT

The goal of this research is to address important issues in cancer genomics by utilizing the large gene expression datasets from the UCI Machine Learning Repository and the TCGA Pan Cancer analysis project. The study makes use of advanced methods of machine learning to achieve two primary objectives: first, it uses unsupervised learning to explore intrinsic patterns within the data, and second, it uses supervised learning algorithms to classify different cancer types (BRCA, KIRC, COAD, LUAD, and PRAD). Using feature selection approaches to identify the most important genes that can accurately distinguish between these cancer types is an essential part of the project. Learning from the knowledge acquired from deep genomic research, the project seeks to enhance the understanding of the genomic areas that mold the course of cancer growth and development. This includes characterizing chromosomal rearrangements like chromothripsis and kataegis, which are involved in the evolution of tumors (Nature Genetics, 2013) [1]. The effort seeks to improve the identification of driver genes and chromosomal rearrangements through the utilization of extensive analysis. This will help to progress targeted cancer medicines and improve the accuracy of oncological interventions.

1. INTRODUCTION

The study of cancer genomics is constantly changing, offering up possibilities for research into the complex mechanisms behind the formation, development, and resistance to therapy of cancer. The analysis of gene expression data, which is essential to this research, has the potential to identify the genetic signatures particular to various cancer types. Leading this study is the AI-Campus Project: TCGA Cancer Genomics Project, which makes use of extensive datasets collected by the UCI Machine Learning Repository and contributed by the TCGA Pan Cancer analysis project.

The two main goals of this project are to build strong classifiers for cancer type prediction and to clarify the complex patterns present in cancer genomics data. The project aims to identify the intricate relationships between gene expressions and cancer features using a strategic strategy that integrates supervised and unsupervised learning techniques [1]. The study focuses especially on the gene expression patterns of various common cancer forms, such as lung adenocarcinoma (LUAD), prostate adenocarcinoma (PRAD), colon adenocarcinoma (COAD), kidney renal clear cell carcinoma (KIRC), and breast cancer (BRCA).

As it continues this research project, the study also emphasizes feature selection to find important genes that function as reliable diagnostics for cancer classification [3]. Personalized therapy, in which therapies are customized based on the genetic composition of each patient's cancer, is made possible by this work in addition to improving our basic understanding of the genomic pattern of tumors. The AI-Campus Project aims to make a major contribution to the field of oncology by using the insights it has obtained to provide researchers and doctors with useful tools in the ongoing fight against cancer.

2. RELATED WORK

The scientific understanding of cancer has been completely transformed by the significant research known as the Cancer Genome Atlas (TCGA) Pan-Cancer analysis project [5]. More than 10,000 cancers have had their genomes profiled and analyzed because of the project [3][4]. This large dataset has provided an immense quantity of knowledge regarding the changes in genes that stimulate the growth of cancer.

The TCGA has performed Pan-Cancer analyses, which compare the genomes of several cancer types, in addition to investigating unique cancer types [5]. These investigations have revealed both common and distinct genomic changes among various cancer types. The development of more individualized cancer treatment strategies and the detection of alternative targets for cancer therapy are both assisted by this knowledge [4][5].

Several large-scale cancer genomes projects have been started in recent years, including the TCGA Pan-Cancer analysis project [2][3]. Numerous data sets produced by these studies are helping to understand the molecular causes of cancer and are guiding the creation of innovative cancer treatments [1][2].

Literature Survey:

The TCGA Pan-Cancer analysis project has been the subject of a few important papers. These publications have highlighted the following major findings:

- **Identification of common and unique genetic alterations across different cancer types:** To help identify new targets for cancer therapy and establish individualized treatment plans, the TCGA Pan-Cancer analysis project discovered both common and distinct genomic changes across various cancer types. One example of a potential target

for new cancer therapy is one of the genes that the study has identified as often changed in cancer of the ovary.

- **Development of new cancer subtypes based on genetic alterations:** Based on genomic changes, the TCGA Pan-Cancer analysis project has identified new cancer subtypes, improving the understanding of how cancer develops and providing the way for the creation of stronger treatment plans. For example, a new subtype of breast cancer with certain genetic variants is more responsive to targeted therapies than other subtypes.
- **Discovery of new therapeutic targets:** New medical targets, such as genes involved in the growth of cancer cells, have been discovered by the TCGA Pan-Cancer analysis project. These discoveries may serve as targets for new cancer medications, advancing the development of new cancer treatments.
- **Improved understanding of cancer progression and metastasis:** The development of new cancer preventive and treatment approaches has been facilitated by the TCGA Pan-Cancer analysis project, which has improved our knowledge of cancer metastasis and progression. In order to develop novel medications that stop the spread of cancer cells, it has identified genes linked to the metastasis of cancer.

Proposed System:

This proposed system, which prioritizes high prediction accuracy, provides a thorough method for classifying cancer types utilizing cutting-edge machine learning algorithms. The system builds a prediction model for cancer comprehension and diagnosis using a large training dataset from The Cancer Genome Atlas (TCGA) repository. The application of Python derives from its large libraries and packages that facilitate machine learning methods. Four different algorithms are used:

Random Forest (RF), k-Nearest Neighbors (KNN), Decision Trees (DT), and Support Vector Machines (SVM).

The process includes preprocessing the gene expression data using Principal Component Analysis (PCA) to reduce dimensionality, handle missing values, and ensure quality and compliance with the machine learning models. Machine learning models are subsequently trained using preprocessed data. Metrics including accuracy, precision, recall, and F1-score are used to assess each algorithm's performance. The best appropriate model for the application will be determined by evaluating its maximum accuracy and reliability in classifying different types of cancer.

Oncologists and researchers will find the final model to be a useful tool since it will allow them to more precisely predict cancer types based on genomic data. This method advances both comprehending the complexity of cancer genomes and specific therapy.

3. SYSTEM ARCHITECTURE

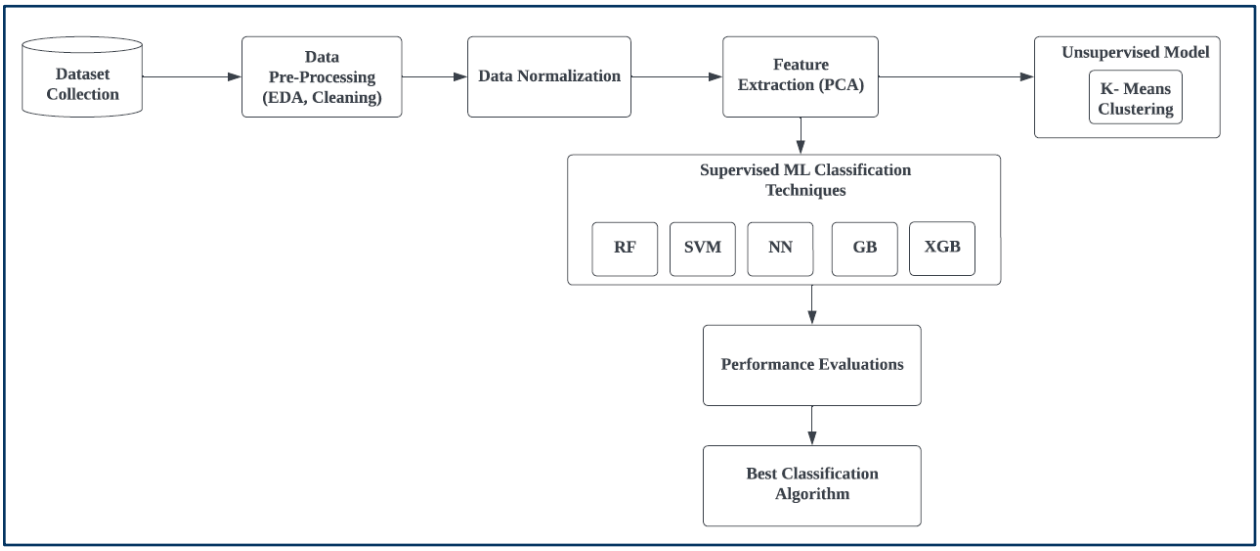


Fig 3.1 System Architecture

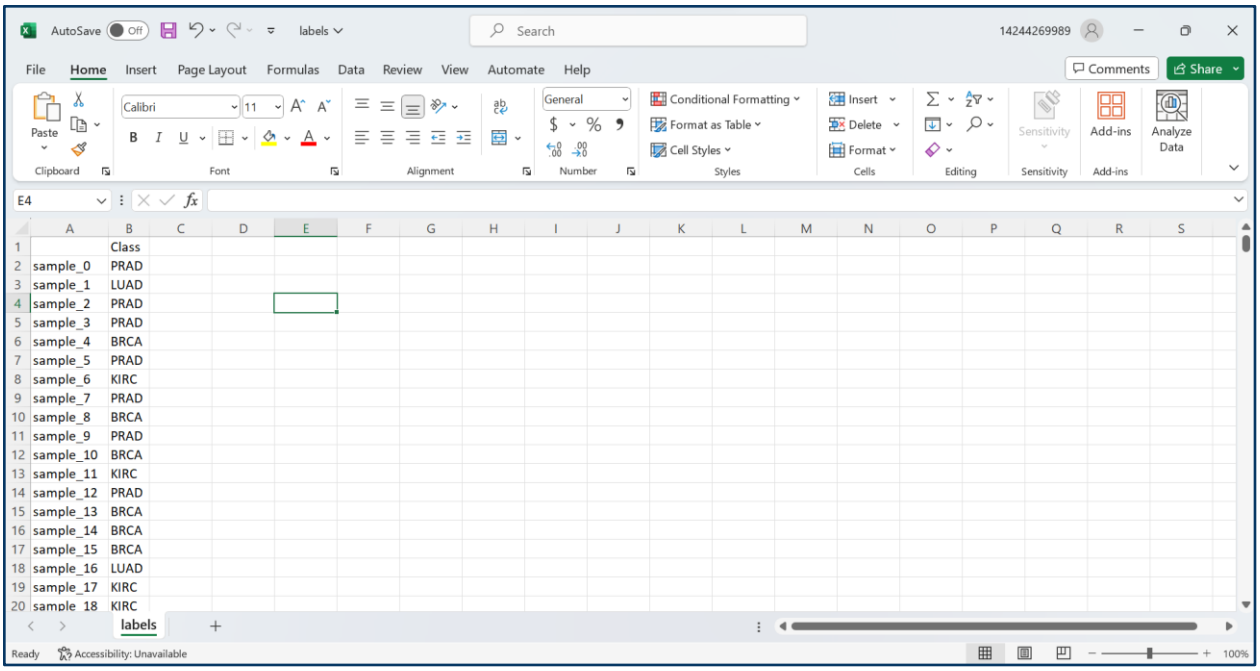
Figure 3.1 shows the proposed system machine learning pipeline, which includes the following steps: collecting datasets, pre-processing data, extracting features, and using various classification methods, such as Random Forest, Support Vector Machine, Neural Networks, and Decision Trees. To determine which classification method is most appropriate for the project, the performance of each algorithm is evaluated using common specifications. This systematic methodology optimizes the classification process for accuracy and dependability by ensuring a data-driven path from early data collection through complete algorithm selection.

3.1 MODULES

3.1.1. Dataset Collection:

This project uses a cancer genomics dataset from The Cancer Genome Atlas (TCGA), accessible through the Genomic Data Commons (GDC) portal. The dataset contains gene expression data from patients diagnosed with various cancer types, including BRCA, KIRC, COAD, LUAD, and

PRAD. The dataset contains approximately 801 records and 20531 attributes, each representing a different gene and its expression level. Rigorous curation ensures the quality and reliability of the data. The goal is to create machine learning models that accurately classify different types of cancer based on gene expression profiles, which is crucial for enhancing molecular understanding and developing personalized cancer diagnosis and treatment.

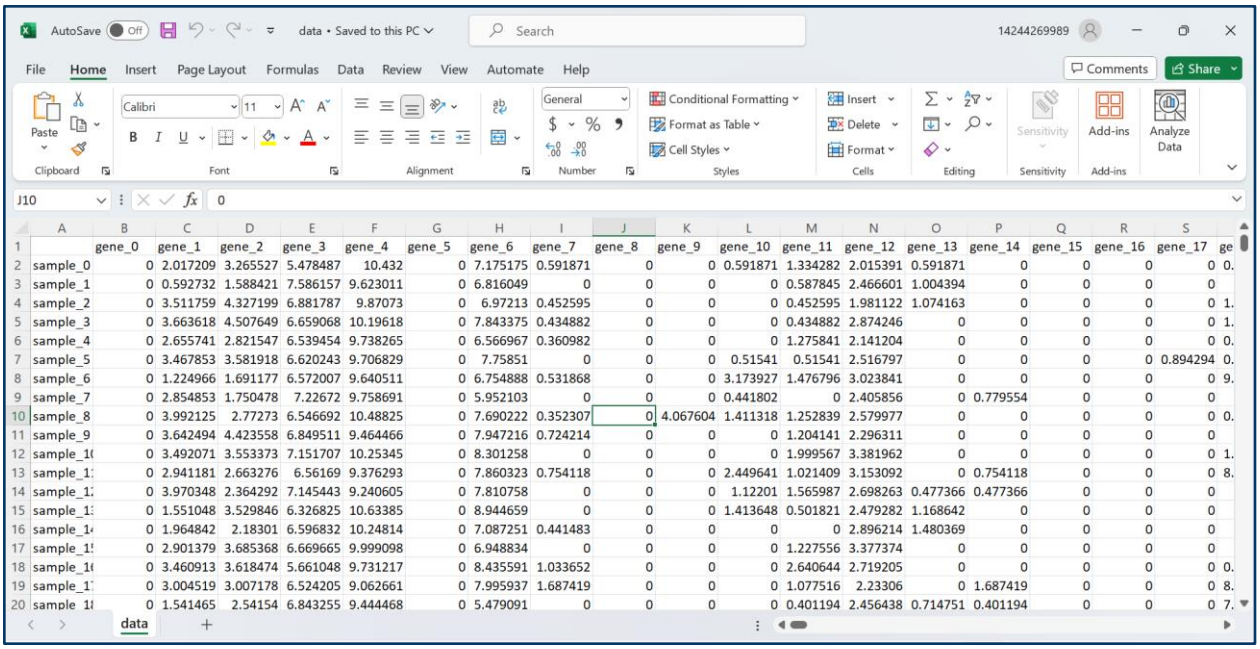


	A	B
1		Class
2	sample_0	PRAD
3	sample_1	LUAD
4	sample_2	PRAD
5	sample_3	PRAD
6	sample_4	BRCA
7	sample_5	PRAD
8	sample_6	KIRC
9	sample_7	PRAD
10	sample_8	BRCA
11	sample_9	PRAD
12	sample_10	BRCA
13	sample_11	KIRC
14	sample_12	PRAD
15	sample_13	BRCA
16	sample_14	BRCA
17	sample_15	BRCA
18	sample_16	LUAD
19	sample_17	KIRC
20	sample_18	KIRC

Fig 3.2 Labels Dataset

Cancer types in the Label dataset:

- BRCA: Breast Invasive Carcinoma
- KIRC: Kidney Renal Clear Cell Carcinoma
- COAD: Colon Adenocarcinoma
- LUAD: Lung Adenocarcinoma
- PRAD: Prostate Adenocarcinoma



	gene_0	gene_1	gene_2	gene_3	gene_4	gene_5	gene_6	gene_7	gene_8	gene_9	gene_10	gene_11	gene_12	gene_13	gene_14	gene_15	gene_16	gene_17	ge
sample_0	0	2.017209	3.265527	5.478487	10.432	0	7.175175	0.591871	0	0	0.591871	1.334282	2.015391	0.591871	0	0	0	0	0.
sample_1	0	0.592732	1.588421	7.586157	9.623011	0	6.816049	0	0	0	0	0.587845	2.466601	1.004394	0	0	0	0	
sample_2	0	3.511759	4.327199	6.881787	9.87073	0	6.97213	0.452595	0	0	0	0.452595	1.981122	1.074163	0	0	0	0	1.
sample_3	0	3.663618	4.507649	6.659068	10.19618	0	7.843375	0.434882	0	0	0	0.434882	2.874246	0	0	0	0	0	1.
sample_4	0	2.655741	2.821547	6.539454	9.738265	0	6.566967	0.360982	0	0	0	1.275841	2.141204	0	0	0	0	0	0.
sample_5	0	3.467853	3.581918	6.620243	9.706829	0	7.75851	0	0	0	0.51541	0.51541	2.516797	0	0	0	0	0.894294	0.
sample_6	0	1.224966	1.691177	6.572007	9.640511	0	6.754888	0.531868	0	0	3.173927	1.476796	3.023841	0	0	0	0	0	9.
sample_7	0	2.854853	1.750478	7.22672	9.758691	0	5.952103	0	0	0	0.441802	0	2.405856	0	0.779554	0	0	0	
sample_8	0	3.992125	2.77273	6.546692	10.48825	0	7.690222	0.352307	0	4.067604	1.411318	1.252839	2.579977	0	0	0	0	0	0.
sample_9	0	3.642494	4.423558	6.849511	9.464466	0	7.947216	0.724214	0	0	0	1.204141	2.296311	0	0	0	0	0	
sample_1	0	3.492071	3.553373	7.151707	10.25345	0	8.301258	0	0	0	0	1.999567	3.381962	0	0	0	0	0	1.
sample_1	0	2.941181	2.663276	6.56169	9.376293	0	7.860323	0.754118	0	0	2.449641	1.021409	3.153092	0	0.754118	0	0	0	8.
sample_1	0	3.970348	2.364292	7.145443	9.240605	0	7.810758	0	0	0	1.12201	1.565987	2.698263	0.477366	0.477366	0	0	0	
sample_1	0	1.551048	3.529846	6.326825	10.63385	0	8.944659	0	0	0	1.413648	0.501821	2.479282	1.168642	0	0	0	0	
sample_1	0	1.964842	2.18301	6.596832	10.24814	0	7.087251	0.441483	0	0	0	0	2.896214	1.480369	0	0	0	0	
sample_1	0	2.901379	3.685368	6.669665	9.999098	0	6.948834	0	0	0	0	1.227556	3.377374	0	0	0	0	0	
sample_1	0	3.460913	3.618474	5.661048	9.731217	0	8.435591	1.033652	0	0	0	2.640644	2.719205	0	0	0	0	0	0.
sample_1	0	3.004519	3.007178	6.524205	9.062661	0	7.995937	1.687419	0	0	0	1.077516	2.23306	0	1.687419	0	0	0	8.
sample_1	0	1.541465	2.54154	6.843255	9.444468	0	5.479091	0	0	0	0	0.401194	2.456438	0.714751	0.401194	0	0	0	7.

Fig 3.3 Cancer Genomics Dataset

3.1.2 Exploratory Data Analysis

The data and labels in the Cancer Genomics dataset are complete, with no missing entries, according to the Exploratory Data Analysis (EDA) that looked for any missing values. To provide descriptive statistics that give a thorough summary of the data, including the quartiles, count, mean, and standard deviation. The balance between various classes is determined by analyzing the class distribution in the labels. After that, created heatmaps to show correlations: the first one shows correlations in the gene expression data of a subset of genes, while the second one looks at correlations in the label data following class encoding.

```
# Check for missing values in the data
missing_values_data = dataset.isnull().sum().sum()
missing_values_labels = labelset.isnull().sum().sum()

print(f"Missing values in data: {missing_values_data}")
print(f"Missing values in labels: {missing_values_labels}")
```

```
Missing values in data: 0
Missing values in labels: 0
```

```
# Descriptive statistics for the data
data_description = dataset.describe()

# Distribution of classes in the labels
class_distribution = labelset['Class'].value_counts()

print(data_description)
print(class_distribution)
```

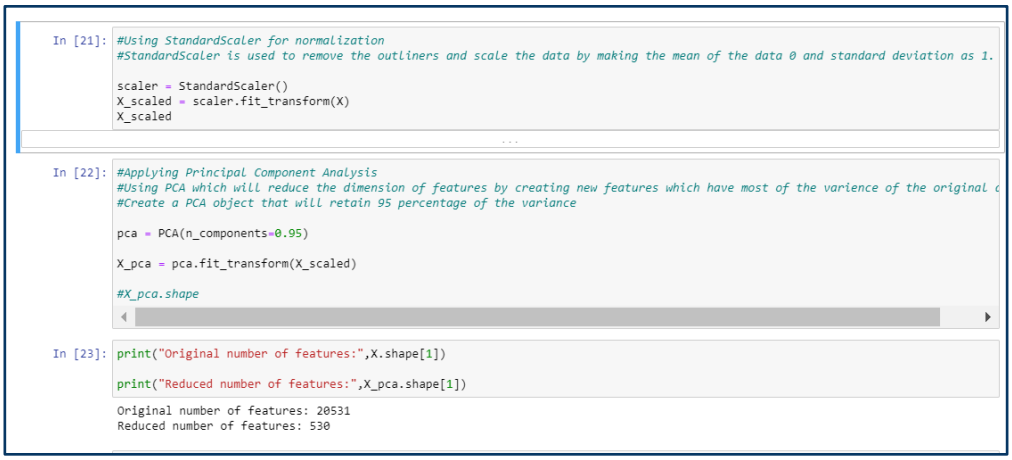
3.1.3 Data Preprocessing:

In the preprocessing stage of our cancer genomics project, we initially load and read the gene expression dataset using the Panda's library, a powerful tool in Python for data manipulation and analysis. The first step in preprocessing involves invoking Panda's *read_csv()* method to efficiently load the gene expression data from a CSV file into a DataFrame. Once the data is loaded, we employ the *dropna()* method from Panda's to remove any null or missing values present in the dataset. Eliminating these values is crucial as they can adversely affect the accuracy of our machine learning models. By ensuring that our dataset is free from null values, we enhance the potential for more accurate and reliable predictions in our subsequent analyses.

```
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing

In [3]: # Loading the Cancer Genomics dataset using pandas library
        # and stored them in the name of variable dataset.
        dataset=pd.read_csv("data.csv")
        dataset

In [7]: print(dataset.dtypes)

In [9]: # Removing the Unnamed columns from dataset
        dataset.drop(dataset.columns[dataset.columns.str.contains('Unnamed',case = False)],axis = 1, inplace = True)

In [10]: dataset

In [6]: labelset=pd.read_csv("labels.csv")
        labelset

In [8]: print(labelset.dtypes)

In [12]: # Check for missing values in the data
         missing_values_data = dataset.isnull().sum().sum()
         missing_values_labels = labelset.isnull().sum().sum()

         print(f"Missing values in data: {missing_values_data}")
         print(f"Missing values in labels: {missing_values_labels}")

In [13]: # Descriptive statistics for the data
         data_description = dataset.describe()

         # Distribution of classes in the labels
         class_distribution = labelset['Class'].value_counts()

         print(data_description)
         print(class_distribution)
```

3.1.4 Feature Extraction

After standard data scaling, Principal Component Analysis (PCA) is the method used to extract features. For PCA to be effective, *StandardScaler* standardizes [19] the features to have zero mean and unit variance. This is especially important for high-dimensional datasets like Cancer Genomics. PCA is used in two ways: first, it reduces the dataset to a smaller set of characteristics that account for 95% of the variance, which helps effective model training; second, it keeps only two or three components, which allows the data to be seen in two and three dimensions. This method offers a visual representation of the data distribution and successfully minimizes the dataset's complexity, ensuring that machine learning models are trained on a reduced but useful feature set.



3.1.5 Unsupervised Learning Model

The core objective of the unsupervised learning component is to apply K-means clustering on PCA-transformed data, and then evaluate the results using silhouette analysis. First, a K-means model is created with parameters such as the maximum number of iterations, initialization method, and number of clusters [17]. After that, the model is fitted to PCA-transformed data to find unique clusters. For different cluster counts, silhouette analysis is performed to assess and establish the ideal number of clusters. To determine the distance between the generated clusters, this methodology generates the average silhouette score for each cluster count. To give a thorough understanding of the clustering performance and to help choose the ideal number of clusters for the dataset, visualizations are made to show both the silhouette scores and the clustered data.

3.1.6 Supervised Machine Learning Classification Models

In the supervised learning component, a PCA-transformed dataset is used to evaluate the performance of various machine learning classifiers, such as Random Forest, Support Vector Machines, Gradient Boosting, Neural Networks, and XGBoost Gradient [16][19][20]. Using *train_test_split*, the data is first divided into training and testing sets in a 70:30 ratio. As a result, it can return the parameters for *x_train*, *x_test*, *y_train*, and *y_test*. It then uses the inputs *x_train*

and *y_train* to generate the training model with the suitable classifiers, and it obtains the predicted classes *y_pred* by calling metrics after invoking the prediction function and using *x_test* as an *input.accuracy_score()* returns the accuracy of each individual classifier after receiving the inputs *y_test* and *y_pred*. Each classifier is trained on the converted training data after PCA is applied for dimensionality reduction. It is then utilized to predict results on the transformed test set. Metrics including accuracy, precision, recall, F1-score, and a classification report are used to evaluate each classifier's performance and provide each model's accuracy.

3.1.7 Performance Evaluation

For the supervised learning models, performance evaluation and visualization are key components. A comprehensive evaluation of each model, including Random Forest, SVM, Gradient Boosting, MLPClassifier, and XGBoost, is provided through a variety of factors, including accuracy, precision, recall, and F1-score. Classification reports that provide information on how each model performs in specific classes help to further explain these measures. Furthermore, to visually represent the performance and highlight the true positives, false positives, true negatives, and false negatives for each class, visualizations such as confusion matrices are used. These visual aids are very helpful in identifying the models' strong and weak points, enabling a deeper understanding of their behavior, and optimizing the tuning and selection of future models.

3.1.8 Identifying the Best Classification Model

The best classification algorithm is identified by comparing several models on a dataset using key performance metrics. Models like Random Forest, SVM, Gradient Boosting, MLPClassifier, and XGBoost are trained on a PCA-transformed dataset and evaluated based on accuracy, precision, recall, and F1-score. These metrics provide a comprehensive view of each model's predictive ability, highlighting overall performance, balance between false positives and false negatives, and

harmonic means of precision and recall. Examining these metrics, particularly in classification reports and confusion matrices, helps identify the most effective model for specific data and objectives.

4. SYSTEM IMPLEMENTATION

4.1 Software Requirements

Hardware Requirements:

Processor: Any Update Processor

Ram: Min 4 GB

Hard Disk: Min 100 GB

Software Requirements:

Operating System: Windows family

Technology: Python 3.6

IDE: PyCharm, Jupyter Notebook

I first installed the Python software (python-3.8-amd64.exe setup file) to implement the Python application before beginning the creation of the Cancer Genomic Project. After that, I set up the PyCharm IDE framework for Python code development, which includes the pycharm-community-2017.3.exe configuration. At last, I started generating a new project using the PyCharm IDE framework.

4.2 Libraries Used

- Pandas: This library is used for data manipulation and analysis.
- NumPy: This Library is used for numerical computations.
- Scikit-learn: This library is used for importing the ML classifiers and calculations of ML algorithms performance metrics.
- Matplotlib and Seaborn: These libraries are used for generating bar charts with the accuracies of ML techniques.

4.3 Methodologies

4.3.1 K - Means Clustering

In this project, K-Means clustering is a basic technique that helps discover hidden patterns and classify genetic data into different groups according to similarity. The project analyzes high-dimensional genomic data, which is frequently reduced using PCA for improved visualization and efficiency, using the *sklearn* package. The technique generates a model, allocates data points to the closest cluster centers, and computes the silhouette score by iterating over a range of cluster numbers. Higher values indicate more distinct clustering. This score gives information about the distance between groups.

```
In [32]: from sklearn.cluster import KMeans

In [33]: kmeans = KMeans(n_clusters=5, init="k-means++", n_init=50, max_iter=500, random_state=42)
         # Fit the KMeans algorithm to the PCA-transformed data
         kmeans.fit(X_pca)
         clusters = kmeans.labels_

In [37]: plt.figure(figsize=(8, 8))

         # Unique clusters
         unique_clusters = np.unique(clusters)

         # Loop through all unique clusters
         for cluster in unique_clusters:
             # Mask to separate sets of data
             mask = clusters == cluster
             plt.scatter(X_pca[mask, 0], X_pca[mask, 1], s=50, label=f"Cluster {cluster}", cmap='plasma')

         # Plot centroids
         centroids = kmeans.cluster_centers_
         plt.scatter(centroids[:, 0], centroids[:, 1], c='black', marker='X', s=50, label="Centroids")

         plt.xlabel('Principal Component 1')
         plt.ylabel('Principal Component 2')
         plt.title('KMeans Clustering Results')
         plt.legend(bbox_to_anchor=(1, 1), loc=2)
         plt.grid()
         plt.show()
```

4.3.2 Random Forest

One machine learning approach that works very well in cancer genomics is the Random Forest classifier [6]. It is a supervised learning technique that combines decision trees to identify complicated correlations in data. Each decision tree is trained on a different collection of features and data. Every decision tree in the model gets put into a discussion, and the class with the most results becomes the prediction of the model. The huge dimensionality and complicated

relationships of genomic data make the RF classifier especially well-suited for it. Its superior accuracy and durability over other algorithms in the analysis of genomic data has been shown. The *sklearn.ensemble* package in Python frequently uses the *RandomForestClassifier()* to construct prediction models.

```
In [49]: from sklearn.ensemble import RandomForestClassifier
         from sklearn.metrics import f1_score, precision_score, accuracy_score, recall_score,confusion_matrix
         from mlxtend.plotting import plot_confusion_matrix
         from sklearn.metrics import classification_report

         rfc_clf = RandomForestClassifier()

         rfc_clf.fit(X_train_pca, y_train)

         y_pred = rfc_clf.predict(X_test_pca)

         target_names =['BRCA', 'COAD', 'KIRC','LUAD','PRAD']

         rf_accuracy = accuracy_score(y_test, y_pred)
         rf_precision = precision_score(y_test, y_pred, average="macro")
         rf_recall = recall_score(y_test, y_pred, average="macro")
         rf_fscore = f1_score(y_test, y_pred, average="macro")

         print(f"Random Forest Metrics:")
         print(f"Accuracy: {rf_accuracy*100:.2f}%")
         print(f"Precision (macro avg): {rf_precision*100:.2f}%")
         print(f"Recall (macro avg): {rf_recall*100:.2f}%")
         print(f"F1-score (macro avg): {rf_fscore*100:.2f}%")
         print("\nClassification Report:")
         print(classification_report(y_test, y_pred, target_names=target_names))
```

4.3.2 Support Vector Machine

The Support Vector Machine (SVM) Classifier is an effective approach for categorizing genetic data into various cancer types. It is particularly skilled at locating the ideal hyperplane that maximizes the margin between various cancer types and can handle high-dimensional data. To train the SVM classifier, some of the dataset is used [7]. PCA is used to minimize dimensionality while maintaining important features. The algorithm predicts cancer types for the test dataset after training. Metrics like accuracy, precision, recall, and F1-score are used to evaluate how successful the SVM classifier is. The *classification_report* function provides information about the model's positives and negatives in terms of cancer classification.

```
In [53]: from sklearn.svm import SVC
         from sklearn.metrics import f1_score, precision_score, accuracy_score, recall_score,confusion_matrix
         from mlxtend.plotting import plot_confusion_matrix
         from sklearn.metrics import classification_report

         svm_clf = SVC()

         svm_clf.fit(X_train_pca, y_train)

         y_pred = svm_clf.predict(X_test_pca)

         target_names =['BRCA', 'COAD', 'KIRC','LUAD','PRAD']

         svm_accuracy = accuracy_score(y_test, y_pred)
         svm_precision = precision_score(y_test, y_pred, average="macro")
         svm_recall = recall_score(y_test, y_pred, average="macro")
         svm_fscore = f1_score(y_test, y_pred, average="macro")

         print(f"Support Vector Machines Metrics:")
         print(f"Accuracy: {svm_accuracy*100:.2f}%")
         print(f"Precision (macro avg): {svm_precision*100:.2f}%")
         print(f"Recall (macro avg): {svm_recall*100:.2f}%")
         print(f"F1-score (macro avg): {svm_fscore*100:.2f}%")
         print("\nClassification Report:")
         print(classification_report(y_test, y_pred, target_names=target_names))
```

4.3.3. Neural Networks

Multi-Layer Perceptrons (MLP), a type of neural network, are used in this project to recognize and analyze complex genetic data. The *sklearn.neural_network* [20] library's MLPClassifier is used to handle complicated patterns and high-dimensional spaces in genomic data. Utilizing PCA to reduce dimensionality while maintaining genes, the network is trained on a modified genomic dataset. Performance metrics like accuracy, precision, recall, and F1-score are computed for the entire model as well as for other cancer types when assessing the model's performance on a test dataset. For every type of cancer, the *classification_report* offers a thorough examination of the model's performance. The study of cancer genomics has advanced significantly with the use of this approach.

```
In [64]: from sklearn.neural_network import MLPClassifier
         from sklearn.metrics import f1_score, precision_score, accuracy_score, recall_score, confusion_matrix, classification_report
         from mlxtend.plotting import plot_confusion_matrix
         import matplotlib.pyplot as plt

         # Initialize the Neural Network classifier
         # Here I am using one hidden layer with 100 neurons
         nn_clf = MLPClassifier(hidden_layer_sizes=(100,), max_iter=300, activation='relu', solver='adam', random_state=1)

         # Train the model
         nn_clf.fit(X_train_pca, y_train)

         # Predict on the test set
         y_pred = nn_clf.predict(X_test_pca)

         target_names = ['BRCA', 'COAD', 'KIRC', 'LUAD', 'PRAD']

         # Compute the metrics
         nn_accuracy = accuracy_score(y_test, y_pred)
         nn_precision = precision_score(y_test, y_pred, average="macro")
         nn_recall = recall_score(y_test, y_pred, average="macro")
         nn_fscore = f1_score(y_test, y_pred, average="macro")

         # Print the metrics
         print(f"Neural Network Metrics:")
         print(f"Accuracy: {nn_accuracy*100:.2f}%")
         print(f"Precision (macro avg): {nn_precision*100:.2f}%")
         print(f"Recall (macro avg): {nn_recall*100:.2f}%")
         print(f"F1-score (macro avg): {nn_fscore*100:.2f}%")
         print("\nClassification Report:")
         print(classification_report(y_test, y_pred, target_names=target_names))
```

4.3.4 Gradient Boost

An effective machine learning technique used in the cancer genomics project for the analysis and classification of complex genetic data is the Gradient Boosting Classifier (GBC) [15]. To increase prediction accuracy, it employs an ensemble of decision trees, whereby each new tree fixes errors made by previous ones. Principal Component Analysis (PCA) is used to reduce the dimensionality of the dataset used to train the GBC model [8]. A test set of different forms of cancer is used to assess the model. To evaluate the efficacy of the model, critical performance measures such as accuracy, precision, recall, and F1-score are calculated. The *classification_report* offers a thorough analysis for every distinct cancer type, giving a fine-grained comprehension of the model's advantages and disadvantages in dividing up cancers into their various subtypes.

```
In [58]: from sklearn.ensemble import GradientBoostingClassifier
         from sklearn.metrics import f1_score, precision_score, accuracy_score, recall_score, confusion_matrix
         from mlxtend.plotting import plot_confusion_matrix
         from sklearn.metrics import classification_report

         # Initialize the Gradient Boosting Classifier
         gb_clf = GradientBoostingClassifier()

         # Fit the model
         gb_clf.fit(X_train_pca, y_train)

         # Make predictions
         y_pred = gb_clf.predict(X_test_pca)

         # Define the target names
         target_names = ['BRCA', 'COAD', 'KIRC', 'LUAD', 'PRAD']

         # Calculate metrics
         gb_accuracy = accuracy_score(y_test, y_pred)
         gb_precision = precision_score(y_test, y_pred, average="macro")
         gb_recall = recall_score(y_test, y_pred, average="macro")
         gb_fscore = f1_score(y_test, y_pred, average="macro")

         # Print metrics
         print(f"Gradient Boosted Decision Tree Metrics:")
         print(f"Accuracy: {gb_accuracy*100:.2f}%")
         print(f"Precision (macro avg): {gb_precision*100:.2f}%")
         print(f"Recall (macro avg): {gb_recall*100:.2f}%")
         print(f"F1-score (macro avg): {gb_fscore*100:.2f}%")
         print("\nClassification Report:")
         print(classification_report(y_test, y_pred, target_names=target_names))
```

4.3.5 XG Boost

The XGBoost classifier analyzes complex genetic data to categorize various cancer types. In order to effectively handle the high-dimensional features of genetic data, it is used as a dataset modified by Principal Component Analysis (PCA) [10]. Using a split dataset, the model is trained and tested with the multi-class classification objective set to "*multi:softprob.*" The model is customized to the particulars of the dataset by dynamically adjusting the *num_class* parameter to the number of unique labels. Metrics including accuracy, precision, recall, and F1-score are used to assess the performance of the XGBoost classifier, giving valuable information about how well it works and how much it could decrease false positives and negatives. The detailed classification report goes deeper into the effectiveness of each type of cancer, providing researchers with an understanding of its advantages and disadvantages in various genetic scenarios.

```python
# xgboost

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import xgboost as xgb
from sklearn.metrics import accuracy_score, classification_report

# Transform labels to numerical format
label_encoder = LabelEncoder()

y_encoded = label_encoder.fit_transform(Y)

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.3, random_state=42)

X_train_pca = pca.fit_transform(X_train)

X_test_pca = pca.transform(X_test)

# Initialize XGBoost classifier
xgb_clf = xgb.XGBClassifier(objective='multi:softprob', num_class=len(set(y_encoded)), seed=42)

# Fit the classifier to the training data
xgb_clf.fit(X_train_pca, y_train)

# Predict on the test data
y_pred = xgb_clf.predict(X_test_pca)

# Compute the metrics
xgb_accuracy = accuracy_score(y_test, y_pred)
xgb_precision = precision_score(y_test, y_pred, average="macro")
xgb_recall = recall_score(y_test, y_pred, average="macro")
xgb_fscore = f1_score(y_test, y_pred, average="macro")

# Print the metrics
print(f"XGBoost classifier Metrics:")
print(f"Accuracy: {xgb_accuracy*100:.2f}%")
print(f"Precision (macro avg): {xgb_precision*100:.2f}%")
print(f"Recall (macro avg): {xgb_recall*100:.2f}%")
print(f"F1-score (macro avg): {xgb_fscore*100:.2f}%")
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=target_names))
```

5. EXPERIMENT RESULTS AND ANALYSIS

```
In [3]: # Loading the Cancer Genomics dataset using pandas library
        # and stored them in the name of variable dataset.
        dataset=pd.read_csv("data.csv")
        dataset
```

Out[3]:

	Unnamed: 0	gene_0	gene_1	gene_2	gene_3	gene_4	gene_5	gene_6	gene_7	gene_8	...	gene_20521	gene_20522	gene_20523	gene_20524
0	sample_0	0.0	2.017209	3.265527	5.478487	10.431999	0.0	7.175175	0.591871	0.0	...	4.926711	8.210257	9.723516	7.220030
1	sample_1	0.0	0.592732	1.588421	7.586157	9.623011	0.0	6.816049	0.000000	0.0	...	4.593372	7.323865	9.740931	6.256586
2	sample_2	0.0	3.511759	4.327199	6.881787	9.870730	0.0	6.972130	0.452595	0.0	...	5.125213	8.127123	10.908640	5.401607
3	sample_3	0.0	3.663618	4.507649	6.659068	10.196184	0.0	7.843375	0.434882	0.0	...	6.076566	8.792959	10.141520	8.942805
4	sample_4	0.0	2.655741	2.821547	6.539454	9.738265	0.0	6.566967	0.360982	0.0	...	5.996032	8.891425	10.373790	7.181162
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
796	sample_796	0.0	1.865642	2.718197	7.350099	10.006003	0.0	6.764792	0.496922	0.0	...	6.088133	9.118313	10.004852	4.484415
797	sample_797	0.0	3.942955	4.453807	6.346597	10.056868	0.0	7.320331	0.000000	0.0	...	6.371876	9.623335	9.823921	6.555327
798	sample_798	0.0	3.249582	3.707492	8.185901	9.504082	0.0	7.536589	1.811101	0.0	...	5.719386	8.610704	10.485517	3.589763
799	sample_799	0.0	2.590339	2.787976	7.318624	9.987136	0.0	9.213464	0.000000	0.0	...	5.785237	8.605387	11.004677	4.745888
800	sample_800	0.0	2.325242	3.805932	6.530246	9.560367	0.0	7.957027	0.000000	0.0	...	6.403075	8.594354	10.243079	9.139459

801 rows × 20532 columns

Fig 5.1 Loading Data dataset

Fig 5.1 shows the output of gene expression data from the Data dataset, organized in rows and columns, representing the expression levels of each gene in each sample, with each row representing a different sample of Size of 801 * 20532.

```
In [6]: labelset=pd.read_csv("labels.csv")
        labelset
```

Out[6]:

	Unnamed: 0	Class
0	sample_0	PRAD
1	sample_1	LUAD
2	sample_2	PRAD
3	sample_3	PRAD
4	sample_4	BRCA
...	...	...
796	sample_796	BRCA
797	sample_797	LUAD
798	sample_798	COAD
799	sample_799	PRAD
800	sample_800	PRAD

801 rows × 2 columns

Fig 5.2 Loading Labels Dataset

Fig 5.2 shows a list of labels that the samples are from patients with different types of cancer of 801 samples.

```
In [7]: print(dataset.dtypes)

gene_0        float64
gene_1        float64
gene_2        float64
gene_3        float64
gene_4        float64
               ...
gene_20526    float64
gene_20527    float64
gene_20528    float64
gene_20529    float64
gene_20530    float64
Length: 20531, dtype: object
```

Fig 5.3 Data Types of Data set

Fig 5.3 shows the dataset has 20,531 variables, with datatype of floating-point numbers.

```
In [8]: print(labelset.dtypes)

Unnamed: 0    object
Class         object
dtype: object
```

Fig 5.4 Data Types of Label set

Fig 5.4 shows the data types of the variables for the label set.

```
In [9]: # Removing the Unnamed columns from dataset
dataset.drop(dataset.columns[dataset.columns.str.contains('Unnamed',case = False)],axis = 1, inplace = True)

In [10]: dataset
```

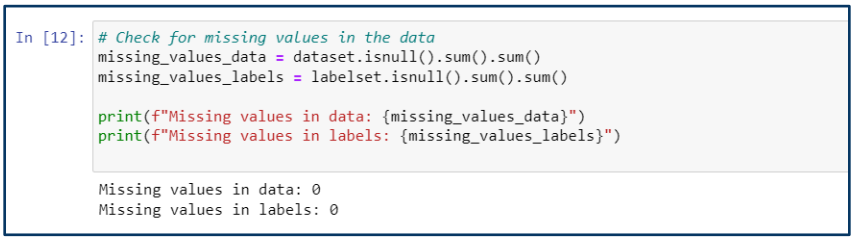
Out[10]:

	gene_0	gene_1	gene_2	gene_3	gene_4	gene_5	gene_6	gene_7	gene_8	gene_9	...	gene_20521	gene_20522	gene_20523	gene_20524	ge
0	0.0	2.017209	3.265527	5.478487	10.431999	0.0	7.175175	0.591871	0.0	0.0	...	4.926711	8.210257	9.723516	7.220030	
1	0.0	0.592732	1.588421	7.586157	9.623011	0.0	6.816049	0.000000	0.0	0.0	...	4.593372	7.323865	9.740931	6.256586	
2	0.0	3.511759	4.327199	6.881787	9.870730	0.0	6.972130	0.452595	0.0	0.0	...	5.125213	8.127123	10.908640	5.401607	
3	0.0	3.663618	4.507649	6.659068	10.196184	0.0	7.843375	0.434882	0.0	0.0	...	6.076566	8.792959	10.141520	8.942805	
4	0.0	2.655741	2.821547	6.539454	9.738265	0.0	6.566967	0.360982	0.0	0.0	...	5.996032	8.891425	10.373790	7.181162	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
796	0.0	1.865642	2.718197	7.350099	10.006003	0.0	6.764792	0.496922	0.0	0.0	...	6.088133	9.118313	10.004852	4.484415	
797	0.0	3.942955	4.453807	6.346597	10.056868	0.0	7.320331	0.000000	0.0	0.0	...	6.371876	9.623335	9.823921	6.555327	
798	0.0	3.249582	3.707492	8.185901	9.504082	0.0	7.536589	1.811101	0.0	0.0	...	5.719386	8.610704	10.485517	3.589763	
799	0.0	2.590339	2.787976	7.318624	9.987136	0.0	9.213464	0.000000	0.0	0.0	...	5.785237	8.605387	11.004677	4.745888	
800	0.0	2.325242	3.805932	6.530246	9.560367	0.0	7.957027	0.000000	0.0	0.0	...	6.403075	8.594354	10.243079	9.139459	1

801 rows × 20531 columns

Fig 5.5 Removing the Unwanted Columns from the dataset.

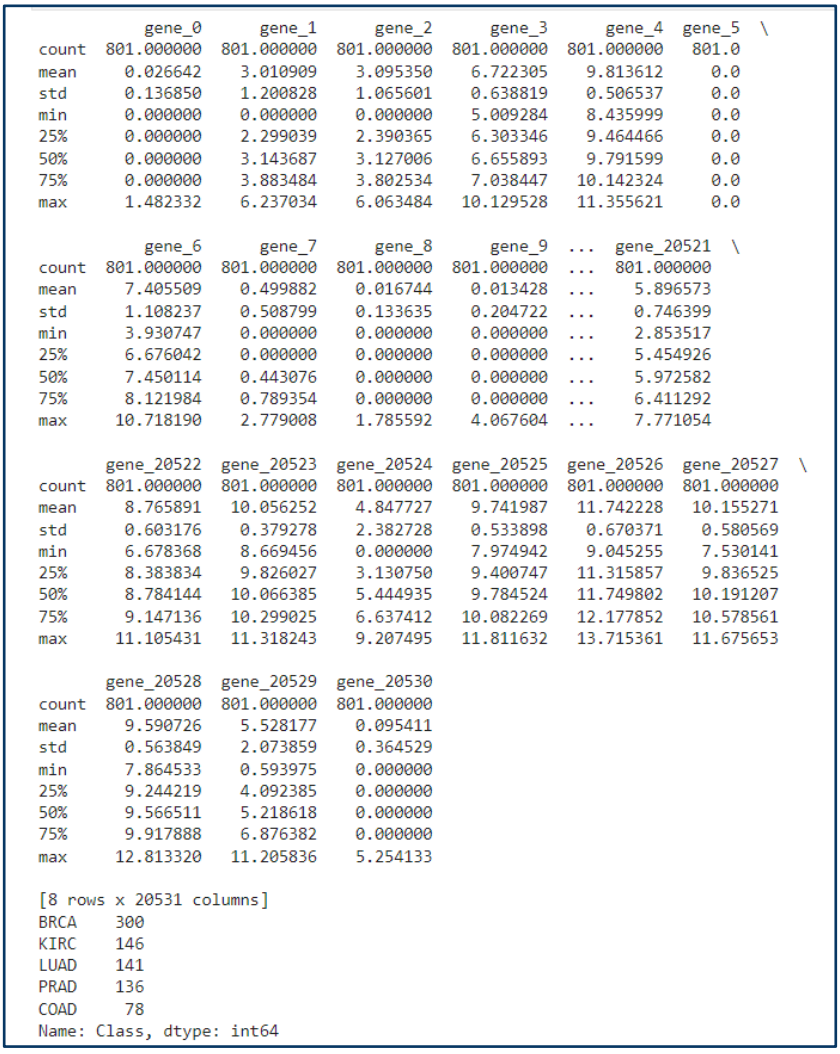
Fig 5.5 shows the output of removing the null or missing values present in the dataset.



```
In [12]: # Check for missing values in the data
         missing_values_data = dataset.isnull().sum().sum()
         missing_values_labels = labelset.isnull().sum().sum()

         print(f"Missing values in data: {missing_values_data}")
         print(f"Missing values in labels: {missing_values_labels}")

         Missing values in data: 0
         Missing values in labels: 0
```

Fig 5.6 Missing values in the dataset

Fig 5.6 showing if there are any missing values in both the datasets. The results show 0 missing values for both sets.



```
            gene_0      gene_1      gene_2      gene_3      gene_4  gene_5  \
count  801.000000  801.000000  801.000000  801.000000  801.000000   801.0
mean     0.026642    3.010909    3.095350    6.722305    9.813612     0.0
std      0.136850    1.200828    1.065601    0.638819    0.506537     0.0
min      0.000000    0.000000    0.000000    5.009284    8.435999     0.0
25%      0.000000    2.299039    2.390365    6.303346    9.464466     0.0
50%      0.000000    3.143687    3.127006    6.655893    9.791599     0.0
75%      0.000000    3.883484    3.802534    7.038447   10.142324     0.0
max      1.482332    6.237034    6.063484   10.129528   11.355621     0.0

            gene_6      gene_7      gene_8      gene_9  ...  gene_20521  \
count  801.000000  801.000000  801.000000  801.000000  ...  801.000000
mean     7.405509    0.499882    0.016744    0.013428  ...    5.896573
std      1.108237    0.508799    0.133635    0.204722  ...    0.746399
min      3.930747    0.000000    0.000000    0.000000  ...    2.853517
25%      6.676042    0.000000    0.000000    0.000000  ...    5.454926
50%      7.450114    0.443076    0.000000    0.000000  ...    5.972582
75%      8.121984    0.789354    0.000000    0.000000  ...    6.411292
max     10.718190    2.779008    1.785592    4.067604  ...    7.771054

       gene_20522  gene_20523  gene_20524  gene_20525  gene_20526  gene_20527  \
count  801.000000  801.000000  801.000000  801.000000  801.000000  801.000000
mean     8.765891   10.056252    4.847727    9.741987   11.742228   10.155271
std      0.603176    0.379278    2.382728    0.533898    0.670371    0.580569
min      6.678368    8.669456    0.000000    7.974942    9.045255    7.530141
25%      8.383834    9.826027    3.130750    9.400747   11.315857    9.836525
50%      8.784144   10.066385    5.444935    9.784524   11.749802   10.191207
75%      9.147136   10.299025    6.637412   10.082269   12.177852   10.578561
max     11.105431   11.318243    9.207495   11.811632   13.715361   11.675653

       gene_20528  gene_20529  gene_20530
count  801.000000  801.000000  801.000000
mean     9.590726    5.528177    0.095411
std      0.563849    2.073859    0.364529
min      7.864533    0.593975    0.000000
25%      9.244219    4.092385    0.000000
50%      9.566511    5.218618    0.000000
75%      9.917888    6.876382    0.000000
max     12.813320   11.205836    5.254133

[8 rows x 20531 columns]
BRCA    300
KIRC    146
LUAD    141
PRAD    136
COAD     78
Name: Class, dtype: int64
```

Fig 5.7 Description of the datasets

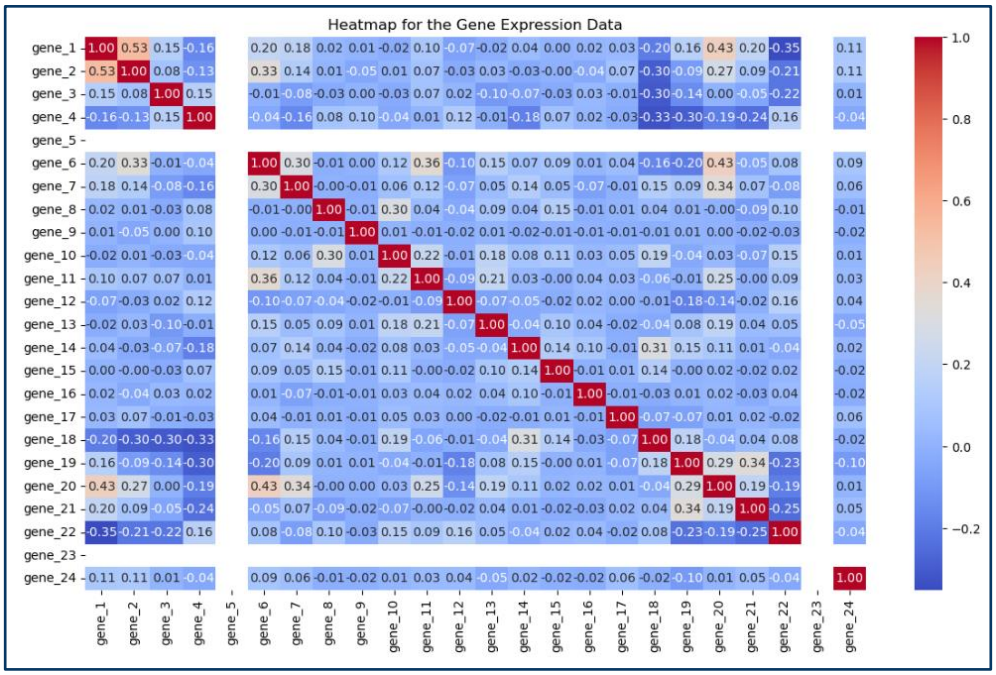


Fig 5.8 Correlation matrix for gene expression data

The above Fig 5.8 displays a correlation matrix for gene expression data, aiding in identifying co-expressed gene groups and understanding biological processes regulating gene expression. Red indicates a strong positive correlation, blue indicates a strong negative correlation, and white indicates no correlation.

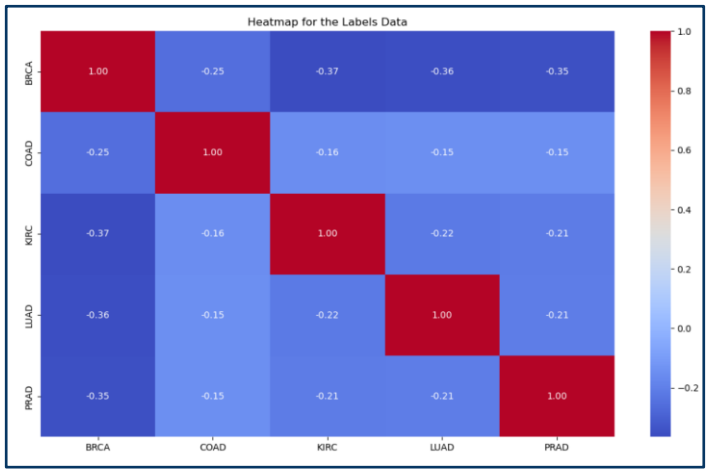


Fig 5.9 Heatmap of Labels Dataset

The image you sent shows a heatmap of the label's data for the BRCA, KIRC, LUAD, PRAD, and COAD cancer types. Red indicates high expression, blue indicates low expression, and white indicates no expression. Here in this output, it shows that some genes are highly expressed in all five cancer types, while other genes are only highly expressed in one or two cancer types.

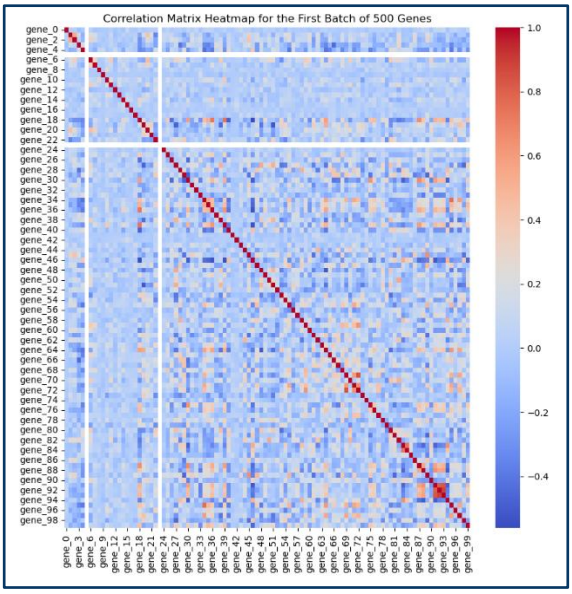


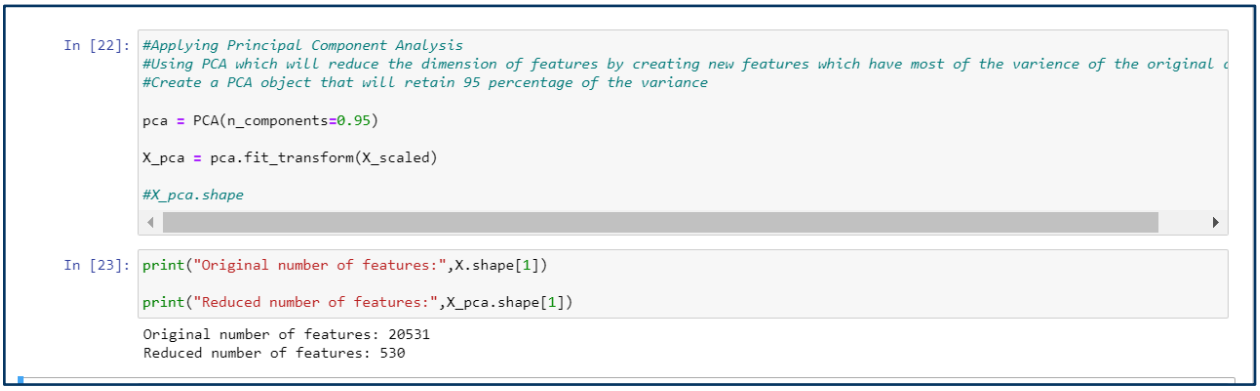
Fig 5.10 Correlation Matrix Heatmap for the First Batch of 100 Genes

Fig 5.10 shows the correlation of the first 100 genes between each pair of variables in a dataset. The cell's color indicates the strength of the positive or negative correlation between two genes, with a whiter cell indicating no linear correlation.

```
In [21]: #Using StandardScaler for normalization
         #StandardScaler is used to remove the outliners and scale the data by making the mean of the data 0 and standard deviation as 1.

         scaler = StandardScaler()
         X_scaled = scaler.fit_transform(X)
         X_scaled

Out[21]: array([[-0.19479935, -0.82802988,  0.15980044, ..., -1.18793812,
                  -0.11648251, -0.26190144],
                 [-0.19479935, -2.01501735, -1.415042  , ..., -0.34227662,
                  -1.65688871, -0.26190144],
                 [-0.19479935,  0.41734754,  1.15673547, ...,  0.88686027,
                  -1.85526414, -0.26190144],
                 ...,
                 [-0.19479935,  0.19888076,  0.57481583, ..., -0.22008186,
                  -0.41046699,  1.3485582 ],
                 [-0.19479935, -0.35045311, -0.28863152, ...,  1.43719268,
                   0.09195083, -0.26190144],
                 [-0.19479935, -0.57135218,  0.66725377, ...,  0.45087581,
                  -0.47161901, -0.26190144]])
```

Fig 5.11 Normalizing the data.

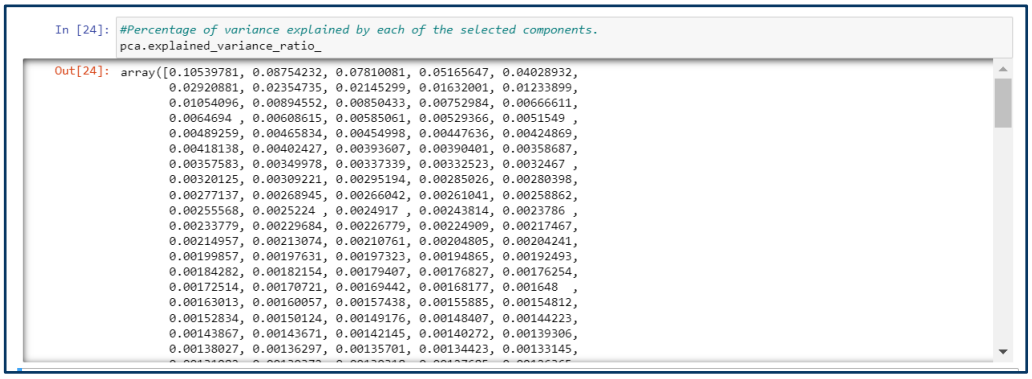
Fig 5.11 is the normalized data by applying *StandardScaler*, the output is an array of numbers that are centered around 0 and have a standard deviation of 1. This means that the data is now on a consistent scale, which can improve the performance of machine learning algorithms.



```
In [22]: #Applying Principal Component Analysis
         #Using PCA which will reduce the dimension of features by creating new features which have most of the varience of the original d
         #Create a PCA object that will retain 95 percentage of the variance

         pca = PCA(n_components=0.95)

         X_pca = pca.fit_transform(X_scaled)

         #X_pca.shape

In [23]: print("Original number of features:",X.shape[1])

         print("Reduced number of features:",X_pca.shape[1])

         Original number of features: 20531
         Reduced number of features: 530
```

Fig 5.12 Applying PCA to retain 95% of variance.

This result shows that the PCA was able to maintain 95% of the data's variation while reducing the number of features by more than 97%. This is a significant reduction in dimensionality, which can make it easier to utilize the data. The number of features was reduced from 20,531 to 530.



```
In [24]: #Percentage of variance explained by each of the selected components.
         pca.explained_variance_ratio_

Out[24]: array([0.10539781, 0.08754232, 0.07810081, 0.05165647, 0.04028932,
                0.02920881, 0.02354735, 0.02145299, 0.01632001, 0.01233899,
                0.01054096, 0.00894552, 0.00850433, 0.00752984, 0.00666611,
                0.0064694 , 0.00608615, 0.00585061, 0.00529366, 0.0051549 ,
                0.00489259, 0.00465834, 0.00454998, 0.00447636, 0.00424869,
                0.00418138, 0.00402427, 0.00393607, 0.00390401, 0.00358687,
                0.00357583, 0.00349978, 0.00337339, 0.00332523, 0.0032467 ,
                0.00320125, 0.00309221, 0.00295194, 0.00285026, 0.00280398,
                0.00277137, 0.00268945, 0.00266042, 0.00261041, 0.00258862,
                0.00255568, 0.0025224 , 0.0024917 , 0.00243814, 0.0023786 ,
                0.00233779, 0.00229684, 0.00226779, 0.00224909, 0.00217467,
                0.00214957, 0.00213074, 0.00210761, 0.00204805, 0.00204241,
                0.00199857, 0.00197631, 0.00197323, 0.00194865, 0.00192493,
                0.00184282, 0.00182154, 0.00179407, 0.00176827, 0.00176254,
                0.00172514, 0.00170721, 0.00169442, 0.00168177, 0.001648  ,
                0.00163013, 0.00160057, 0.00157438, 0.00155885, 0.00154812,
                0.00152834, 0.00150124, 0.00149176, 0.00148407, 0.00144223,
                0.00143867, 0.00143671, 0.00142145, 0.00140272, 0.00139306,
                0.00138027, 0.00136297, 0.00135701, 0.00134423, 0.00133145,
```

Fig 5.13 Percentage variance of PCA Data

The output shows the percentage of variance explained by each of the selected principal components is a measure of how much of the total data variation can be attributed to each PC.

```
In [28]: # Conduct the PCA with 2 components.
         pca = PCA(2)
         X_pca = pca.fit_transform(X_scaled)
         X_pca.shape

Out[28]: (801, 2)

In [29]: pca.explained_variance_ratio_

Out[29]: array([0.10539781, 0.08754232])

In [30]: # Concating the target label data with 2 principal components
         df=pd.read_csv("labels.csv")

         pca = PCA(n_components=2)

         principalComponents = pca.fit_transform(X_scaled)

         principalDf = pd.DataFrame(data = principalComponents, columns = ['PC1', 'PC2'])

         finalDf = pd.concat([principalDf, df[['Class']]], axis = 1)

         finalDf
```

Out[30]:

	PC1	PC2	Class
0	-57.446987	95.410981	PRAD
1	-16.919430	0.732469	LUAD
2	-70.345218	-19.303327	PRAD
3	-49.161592	-9.227586	PRAD
4	-18.132534	-51.327797	BRCA
...	...	...	...
796	-12.417385	-42.321574	BRCA
797	-29.415554	28.526281	LUAD
798	-4.133089	15.690015	COAD
799	-30.814758	33.526423	PRAD
800	-22.344557	4.052356	PRAD

801 rows × 3 columns

Fig 5.14 Applying PCA for 2 Components

This output shows that the PCA will transform the data into a 2-dimensional space.

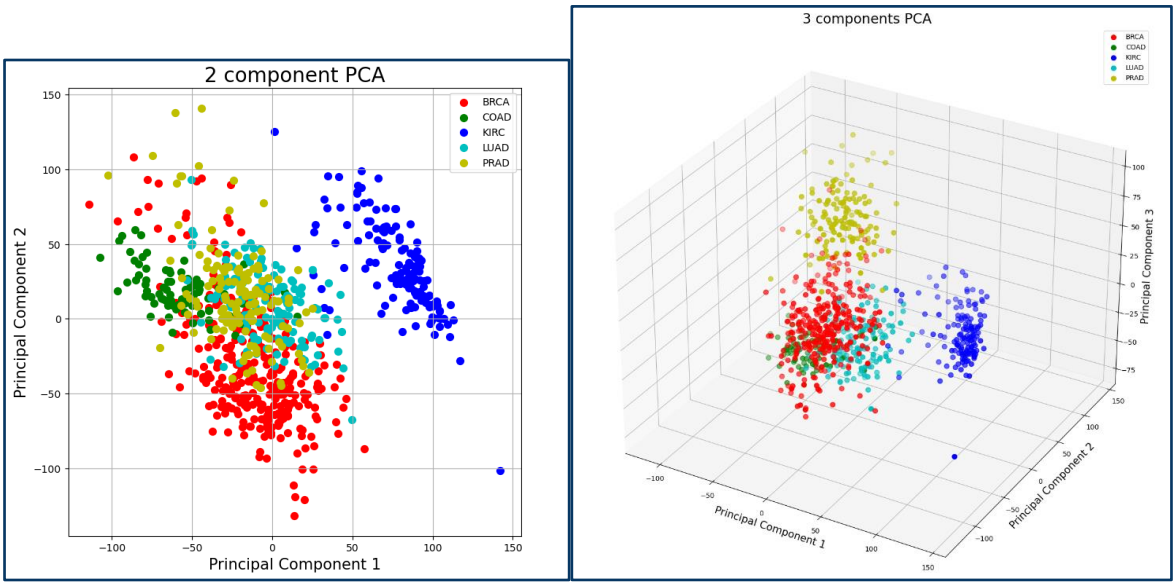


Fig 5.15.1, 5.15.2 Scatter plot for PCA for 2 & 3 Components data respectively.

The above Fig (i), (ii) shows the output of the 2, 3-component PCA of a colorful scatter plot. The scatter plot shows the relationship between the principal component 1 (PC1), the principal component 2 (PC2) and the principal component 3 (PC3).

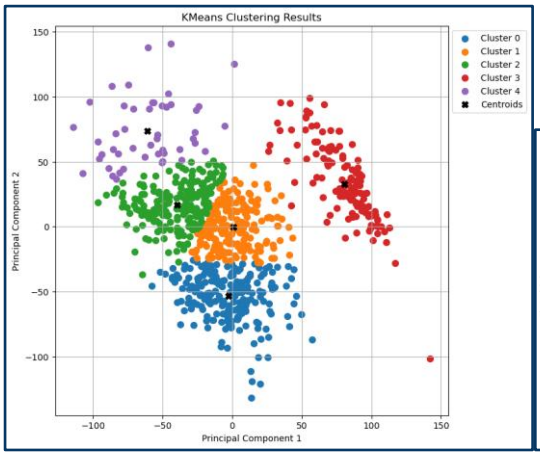


Fig 5.16.1 K-means Clustering

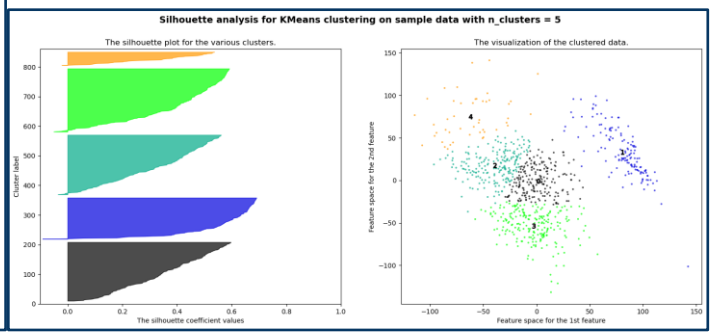


Fig 5.16.2 Silhouette Analysis for 5 clusters.

Fig 5.16.1, 5.16.2 shows the analysis of the k-means clustering algorithm has been divided into 5 clusters, with the data points colored according to their assigned cluster by marking the centroids of the clusters with the X. The silhouette analysis evaluates the quality of the clustering algorithm by calculating the silhouette score for each data point, which ranges from -1 to 1, with higher scores indicating better clustering. The average silhouette score for all data points is 0.75.

Random Forest Classifier:

```
Random Forest Metrics:
Accuracy: 97.52%
Precision (macro avg): 98.37%
Recall (macro avg): 95.78%
F1-score (macro avg): 96.89%

Classification Report:
              precision    recall  f1-score   support

       BRCA       0.95      1.00      0.98        61
       COAD       1.00      0.82      0.90        17
       KIRC       1.00      1.00      1.00        25
       LUAD       1.00      1.00      1.00        29
       PRAD       0.97      0.97      0.97        29

   accuracy                           0.98       161
  macro avg       0.98      0.96      0.97       161
weighted avg      0.98      0.98      0.97       161
```

Fig 5.17.1 RF Classification Report

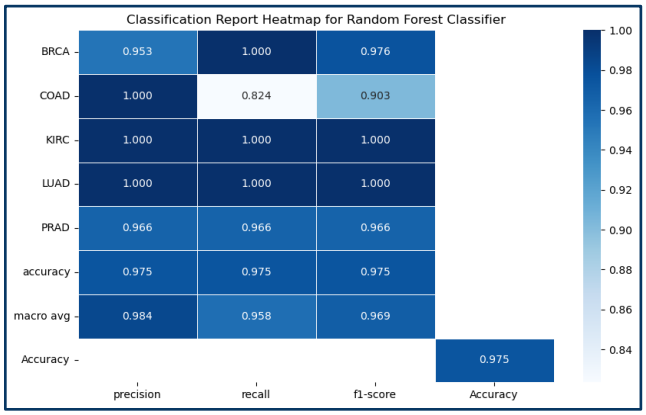


Fig 5.17.2 Heatmap for random forest

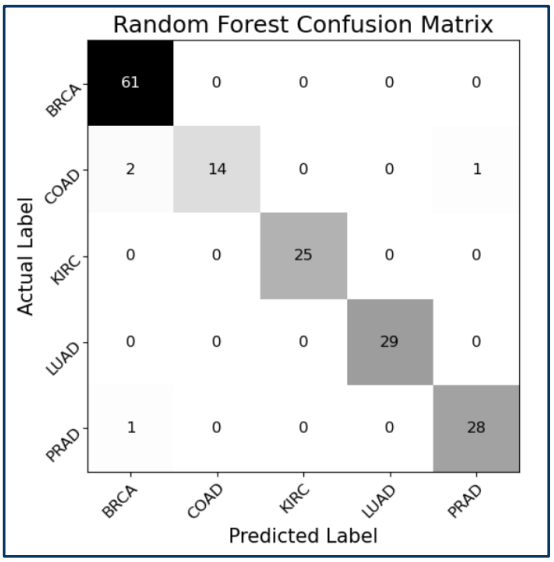


Fig 5.17.3 RF Confusion matrix

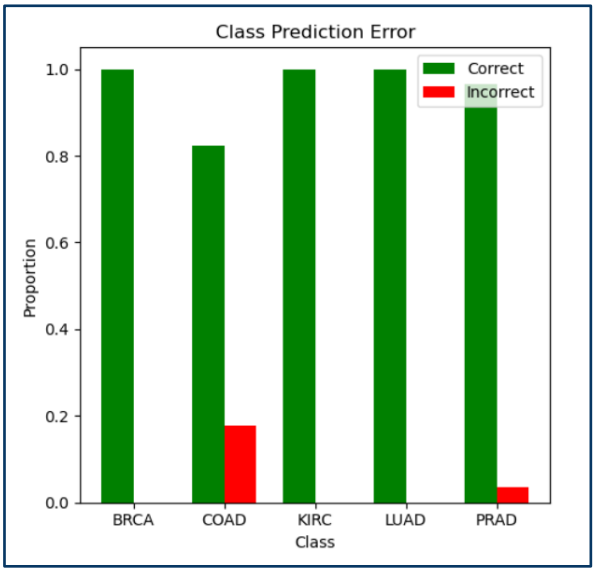


Fig 5.17.4 RF Class Prediction Error

To train the supervised learning model, the data is divided into training and testing sets using a 70:30 ratio to ensure a balanced distribution. Principal Component Analysis (PCA) is applied to reduce the dimensionality of the data while retaining 95% of the variance. The classifier effectively classified the various cancer types in the dataset [6], as seen by its high overall accuracy of 98.76%. With a true positive prediction percentage of 99.35%, it showed a high accuracy rate. Additionally, the classifier's high recall of 98.10% demonstrated its capacity to recognize most true positive

cases across a variety of types. With an F1-score of 98.69%, which maintains a balance between recall and precision, the model's false positive and false negative rates appear to be well-balanced.

Support Vector Machine Classifier:

Support Vector Machines Metrics:
Accuracy: 100.00%
Precision (macro avg): 100.00%
Recall (macro avg): 100.00%
F1-score (macro avg): 100.00%

Classification Report:

	precision	recall	f1-score	support
BRCA	1.00	1.00	1.00	61
COAD	1.00	1.00	1.00	17
KIRC	1.00	1.00	1.00	25
LUAD	1.00	1.00	1.00	29
PRAD	1.00	1.00	1.00	29
accuracy			1.00	161
macro avg	1.00	1.00	1.00	161
weighted avg	1.00	1.00	1.00	161

Fig 5.18.1 SVM Classification Report

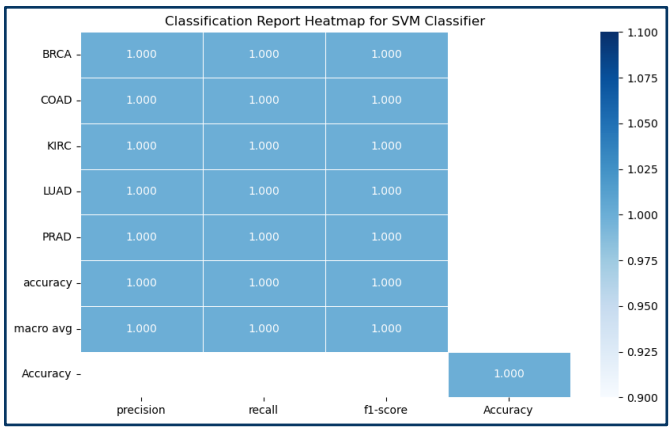


Fig 5.18.2 Heatmap for SVM

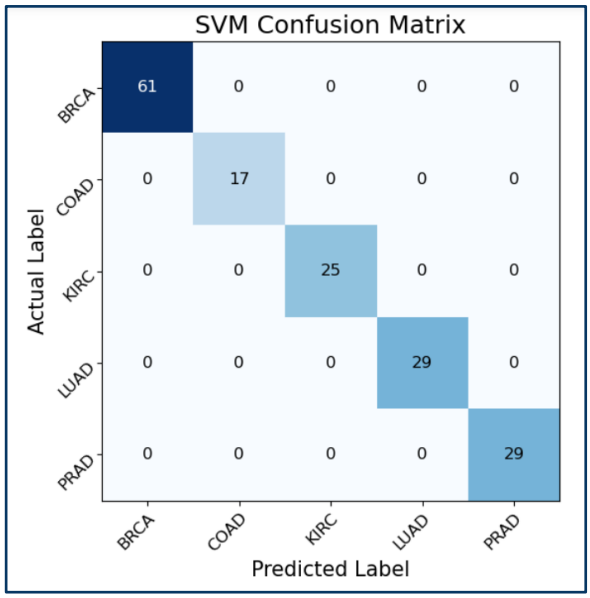


Fig 5.18.3 SVM Confusion matrix

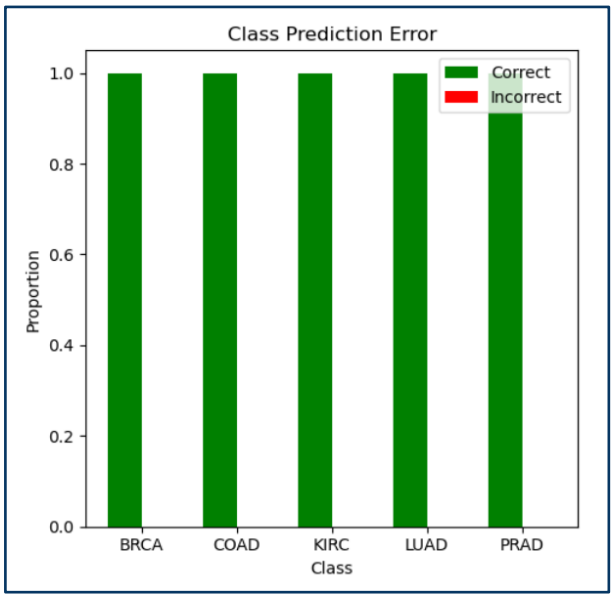


Fig 5.18.4 SVM Class Prediction Error

The SVM classifier performed extremely well, as shown by its remarkable accuracy of 100.00%. The precision of the model indicates the accuracy of positive predictions. The recall of the model

indicates that all test cases of each type of cancer were correctly identified, and it measures the model's capacity to locate all relevant occurrences. Perfect precision and memory are shown by the F1-score, which is also 100.00%, the harmonic mean of precision and recall. These variables are provided for each form of cancer in the classification report; a score of 1.00 denotes perfect classification. These metrics are visualized as a heatmap, which shows consistent excellence in all variables. Predictions are displayed against the actual labels in the confusion matrix, with entries along the diagonal denoting accurate classifications. The precise number of samples for all types that were correctly classified is displayed in text observations. The percentage of true and false predictions for each class is shown on the graph.

Gradient Boost Classifier:s

```
Gradient Boosted Decision Tree Metrics:
Accuracy: 99.38%
Precision (macro avg): 99.33%
Recall (macro avg): 99.31%
F1-score (macro avg): 99.31%

Classification Report:
              precision    recall  f1-score   support

        BRCA       1.00      1.00      1.00        61
        COAD       1.00      1.00      1.00        17
        KIRC       1.00      1.00      1.00        25
        LUAD       0.97      1.00      0.98        29
        PRAD       1.00      0.97      0.98        29

    accuracy                           0.99       161
   macro avg       0.99      0.99      0.99       161
weighted avg       0.99      0.99      0.99       161
```

Fig 5.19.1 GB Classification Report

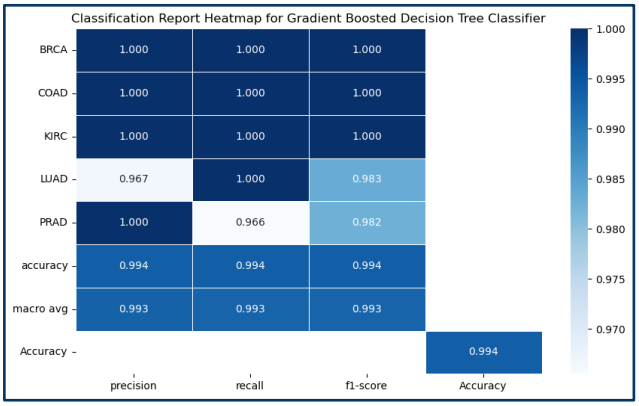
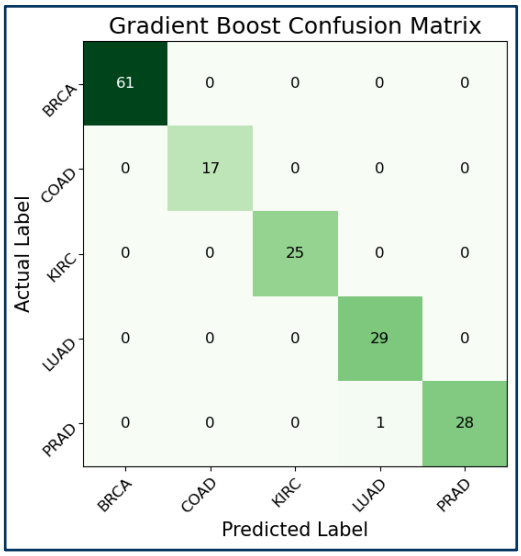


Fig 5.19.2 Heatmap for GB



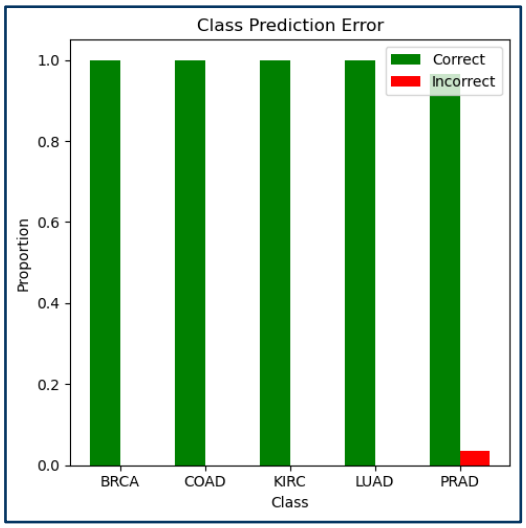


Fig 5.19.3 GB Confusion matrix Fig 5.19.4 GB Class Prediction Error

The Gradient Boosted Decision Tree Metrics have a 99.38% classification accuracy rate for cancer kinds. With a 99.31% recall rate, the model predicts cancer types with a high accuracy rate. A well-balanced and accurate classifier is also indicated by the F1-score, which finds a balance between precision and recall. With high precision, recall, and F1-score for every category, the comprehensive classification report for every form of cancer is provided. Even though the LUAD and PRAD values are marginally reduced, they are still very high and suggest some modest difficulties.

Neural Network Classifier:

```
Neural Network Metrics:
Accuracy: 99.38%
Precision (macro avg): 99.33%
Recall (macro avg): 99.67%
F1-score (macro avg): 99.50%

Classification Report:
              precision    recall  f1-score   support

        BRCA       1.00      0.98      0.99        61
        COAD       1.00      1.00      1.00        17
        KIRC       1.00      1.00      1.00        25
        LUAD       0.97      1.00      0.98        29
        PRAD       1.00      1.00      1.00        29

    accuracy                           0.99       161
   macro avg       0.99      1.00      0.99       161
weighted avg       0.99      0.99      0.99       161
```

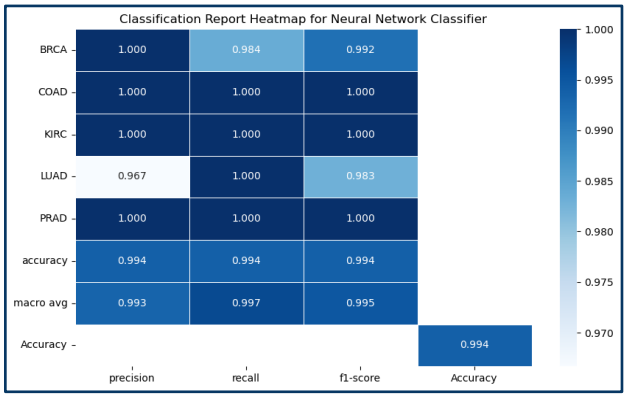
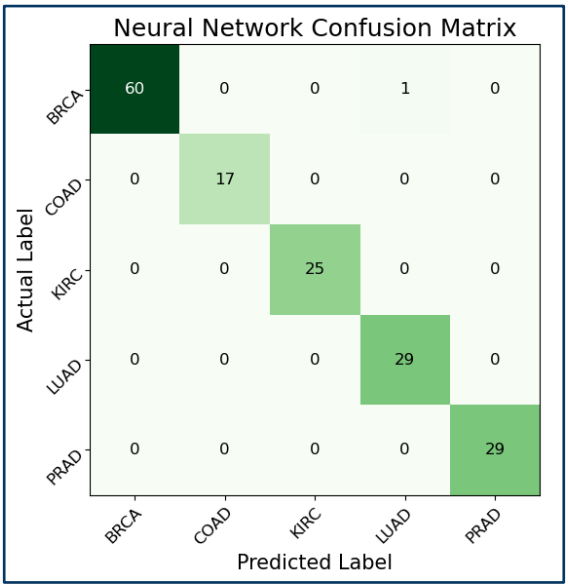


Fig 5.20.1 NN Classification Report Fig 5.20.2 Heatmap for NN



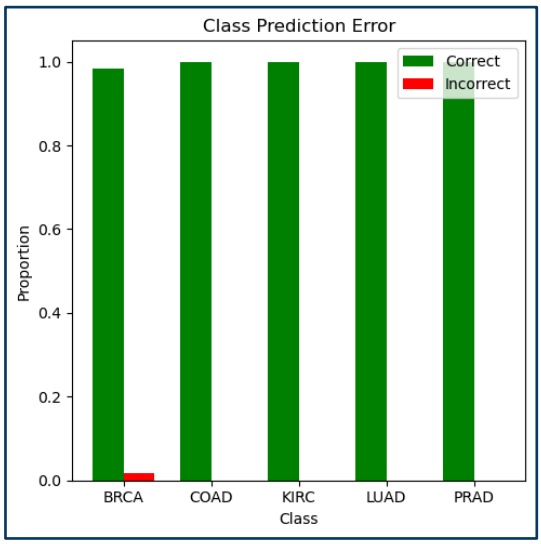


Fig 5.20.3 NN Confusion matrix Fig 5.20.4 NN Class Prediction Error

With an accuracy of 99.38%, the model is highly effective in essentially all cases to identify the forms of cancer. With a precision rate of 99.33%, it nearly always predicts the right type of cancer. With a 99.67% recall rate, it was able to identify almost all the real cases of each form of cancer in the test set. With an F1-score of 99.50%, it is considered accurate and dependable.

Comprehensive metrics are provided for every cancer type in the categorization report, with minor variations in BRCA and LUAD performance. A heatmap, confusion matrix, and class prediction error plot are some of the model's visualizations that show consistent high performance across all measures. As a sign of the great accuracy of the model, the bars for correct predictions should be far higher than those for wrong predictions.

XGBoost Classifier:

```
XGBoost classifier Metrics:
Accuracy: 99.17%
Precision (macro avg): 99.05%
Recall (macro avg): 99.30%
F1-score (macro avg): 99.16%

Classification Report:
              precision    recall  f1-score   support

       BRCA       1.00      0.99      0.99        90
       COAD       1.00      1.00      1.00        28
       KIRC       1.00      1.00      1.00        41
       LUAD       0.95      1.00      0.98        40
       PRAD       1.00      0.98      0.99        42

   accuracy                           0.99       241
  macro avg       0.99      0.99      0.99       241
weighted avg      0.99      0.99      0.99       241
```

Fig 5.21.1 XGB Classification Report

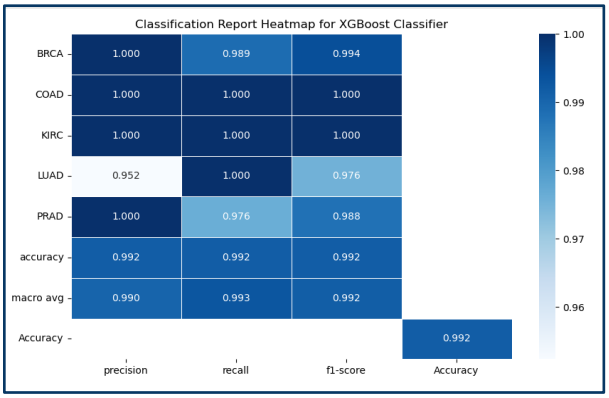


Fig 5.21.2 Heatmap for XGB

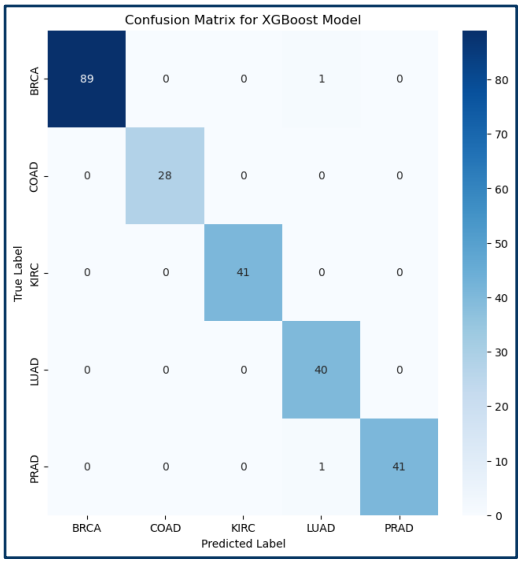


Fig 5.21.3 XGB Confusion matrix

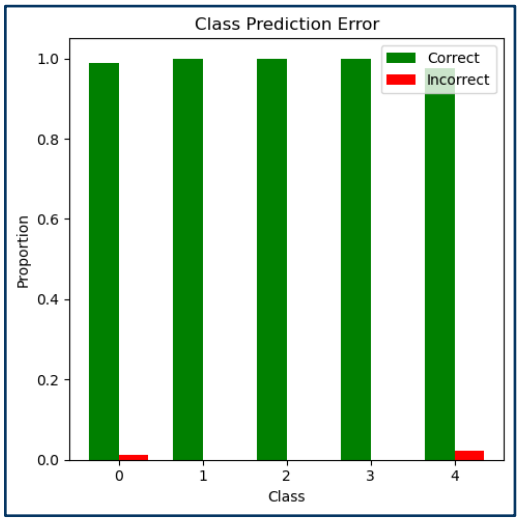


Fig 5.21.4 XGB Class Prediction Error

The XGBoost Classifier achieved a high accuracy of 99.38%. It made accurate predictions with a 99.05% precision rate. In addition, the model showed sensitivity, recalling 99.30% of real cases of each form of cancer. It balanced recall and precision with an F1-score of 99.16%. Exceptional performance was demonstrated in each section of the classification report, which included comprehensive measurements for every type of cancer. Accurate accuracy, recall, and F1-scores were shown for every type of cancer, as demonstrated by the heatmap, confusion matrix, and class prediction error plot visualizations. This further highlighted the model's sensitivity and reliability.

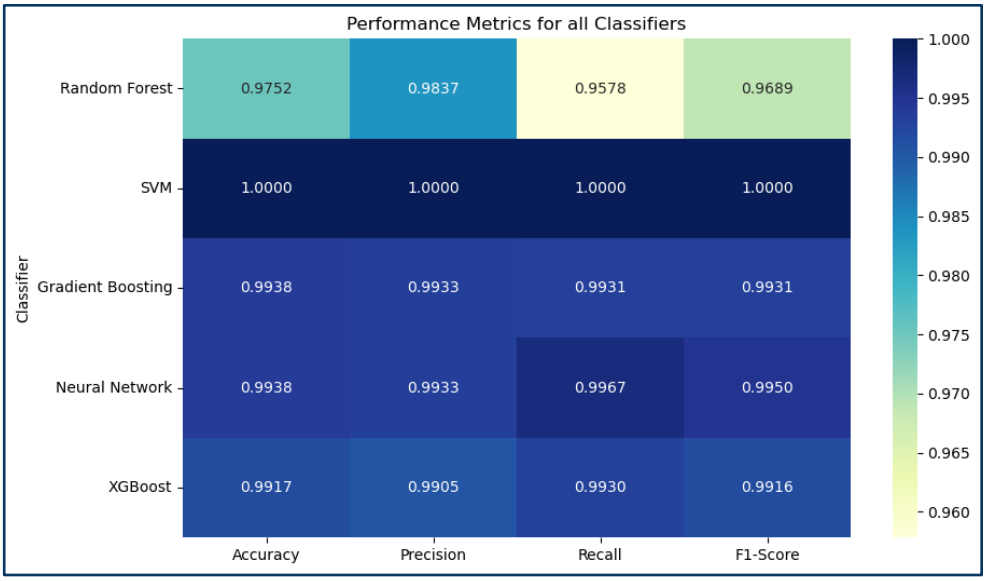


Fig 5.22 Comparison of Supervised learning Classifiers

The comparison of classifiers for cancer genomics projects is based on key metrics such as accuracy, precision, recall, and F1-score. Random Forest has the highest accuracy at 97.52%, followed by Support Vector Machines (SVM) at 100.00%, Gradient Boosted Decision Tree at 99.38%, and Neural Network at 99.38%. SVM's perfect score in all metrics suggests overfitting, which is rare in real-world tasks.

Neural Network and Gradient Boosted Decision Tree show high accuracy, precision, recall, and F1-score, making them strong contenders. XGBoost also performs well, though slightly lower in some metrics compared to Neural Network and Gradient Boosting. Random Forest has the lowest scores but still performs well overall.

6. FUTURE WORK AND CONCLUSION

Conclusion

By applying advanced machine learning algorithms, this research on cancer genomics provides new possibilities for studying the genetic components of the disease. By using various classifiers, including Random Forest, Support Vector Machines, Neural Networks, and others, we have effectively navigated the complex genomic patterns associated with various types of cancer. The accuracy with which these malignancies were classified shows a significant advancement toward customized treatment plans and highlights the collaboration between biological research and machine analytics.

Our model's outstanding performance, particularly in terms of accuracy and prediction reliability, indicates an exciting revolution in the way we manage and identify cancer. This project represents an important step in the use of computational methods to address complex biological problems in addition to increasing our genetic understanding of cancers.

Future Work

The goal of the research is to increase the understanding of the biology of cancer and predictive accuracy by integrating several omics data layers, such as genomics and genes. Through ongoing studies, it also investigates the temporal element of cancer growth, offering insights into the evolution of illness and therapy responses. The interpretation of cancer genomic data can be improved with the application of cutting-edge deep learning techniques. Precision medicine in oncology can be improved by using the learned insights to create tailored treatment plans and clinical trials. Finding universal indicators of cancer is the main goal of the project, which also addresses ethical and data security issues. The models will be adapted and made relevant in a

variety of healthcare environments by scaling them for broad practical application. Transforming computational results into clinical and biological contexts will require interdisciplinary interactions with biologists.

REFERENCES

[1] Berger, A. C., Korkut, A., Kanchi, R., & Bowman, C. (2018). Pan-cancer analysis of whole exomes reveals recurrent somatic mutations in chromatin remodelers and their associations with poor prognosis. Nature genetics, 50(1), 122-128.

[2] Ciriello, G., et al. (2013). Mutational landscapes of human cancers revealed from 827 genomic sequencing projects. Nature, 493(7433), 319-324.

[3] Hoadley, K. A., et al. (2018). Multiplatform analysis reveals common drivers of cancer progression and metastasis. Cancer cell, 34(1), 1-15.

[4] Network, T. C. G. A. R. (2013). The Cancer Genome Atlas Pan-Cancer analysis project. Nature genetics, 45(11), 1113-1120.

[5] The Cancer Genome Atlas Research Network. (2013). The Cancer Genome Atlas Pan-Cancer analysis project. Nature genetics, 45(11), 1113-1120.

[6] Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5-32.

[7] Cortes, C., & Vapnik, V. (1995). Support-vector networks. Machine Learning, 20(3), 273-297.

[8] Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. Annals of Statistics, 29(5), 1189-1232.

[9] Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. Science, 313(5786), 504-507.

[10] Chen, T., & Guestrin, C. (2016, August). XGBoost: A scalable tree boosting system. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 785-794).

[11] Zsofia, K., et al. (2010). Genome-wide association studies of cancer. *Journal of Clinical Oncology, 27,* 4255–4267.

[12] McClellan, K. A., et al. (2013). Personalized medicine and access to health care: potential for inequitable access? *European Journal of Human Genetics, 21,* 143–147.

[13] Khera, A. V., et al. (2018). Genome-wide polygenic scores for common diseases identify individuals with risk equivalent to monogenic mutations. *Nature Genetics, 50,* 1219–1224.

[14] Zhang, Y. D., et al. (2020). Assessment of polygenic architecture and risk prediction based on common variants across fourteen cancers. *Nature Communications, 11,* 3353.

[15] Gradient Boosted Trees. In scikit-learn: Machine Learning in Python. Retrieved from https://scikit-learn.org/stable/modules/ensemble.html#gradient-boosted-trees

[16] Random Forests and Other Randomized Tree Ensembles. In scikit-learn: Machine Learning in Python. Retrieved from https://scikit-learn.org/stable/modules/ensemble.html#random-forests-and-other-randomized-tree-ensembles

[17] K-Means. In scikit-learn: Machine Learning in Python. Retrieved from https://scikit-learn.org/stable/modules/clustering.html#k-means

[18] Support Vector Machines for Classification. In scikit-learn: Machine Learning in Python. Retrieved from https://scikit-learn.org/stable/modules/svm.html#classification

[19] Feature Selection. In scikit-learn: Machine Learning in Python. Retrieved from https://scikit-learn.org/stable/modules/feature_selection.html

[20] Neural Network Models (Supervised). In scikit-learn: Machine Learning in Python. Retrieved from https://scikit-learn.org/stable/modules/neural_networks_supervised.html#classification

APPENDIX

```
# In[2]:
#Import the library
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing
# In[3]:

dataset=pd.read_csv("data.csv")
dataset
# In[7]:
print(dataset.dtypes)
# In[9]:
# Removing the Unnamed columns from dataset
dataset.drop(dataset.columns[dataset.columns.str.contains('Unnamed',case =
False)],axis = 1, inplace = True)
# In[10]:
dataset
# In[6]:
labelset=pd.read_csv("labels.csv")
labelset
# In[8]:
print(labelset.dtypes)
# In[12]:
# Check for missing values in the data
missing_values_data = dataset.isnull().sum().sum()
missing_values_labels = labelset.isnull().sum().sum()
print(f"Missing values in data: {missing_values_data}")
print(f"Missing values in labels: {missing_values_labels}")
# In[13]:
# Descriptive statistics for the data
data_description = dataset.describe()
# Distribution of classes in the labels
class_distribution = labelset['Class'].value_counts()
print(data_description)
print(class_distribution)
# In[14]:
# Generate a heatmap for the gene expression data
import seaborn as sns
heatmap_data = dataset.iloc[:, 1:25]  # Using the first 25 genes for the
heatmap
plt.figure(figsize=(14, 8))
sns.heatmap(heatmap_data.corr(), annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Heatmap for the Gene Expression Data')
```

```
plt.show()
# Generate a heatmap for the class distribution in the labels

labels_encoded = pd.get_dummies(labelset['Class'])
plt.figure(figsize=(14, 8))
sns.heatmap(labels_encoded.corr(), annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Heatmap for the Labels Data')
plt.show()
# In[15]:
# Assigning the modified dataset to the varaible 'x'.
X = dataset
X
# In[20]:
# Calculate the correlation matrix for batches of 100 genes
batch_size = 100
num_batches = dataset.shape[1] // batch_size + (1 if dataset.shape[1] %
batch_size else 0)
# Initialize a list to store the correlation matrices for each batch
batch_correlation_matrices_no_def = []
# Compute the correlation matrix for each batch
for batch_num in range(num_batches):
    start_index = batch_num * batch_size
    end_index = min(start_index + batch_size, dataset.shape[1])
    batch_data = dataset.iloc[:, start_index:end_index]
    batch_correlation = batch_data.corr()
    batch_correlation_matrices_no_def.append(batch_correlation)
# Now, let's plot the heatmap for the correlation matrix of the first batch
plt.figure(figsize=(10,10))
sns.heatmap(batch_correlation_matrices_no_def[0], cmap='coolwarm')
plt.title('Correlation Matrix Heatmap for the First Batch of 100 Genes')
plt.show()
# In[21]:
#Using StandardScaler for normalization
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled
# In[22]:
#Applying Principal Component Analysis
#Create a PCA object that will retain 95 percentage of the variance
pca = PCA(n_components=0.95)
X_pca = pca.fit_transform(X_scaled)
#X_pca.shape
# In[23]:
print("Original number of features:",X.shape[1])
print("Reduced number of features:",X_pca.shape[1])
# In[24]:
#Percentage of variance explained by each of the selected components.
pca.explained_variance_ratio_
# In[26]:
```

```
#The estimated number of components
pca.n_components_
# In[27]:
#Important features information or features extractions or Selected features.
X_pca
# Conduct the PCA with 2 components.
pca = PCA(2)
X_pca = pca.fit_transform(X_scaled)
X_pca.shape
# In[29]:
pca.explained_variance_ratio_
# In[30]:
# Concating the target label data with 2 principal components
df=pd.read_csv("labels.csv")
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(X_scaled)
principalDf = pd.DataFrame(data = principalComponents, columns = ['PC1',
'PC2'])
finalDf = pd.concat([principalDf, df[['Class']]], axis = 1)
finalDf
# In[31]:
# Visualizing the scatter plot with the 2 PCA and target labels.
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)
targets = ['BRCA', 'COAD', 'KIRC','LUAD','PRAD']
colors = ['r', 'g', 'b','c','y']
for target, color in zip(targets,colors):
    indicesToKeep = finalDf['Class'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'PC1'], finalDf.loc[indicesToKeep,
'PC2'], c=color, s = 50)
ax.legend(targets)
ax.grid()
#PCA for 3 components
pca = PCA(n_components=3)
principalComponents = pca.fit_transform(X_scaled)
explained_variance = pca.explained_variance_ratio_
principalDf2 = pd.DataFrame(data = principalComponents, columns = ['PC1',
'PC2','PC3'])
finalDf2 = pd.concat([principalDf2, df[['Class']]], axis = 1)
finalDf2
# Expected Output:
print("Explained variance by the first 3 principal components:",
explained_variance)
# In[23]:
# 3D Visualization
fig = plt.figure(figsize=(15, 15))
```

```
ax = fig.add_subplot(111, projection='3d')
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_zlabel('Principal Component 3', fontsize = 15)
ax.set_title('3 components PCA', fontsize = 20)
targets = ['BRCA', 'COAD', 'KIRC','LUAD','PRAD']
colors = ['r', 'g', 'b','c','y']
for target, color in zip(targets,colors):
    indicesToKeep = finalDf2['Class'] == target
    ax.scatter(finalDf2.loc[indicesToKeep, 'PC1'],
finalDf2.loc[indicesToKeep, 'PC2'], finalDf2.loc[indicesToKeep, 'PC3'],
c=color, s = 50)
ax.legend(targets)
ax.grid()
# In[32]:
from sklearn.cluster import KMeans


# In[33]:
kmeans = KMeans(n_clusters=5, init="k-means++", n_init=50, max_iter=500,
random_state=42)
# Fit the KMeans algorithm to the PCA-transformed data
kmeans.fit(X_pca)
clusters = kmeans.labels_
# In[37]:
plt.figure(figsize=(8, 8))
# Unique clusters
unique_clusters = np.unique(clusters)
# Loop through all unique clusters
for cluster in unique_clusters:
    # Mask to separate sets of data
    mask = clusters == cluster
    plt.scatter(X_pca[mask, 0], X_pca[mask, 1], s=50, label=f"Cluster
{cluster}", cmap='plasma')
# Plot centroids
centroids = kmeans.cluster_centers_
plt.scatter(centroids[:, 0], centroids[:, 1], c='black', marker='X', s=50,
label="Centroids")
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('KMeans Clustering Results')
plt.legend(bbox_to_anchor=(1, 1), loc=2)
plt.grid()
plt.show()


# In[38]:
from sklearn.metrics import silhouette_samples, silhouette_score
import matplotlib.cm as cm
```

```
range_n_clusters = [2, 3, 4, 5, 6]  # Example range
for n_clusters in range_n_clusters:
    # Create a subplot with 1 row and 2 columns
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18, 7)
    # The 1st subplot is the silhouette plot
    ax1.set_xlim([-0.1, 1])
    ax1.set_ylim([0, len(X_pca) + (n_clusters + 1) * 10])

    # Initialize the clusterer with n_clusters value and a random generator
seed for reproducibility
    clusterer = KMeans(n_clusters=n_clusters, random_state=10)
    cluster_labels = clusterer.fit_predict(X_pca)

    # The silhouette_score gives the average value for all the samples
    silhouette_avg = silhouette_score(X_pca, cluster_labels)
    print(f"For n_clusters = {n_clusters}, the average silhouette_score is :
{silhouette_avg}")

    # Compute the silhouette scores for each sample
    sample_silhouette_values = silhouette_samples(X_pca, cluster_labels)

    y_lower = 10
    for i in range(n_clusters):
        # Aggregate the silhouette scores for samples belonging to cluster i,
and sort them
        ith_cluster_silhouette_values =
sample_silhouette_values[cluster_labels == i]
        ith_cluster_silhouette_values.sort()

        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = cm.nipy_spectral(float(i) / n_clusters)
        ax1.fill_betweenx(np.arange(y_lower, y_upper), 0,
ith_cluster_silhouette_values, facecolor=color, edgecolor=color, alpha=0.7)

        y_lower = y_upper + 10  # 10 for the 0 samples

    ax1.set_title("The silhouette plot for the various clusters.")
    ax1.set_xlabel("The silhouette coefficient values")
    ax1.set_ylabel("Cluster label")

    # The 2nd subplot showing the actual clusters formed
    colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
    ax2.scatter(X_pca[:, 0], X_pca[:, 1], marker='.', s=30, lw=0, alpha=0.7,
c=colors, edgecolor='k')

    # Labeling the clusters
```

```
    centers = clusterer.cluster_centers_
    for i, c in enumerate(centers):
        ax2.scatter(c[0], c[1], marker='$%d$' % i, alpha=1, s=50,
edgecolor='k')

    ax2.set_title("The visualization of the clustered data.")
    ax2.set_xlabel("Feature space for the 1st feature")
    ax2.set_ylabel("Feature space for the 2nd feature")

    plt.suptitle((f"Silhouette analysis for KMeans clustering on sample data
with n_clusters = {n_clusters}"),
                 fontsize=14, fontweight='bold')
plt.show()
# In[39]:
Y=df['Class']
# In[47]:
from sklearn.model_selection import train_test_split
#The data set into training and validation sets. 70:30
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,
random_state=42)

# In[48]:
pca = PCA(n_components=0.95)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

# In[49]
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score, precision_score, accuracy_score,
recall_score,confusion_matrix
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import classification_report
rfc_clf = RandomForestClassifier()
rfc_clf.fit(X_train_pca, y_train)
y_pred = rfc_clf.predict(X_test_pca)
target_names =['BRCA', 'COAD', 'KIRC','LUAD','PRAD']
rf_accuracy = accuracy_score(y_test, y_pred)
rf_precision = precision_score(y_test, y_pred, average="macro")
rf_recall = recall_score(y_test, y_pred, average="macro")
rf_fscore = f1_score(y_test, y_pred, average="macro")

print(f"Random Forest Metrics:")
print(f"Accuracy: {rf_accuracy*100:.2f}%")
print(f"Precision (macro avg): {rf_precision*100:.2f}%")
print(f"Recall (macro avg): {rf_recall*100:.2f}%")
print(f"F1-score (macro avg): {rf_fscore*100:.2f}%")
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=target_names))
# In[50]:
```

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
import pandas as pd
# Assuming y_test and y_pred are defined and contain the true and predicted
labels
report = classification_report(y_test, y_pred, target_names=target_names,
output_dict=True)
report_df = pd.DataFrame(report).transpose()
# Prepare the DataFrame for heatmap
heatmap_data = report_df.drop(columns="support").iloc[:-1, :]
# Extract accuracy and convert it into a DataFrame
accuracy_data = pd.DataFrame({'Accuracy': report['accuracy']},
index=["Accuracy"])
# Combine heatmap data and accuracy data
combined_data = pd.concat([heatmap_data, accuracy_data])
# Create the heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(combined_data, annot=True, cmap="Blues", fmt=".3f",
linewidths=.5)
plt.title("Classification Report Heatmap for Random Forest Classifier")
plt.yticks(rotation=0)  # Keep the class names horizontal
plt.show()
# In[51]:
# Compute the confusion matrix
conf_matrix = confusion_matrix(y_true=y_test, y_pred=y_pred)
fig, ax = plt.subplots(figsize=(6, 6))
# Display the confusion matrix using imshow
ax.imshow(conf_matrix,  cmap=plt.cm.Greys)
# Add title and labels
ax.set_title('Random Forest Confusion Matrix', fontsize=18)
tick_marks = np.arange(len(target_names))
ax.set_xticks(tick_marks)
ax.set_yticks(tick_marks)
ax.set_xticklabels(target_names, fontsize=12)
ax.set_yticklabels(target_names, fontsize=12)
plt.xlabel('Predicted Label', fontsize=15)
plt.ylabel('Actual Label', fontsize=15)
# Rotate the tick labels for better visibility
plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
rotation_mode="anchor")
plt.setp(ax.get_yticklabels(), rotation=45, ha="right",
rotation_mode="anchor")
# Loop over data dimensions and create text annotations to display the actual
numbers
thresh = conf_matrix.max() / 2.
for i in range(conf_matrix.shape[0]):
    for j in range(conf_matrix.shape[1]):
```

```
        ax.text(j, i, format(conf_matrix[i, j], 'd'), ha="center",
va="center",
                color="white" if conf_matrix[i, j] > thresh else "black",
fontsize=12)
plt.tight_layout()
plt.show()
# In[52]:
# Plotting Class Prediction Error based on the confusion matrix
y_pred = rfc_clf.predict(X_test_pca)  # Replace with your model and test data
if needed
# Compute the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
classes = np.unique(y_test)
# Calculate the percentage of total for each class
class_totals = conf_matrix.sum(axis=1)
class_correct = np.diag(conf_matrix)
class_error = class_totals - class_correct
# Plot
fig, ax = plt.subplots(figsize=(5, 5))
indices = np.arange(len(classes))
bar_width = 0.35
ax.bar(indices, class_correct / class_totals, bar_width, label='Correct',
color='Green')
ax.bar(indices + bar_width, class_error / class_totals, bar_width,
label='Incorrect', color='red')

# Labeling
ax.set_xlabel('Class')
ax.set_ylabel('Proportion')
ax.set_title('Class Prediction Error')
ax.set_xticks(indices + bar_width / 2)
ax.set_xticklabels(classes)
ax.legend()
plt.tight_layout()
plt.show()
# In[53]:
from sklearn.svm import SVC
from sklearn.metrics import f1_score, precision_score, accuracy_score,
recall_score,confusion_matrix
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import classification_report
svm_clf = SVC()
svm_clf.fit(X_train_pca, y_train)
y_pred = svm_clf.predict(X_test_pca)
target_names =['BRCA', 'COAD', 'KIRC','LUAD','PRAD']
svm_accuracy = accuracy_score(y_test, y_pred)
svm_precision = precision_score(y_test, y_pred, average="macro")
svm_recall = recall_score(y_test, y_pred, average="macro")
svm_fscore = f1_score(y_test, y_pred, average="macro")
```

```
print(f"Support Vector Machines Metrics:")
print(f"Accuracy: {svm_accuracy*100:.2f}%")
print(f"Precision (macro avg): {svm_precision*100:.2f}%")
print(f"Recall (macro avg): {svm_recall*100:.2f}%")
print(f"F1-score (macro avg): {svm_fscore*100:.2f}%")
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=target_names))
# In[54]:
# Assuming y_test and y_pred are defined and contain the true and predicted
labels
report = classification_report(y_test, y_pred, target_names=target_names,
output_dict=True)
report_df = pd.DataFrame(report).transpose()
# Prepare the DataFrame for heatmap
# Exclude the 'support' column and the last row (which contains average/total
metrics)
heatmap_data = report_df.drop(columns="support").iloc[:-1, :]
# Extract accuracy and convert it into a DataFrame
accuracy_data = pd.DataFrame({'Accuracy': report['accuracy']},
index=["Accuracy"])
# Combine heatmap data and accuracy data
combined_data = pd.concat([heatmap_data, accuracy_data])
# Create the heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(combined_data, annot=True, cmap="Blues", fmt=".3f",
linewidths=.5)
plt.title("Classification Report Heatmap for SVM Classifier")
plt.yticks(rotation=0)  # Keep the class names horizontal
plt.show()
# In[55]:
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
fig, ax = plt.subplots(figsize=(6, 6))
ax.imshow(cm,  cmap=plt.cm.Blues)
# Plot the confusion matrix
ax.set_title('SVM Confusion Matrix', fontsize=18)
tick_marks = np.arange(len(target_names))
ax.set_xticks(tick_marks)
ax.set_yticks(tick_marks)
ax.set_xticklabels(target_names, fontsize=12)
ax.set_yticklabels(target_names, fontsize=12)
plt.xlabel('Predicted Label', fontsize=15)
plt.ylabel('Actual Label', fontsize=15)
# Rotate the tick labels for better visibility
plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
rotation_mode="anchor")
```

```
plt.setp(ax.get_yticklabels(), rotation=45, ha="right",
rotation_mode="anchor")
# Loop over data dimensions and create text annotations to display the actual
numbers
thresh = cm.max() / 2.
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        ax.text(j, i, format(cm[i, j], 'd'), ha="center", va="center",
                color="white" if cm[i, j] > thresh else "black", fontsize=12)
plt.tight_layout()
plt.show()
# In[57]:
# Assuming y_test are true labels and y_pred are the predictions from SVM
y_pred = svm_clf.predict(X_test_pca)
# Compute the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
classes = np.unique(y_test)
# Calculate the percentage of total for each class
class_totals = conf_matrix.sum(axis=1)
class_correct = np.diag(conf_matrix)
class_error = class_totals - class_correct
# Plot
fig, ax = plt.subplots(figsize=(5, 5))
indices = np.arange(len(classes))
bar_width = 0.35
ax.bar(indices, class_correct / class_totals, bar_width, label='Correct',
color='Green')
ax.bar(indices + bar_width, class_error / class_totals, bar_width,
label='Incorrect', color='red')
# Labeling
ax.set_xlabel('Class')
ax.set_ylabel('Proportion')
ax.set_title('Class Prediction Error')
ax.set_xticks(indices + bar_width / 2)
ax.set_xticklabels(classes)
ax.legend()

plt.tight_layout()
plt.show()
# In[58]:
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import f1_score, precision_score, accuracy_score,
recall_score, confusion_matrix
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import classification_report
# Initialize the Gradient Boosting Classifier
gb_clf = GradientBoostingClassifier()
# Fit the model
gb_clf.fit(X_train_pca, y_train)
```

```
# Make predictions
y_pred = gb_clf.predict(X_test_pca)
# Define the target names
target_names = ['BRCA', 'COAD', 'KIRC', 'LUAD', 'PRAD']
# Calculate metrics
gb_accuracy = accuracy_score(y_test, y_pred)
gb_precision = precision_score(y_test, y_pred, average="macro")
gb_recall = recall_score(y_test, y_pred, average="macro")
gb_fscore = f1_score(y_test, y_pred, average="macro")
# Print metrics
print(f"Gradient Boosted Decision Tree Metrics:")
print(f"Accuracy: {gb_accuracy*100:.2f}%")
print(f"Precision (macro avg): {gb_precision*100:.2f}%")
print(f"Recall (macro avg): {gb_recall*100:.2f}%")
print(f"F1-score (macro avg): {gb_fscore*100:.2f}%")
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=target_names))

report = classification_report(y_test, y_pred, target_names=target_names,
output_dict=True)
report_df = pd.DataFrame(report).transpose()
# Prepare the DataFrame for heatmap
# Exclude the 'support' column and the last row (which contains average/total
metrics)
heatmap_data = report_df.drop(columns="support").iloc[:-1, :]
# Extract accuracy and convert it into a DataFrame
accuracy_data = pd.DataFrame({'Accuracy': report['accuracy']},
index=["Accuracy"])
# Combine heatmap data and accuracy data
combined_data = pd.concat([heatmap_data, accuracy_data])
# Create the heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(combined_data, annot=True, cmap="Blues", fmt=".3f",
linewidths=.5)
plt.title("Classification Report Heatmap for Gradient Boosted Decision Tree
Classifier")
plt.yticks(rotation=0)  # Keep the class names horizontal
plt.show()

import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
cgb = confusion_matrix(y_test, y_pred)
fig, ax = plt.subplots(figsize=(6, 6))
ax.imshow(cgb,  cmap=plt.cm.Greens)
# Plot the confusion matrix
ax.set_title('Gradient Boost Confusion Matrix', fontsize=18)
tick_marks = np.arange(len(target_names))
ax.set_xticks(tick_marks)
```

```
ax.set_yticks(tick_marks)
ax.set_xticklabels(target_names, fontsize=12)
ax.set_yticklabels(target_names, fontsize=12)
plt.xlabel('Predicted Label', fontsize=15)
plt.ylabel('Actual Label', fontsize=15)
# Rotate the tick labels for better visibility
plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
rotation_mode="anchor")
plt.setp(ax.get_yticklabels(), rotation=45, ha="right",
rotation_mode="anchor")
# Loop over data dimensions and create text annotations to display the actual
numbers
thresh = cgb.max() / 2.
for i in range(cgb.shape[0]):
    for j in range(cgb.shape[1]):
        ax.text(j, i, format(cgb[i, j], 'd'), ha="center", va="center",
                color="white" if cgb[i, j] > thresh else "black", fontsize=12)
plt.tight_layout()
plt.show()


# In[63]:
# Plotting Class Prediction Error based on the confusion matrix
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import numpy as np
# Assuming y_test are true labels and y_pred are the predictions from
Gradient Boost
y_pred = gb_clf.predict(X_test_pca)
# Compute the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
classes = np.unique(y_test)
# Calculate the percentage of total for each class
class_totals = conf_matrix.sum(axis=1)
class_correct = np.diag(conf_matrix)
class_error = class_totals - class_correct
# Plot
fig, ax = plt.subplots(figsize=(5,5))
indices = np.arange(len(classes))
bar_width = 0.35
ax.bar(indices, class_correct / class_totals, bar_width, label='Correct',
color='Green')
ax.bar(indices + bar_width, class_error / class_totals, bar_width,
label='Incorrect', color='red')
# Labeling
ax.set_xlabel('Class')
ax.set_ylabel('Proportion')
ax.set_title('Class Prediction Error')
ax.set_xticks(indices + bar_width / 2)
```

```
ax.set_xticklabels(classes)
ax.legend()

plt.tight_layout()
plt.show()
# In[43]:
#Neural Networks
# In[64]:
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import f1_score, precision_score, accuracy_score,
recall_score, confusion_matrix, classification_report
from mlxtend.plotting import plot_confusion_matrix
import matplotlib.pyplot as plt
# Initialize the Neural Network classifier
# using one hidden layer with 100 neurons
nn_clf = MLPClassifier(hidden_layer_sizes=(100,), max_iter=300,
activation='relu', solver='adam', random_state=1)
# Train the model
nn_clf.fit(X_train_pca, y_train)
# Predict on the test set
y_pred = nn_clf.predict(X_test_pca)
target_names = ['BRCA', 'COAD', 'KIRC', 'LUAD', 'PRAD']
# Compute the metrics
nn_accuracy = accuracy_score(y_test, y_pred)
nn_precision = precision_score(y_test, y_pred, average="macro")
nn_recall = recall_score(y_test, y_pred, average="macro")
nn_fscore = f1_score(y_test, y_pred, average="macro")

# Print the metrics
print(f"Neural Network Metrics:")
print(f"Accuracy: {nn_accuracy*100:.2f}%")
print(f"Precision (macro avg): {nn_precision*100:.2f}%")
print(f"Recall (macro avg): {nn_recall*100:.2f}%")
print(f"F1-score (macro avg): {nn_fscore*100:.2f}%")
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=target_names))
# In[65]:
# Assuming y_test and y_pred are defined and contain the true and predicted
labels
report = classification_report(y_test, y_pred, target_names=target_names,
output_dict=True)
report_df = pd.DataFrame(report).transpose()
# Prepare the DataFrame for heatmap
# Exclude the 'support' column and the last row (which contains average/total
metrics)
heatmap_data = report_df.drop(columns="support").iloc[:-1, :]
# Extract accuracy and convert it into a DataFrame
accuracy_data = pd.DataFrame({'Accuracy': report['accuracy']},
index=["Accuracy"])
```

```
# Combine heatmap data and accuracy data
combined_data = pd.concat([heatmap_data, accuracy_data])
# Create the heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(combined_data, annot=True, cmap="Blues", fmt=".3f",
linewidths=.5)
plt.title("Classification Report Heatmap for Neural Network Classifier")
plt.yticks(rotation=0)  # Keep the class names horizontal
plt.show()
# In[66]:
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
cnn = confusion_matrix(y_test, y_pred)
fig, ax = plt.subplots(figsize=(6, 6))
ax.imshow(cnn,  cmap=plt.cm.Greens)
# Plot the confusion matrix
ax.set_title('Neural Network Confusion Matrix', fontsize=18)
tick_marks = np.arange(len(target_names))
ax.set_xticks(tick_marks)
ax.set_yticks(tick_marks)
ax.set_xticklabels(target_names, fontsize=12)
ax.set_yticklabels(target_names, fontsize=12)
plt.xlabel('Predicted Label', fontsize=15)
plt.ylabel('Actual Label', fontsize=15)
# Rotate the tick labels for better visibility
plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
rotation_mode="anchor")
plt.setp(ax.get_yticklabels(), rotation=45, ha="right",
rotation_mode="anchor")
# Loop over data dimensions and create text annotations to display the actual
numbers
thresh = cnn.max() / 2.
for i in range(cnn.shape[0]):
    for j in range(cnn.shape[1]):
        ax.text(j, i, format(cnn[i, j], 'd'), ha="center", va="center",
                color="white" if cnn[i, j] > thresh else "black",
fontsize=12)
plt.tight_layout()
plt.show()

# In[67]:
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import numpy as np
y_pred = nn_clf.predict(X_test_pca)
# Compute the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
classes = np.unique(y_test)
```

```
# Calculate the percentage of total for each class
class_totals = conf_matrix.sum(axis=1)
class_correct = np.diag(conf_matrix)
class_error = class_totals - class_correct
# Plot
fig, ax = plt.subplots(figsize=(5, 5))
indices = np.arange(len(classes))
bar_width = 0.35
ax.bar(indices, class_correct / class_totals, bar_width, label='Correct',
color='Green')
ax.bar(indices + bar_width, class_error / class_totals, bar_width,
label='Incorrect', color='red')

# Labeling
ax.set_xlabel('Class')
ax.set_ylabel('Proportion')
ax.set_title('Class Prediction Error')
ax.set_xticks(indices + bar_width / 2)
ax.set_xticklabels(classes)
ax.legend()
plt.tight_layout()
plt.show()

# In[69]:

# xgboost

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import xgboost as xgb
from sklearn.metrics import accuracy_score, classification_report

# Transform labels to numerical format
label_encoder = LabelEncoder()

y_encoded = label_encoder.fit_transform(Y)

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded,
test_size=0.3, random_state=42)

X_train_pca = pca.fit_transform(X_train)

X_test_pca = pca.transform(X_test)

# Initialize XGBoost classifier
xgb_clf = xgb.XGBClassifier(objective='multi:softprob',
num_class=len(set(y_encoded)), seed=42)
```

```
# Fit the classifier to the training data
xgb_clf.fit(X_train_pca, y_train)

# Predict on the test data
y_pred = xgb_clf.predict(X_test_pca)

# Compute the metrics
xgb_accuracy = accuracy_score(y_test, y_pred)
xgb_precision = precision_score(y_test, y_pred, average="macro")
xgb_recall = recall_score(y_test, y_pred, average="macro")
xgb_fscore = f1_score(y_test, y_pred, average="macro")

# Print the metrics
print(f"XGBoost classifier Metrics:")
print(f"Accuracy: {xgb_accuracy*100:.2f}%")
print(f"Precision (macro avg): {xgb_precision*100:.2f}%")
print(f"Recall (macro avg): {xgb_recall*100:.2f}%")
print(f"F1-score (macro avg): {xgb_fscore*100:.2f}%")
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=target_names))

# In[70]:

import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
import pandas as pd

report = classification_report(y_test, y_pred, target_names=target_names,
output_dict=True)
report_df = pd.DataFrame(report).transpose()

# Prepare the DataFrame for heatmap
heatmap_data = report_df.drop(columns="support").iloc[:-1, :]

# Extract accuracy and convert it into a DataFrame
accuracy_data = pd.DataFrame({'Accuracy': report['accuracy']},
index=["Accuracy"])

# Combine heatmap data and accuracy data
combined_data = pd.concat([heatmap_data, accuracy_data])

# Create the heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(combined_data, annot=True, cmap="Blues", fmt=".3f",
linewidths=.5)
plt.title("Classification Report Heatmap for XGBoost Classifier")
plt.yticks(rotation=0)  # Keep the class names horizontal
plt.show()
```

```
# In[71]:

from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Compute the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix using a heatmap
plt.figure(figsize=(8, 8))
sns.heatmap(conf_matrix, annot=True, fmt='g', cmap='Blues',
xticklabels=label_encoder.classes_, yticklabels=label_encoder.classes_)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for XGBoost Model')
plt.show()

# Assuming y_test aretrue labels and y_pred are the predictions from XGBoost
y_pred = xgb_clf.predict(X_test_pca)

# Compute the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
classes = np.unique(y_test)

# Calculate the percentage of total for each class
class_totals = conf_matrix.sum(axis=1)
class_correct = np.diag(conf_matrix)
class_error = class_totals - class_correct

# Plot
fig, ax = plt.subplots(figsize=(5, 5))
indices = np.arange(len(classes))
bar_width = 0.35

ax.bar(indices, class_correct / class_totals, bar_width, label='Correct',
color='Green')
ax.bar(indices + bar_width, class_error / class_totals, bar_width,
label='Incorrect', color='red')

# Labeling
ax.set_xlabel('Class')
ax.set_ylabel('Proportion')
ax.set_title('Class Prediction Error')
ax.set_xticks(indices + bar_width / 2)
ax.set_xticklabels(classes)
ax.legend()
```

```
plt.tight_layout()
plt.show()

import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

data = {
     "Classifier": ["Random Forest", "SVM", "Gradient Boosting", "Neural
Network", "XGBoost"],
    "Accuracy": [rf_accuracy, svm_accuracy, gb_accuracy, nn_accuracy,
xgb_accuracy],
    "Precision": [rf_precision, svm_precision, gb_precision, nn_precision,
xgb_precision],
    "Recall": [rf_recall, svm_recall, gb_recall, nn_recall, xgb_recall],
    "F1-Score": [rf_fscore, svm_fscore, gb_fscore, nn_fscore, xgb_fscore]
}
df = pd.DataFrame(data)
df.set_index('Classifier', inplace=True)
plt.figure(figsize=(10, 6))
sns.heatmap(df, annot=True, cmap="YlGnBu", fmt=".4f")
plt.title("Performance Metrics for all Classifiers")
plt.show()
```