Cancer Type Prediction and Exploration of TCGA Data

\_\_\_\_\_\_\_\_\_\_\_\_

A Project

Presented

to the Faculty of

California State University, Dominguez Hills

\_\_\_\_\_\_\_\_\_\_\_\_

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

\_\_\_\_\_\_\_\_\_\_\_\_

by

Vishnugupthaa Ramidi

Spring 2024

PROJECT: CANCER TYPE PREDICTION AND EXPLORATION OF TCGA DATA

AUTHOR: VISHNUGUPTHAA RAMIDI

APPROVED:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Bin Tang, Ph.D.

Project Committee Chair

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Jianchao (Jack) Han, PhD.

Committee Member

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ryan Urbanowicz, Ph.D.

Cedar Sinai Medical Center

# **ABSTRACT:**

This study conducts a multi-class classification analysis of RNA-seq gene expression data to identify subtypes of five prevalent tumor types. Utilizing the TCGA Pan Cancer dataset, the analysis processes the data to reveal molecular signatures characteristic of each cancer subtype. A combination of unsupervised and supervised learning techniques is employed, encompassing data cleaning, clustering, feature selection, and model construction. The research uncovers latent data structures and accurately classifies cancer types, contributing insights for personalized treatment development. The findings highlight shared molecular pathways and unique characteristics across cancer lineages, offering significant implications for targeted therapies and prognostic evaluations.

**TABLE OF CONTENTS**

[**ABSTRACT:** 3](#_Toc165651860)

[**1.** **INTRODUCTION:** 7](#_Toc165651861)

[**2.** **LITERATURE REVIEW:** 8](#_Toc165651862)

[**3.** **SYSTEM DESIGN:** 10](#_Toc165651863)

[**4.** **DATASET:** 12](#_Toc165651864)

[**5.** **METHODOLOGY:** 14](#_Toc165651865)

[**6.** **EVALUATION AND ANALYSIS**: 34](#_Toc165651866)

[**7.** **CONCLUSION:** 36](#_Toc165651867)

[**8.** **FUTURE WORK:** 37](#_Toc165651868)

[**APPENDIX:** 38](#_Toc165651869)

[**REFERENCES:** 47](#_Toc165651870)

**LIST OF FIGURES**

[Figure 1 : Flowchart of System Design 10](#_Toc165554147)

[Figure 2: Distribution of Data 12](#_Toc165554148)

[Figure 3: Preliminary Exploratory Analysis & Histogram of first 25 gene Data 13](#_Toc165554149)

[Figure 4 : Sample out of the Normalized data 13](#_Toc165554150)

[Figure 5: Below is an Explained Variance of the above code snippet for PCA 15](#_Toc165554151)

[Figure 6 : Explained Variance for PCA with 2 Components 16](#_Toc165554152)

[Figure 7 : Silhouette analysis for K\_Means Clusters ranging from 2 – 6 18](#_Toc165554153)

[Figure 8: Visualized map of UMAP algorithm 20](#_Toc165554154)

[Figure 9: Explained UMAP with Davies Bouldin analysis and Hierarchical Cluster 21](#_Toc165554155)

[Figure 10: Output for the RandomForest Classifier model using PCA 24](#_Toc165554156)

[Figure 11: Output of XGBoost Classifier with PCA 27](#_Toc165554157)

[Figure 12: Output of Neural Network Model with PCA 29](#_Toc165554158)

[Figure 13: Output of GradientBoosting Classifier with PCA 32](#_Toc165554159)

[Figure 14: Comparative Analysis of the Performance Metrics 35](#_Toc165554160)

LIST OF TABLES

**Table 1: Table of Performance Metrics of Supervised models using UMAP ………………33**

**Table** **2: Table of Performance Metrics ……………………………………………………….35**

# **INTRODUCTION:**

Cancer, with its myriad forms and elusive nature, presents a labyrinth of challenges to healthcare and scientific inquiry. In the face of such complexity, the dissection of cancer subtypes at the molecular level stands as a critical objective within the field of oncology. This study leverages the expansive RNA-seq gene expression data to demystify the genetic variances that delineate five notable tumor types: Breast Invasive Carcinoma (BRCA), Kidney Renal Clear Cell Carcinoma (KIRC), Colon Adenocarcinoma (COAD), Lung Adenocarcinoma (LUAD), and Prostate Adenocarcinoma (PRAD)[1].

Anchored in the groundbreaking TCGA Pan Cancer analysis project and enriched by the UCI Machine Learning Repository, the dataset serves as a fountain of genomic knowledge. With rigorous data preparation and systematic exploration, this research seeks to illuminate the molecular signatures that are pivotal for the classification of cancer subtypes. A dual approach, integrating unsupervised learning for data discovery and supervised learning for targeted classification, is employed to reveal latent patterns, forge significant correlations, and identify key biomarkers for precise tumor identification.

The ramifications of this study extend well beyond the academic sphere, holding substantial promise for clinical application and the innovation of treatment strategies. By decoding the complex network of genetic mutations across various cancer types, this work aims to enhance prognostic accuracy, foster the creation of individualized treatment plans, and propel the field of precision medicine forward[1]. In essence, this research endeavors to transcend traditional categorization methods, deepening our comprehension of cancer's biological underpinnings and catalyzing the conversion of genomic discoveries into concrete health outcomes for individuals around the globe.

# **LITERATURE REVIEW:**

The landscape of cancer research has been profoundly transformed by technological advancements and collaborative scientific endeavors. Over recent decades, the concerted efforts of the global research community have yielded pivotal insights into the genetic and epigenetic mechanisms underpinning oncogenesis. Pioneering large-scale initiatives, notably the International Cancer Genomics Consortium (ICGC) and the Cancer Genome Atlas (TCGA)[1], have spearheaded the systematic characterization of a myriad of tumor types. These projects have amassed comprehensive datasets that span genomic, epigenomic, transcriptomic, and proteomic profiles, offering an unprecedented opportunity to discern patterns and variances across diverse cancer forms[1].

A cornerstone of these collaborative ventures is the quest to delineate the genetic catalysts of cancer's onset and progression. Researchers have meticulously sifted through vast arrays of data to distinguish driver mutations, which propel cancer cell proliferation, from passenger mutations, which are merely byproducts of cancer's evolution. This critical differentiation paves the way for the identification of therapeutic targets and a deeper understanding of cancer's genesis. Cross-tumor analyses have illuminated the nuanced interplay between genomic alterations and tissue-specific gene expression, revealing both shared oncogenic pathways and lineage-specific anomalies. These findings emphasize the necessity of considering tumor lineage in both research and clinical settings. The advent of multi-omics data integration has been a game-changer, offering novel perspectives on the complex molecular architecture of cancer. The synergy of genomics, transcriptomics, epigenomics, and proteomics has enriched our comprehension of cancer's molecular basis. Furthermore, the emergence of single-cell sequencing and advanced mass spectrometry techniques has facilitated an intricate examination of tumor heterogeneity and protein dynamics.

Despite these strides, challenges persist in cross-tumor analysis. Issues such as batch effects, variable mutation rates across tissues, and the experimental validation of computational predictions remain as hurdles to be overcome[2]. Moreover, translating these scientific discoveries into clinical applications demands the development of robust biomarkers and efficacious treatment modalities. Looking ahead, the ambition is to broaden the horizons of cross-tumor analysis, integrate additional data modalities, and enhance computational methods[2]. Through these endeavors, the research community strives to deepen our grasp of cancer biology and, ultimately, to forge pathways that will lead to improved patient outcomes.

# **SYSTEM DESIGN:**

The system architecture for Gene Expression Analysis and Comprehensive Tumor Classification can be envisioned as a methodical blueprint that navigates through the sequential stages of data processing and analytical model development[3].

**A diagram of a model

Description automatically generated**

Figure 1 : Flowchart of System Design

**1. Data Collection:** The journey begins with *data collection*, the cornerstone of our analysis. This phase entails the meticulous aggregation of gene expression data and corresponding tumor class labels, laying the groundwork for subsequent exploration.

**2. Data Analysis:** *Data analysis* follows, serving as the initial foray into the dataset. Here, we scrutinize the data's structure and attributes, employing visual tools like distribution graphs to identify patterns and outliers, setting the stage for deeper investigation.

**3. Dimensionality Reduction:** As we delve further, *dimensionality reduction* techniques such as Principal Component Analysis (PCA) come into play. PCA streamlines the dataset, distilling a vast array of variables into their most informative essences, thereby simplifying complexity without sacrificing critical information.

**4. Data Preparation:** *Data preparation* is pivotal, as we refine the data for model training. Normalization processes standardize the data, ensuring uniformity and comparability akin to harmonizing diverse dialects into a common language for clearer communication.

**5. Model Development:** With the data primed, we advance to *model development*. This stage is where theoretical concepts materialize into tangible models through the application of various classification algorithms, each selected for its suitability to the task at hand.

**6. Model Evaluation:** Upon training completion, *model evaluation* commences. We deploy metrics such as accuracy, precision, recall and F1-score to gauge our models' performance, drawing parallels to a meticulous vetting process for potential candidates.

**7. Model Selection:** *Model selection* is the culmination of evaluation efforts. We weigh each model's merits, seeking the one that stands out in efficiency and efficacy for our tumor classification endeavor, much like choosing the most qualified applicant from a pool of talent.

**8. Conclusion:** In conclusion, we reflect on the entire process from data collection to model selection and distill our findings into actionable insights. This allows us to ascertain the superior model for tumor classification and to derive meaningful interpretations from the gene expression data.

# **DATASET:**

The foundation of our study is built upon a robust collection of gene expression data, meticulously gathered from the TCGA Pan Cancer analysis project. This comprehensive dataset spans a diverse array of tumor types, providing a rich tapestry of genetic information crucial for our analysis. To complement this, we have incorporated additional datasets from the UCI Machine Learning Repository, which serve to enhance the breadth and depth of our training and evaluation phases. The amalgamation of these datasets ensures a comprehensive representation of the genomic landscape across various cancer subtypes.

A pie chart with different colored circles

Description automatically generated

Figure 2: Distribution of Data

* 1. **Data Preparation**

Prior to analysis, the data underwent a rigorous preprocessing regimen to ensure its integrity and uniformity. This included *data cleaning* to remove any inconsistencies or incomplete entries, *normalization* to standardize expression levels across different samples, and *merging* of datasets from disparate sources to create a cohesive pool of information. These steps were critical in establishing a harmonized dataset, thereby facilitating accurate and reliable downstream analysis. The prepared data sets the stage for the subsequent phases of dimensionality reduction, feature selection, and model development, ensuring that the insights gleaned are reflective of the underlying biological processes.

A screenshot of a computer code

Description automatically generated A blue and red data

Description automatically generated with medium confidence

Figure 3: Preliminary Exploratory Analysis & Histogram of first 25 gene Data

|  |
| --- |
| #Using StandardScalar module for Normalization  X = dataset  scaler = StandardScaler()  X\_scaled = scaler.fit\_transform(X)  X\_scaled |

Figure 4 : Sample out of the Normalized data

A screenshot of a computer

Description automatically generated

# **METHODOLOGY:**

The methodology for the Gene Expression Analysis and Comprehensive Tumor Classification project is a blend of unsupervised and supervised learning approaches designed to extract meaningful insights from complex gene expression data. This project employs a dual-faceted approach to gene expression analysis for comprehensive tumor classification, utilizing both supervised and unsupervised machine learning techniques to extract and analyze patterns within the data.

* 1. **UNSUPERVISED LEARNING APPROACH:**

1. **PRINCIPAL COMPONENT ANALYSIS:** Initially, PCA is applied to the gene expression data to reduce dimensionality while retaining the most informative aspects of the data[4]. This step simplifies the complex high-dimensional data, making it more manageable for subsequent analysis.

|  |
| --- |
| # Perform PCA to retain 95% of the variance  pca = PCA(n\_components=0.95)  X\_pca = pca.fit\_transform(X\_scaled)  # Plotting the explained variance  plt.figure(figsize=(8, 5))  plt.bar(range(1, len(pca.explained\_variance\_ratio\_) + 1), pca.explained\_variance\_ratio\_, alpha=0.5, align='center', label='Individual Explained Variance')  plt.step(range(1, len(pca.explained\_variance\_ratio\_.cumsum()) + 1), pca.explained\_variance\_ratio\_.cumsum(), where='mid', label='Cumulative Explained Variance')  plt.axvline(x=pca.n\_components\_, color='r', linestyle='--', label=f'95% variance at {pca.n\_components\_} components')  plt.ylabel('Explained Variance Ratio')  plt.xlabel('Principal Component Index')  plt.title('Explained Variance by PCA Components')  plt.legend(loc='best')  plt.show()  # Print the number of components  print(f"Number of components to retain 95% variance: {pca.n\_components\_}")  #Print the comparision of number of Feature reduces through PCA  print("Original Number of Features",X.shape)  print("Reduced Number of Features after PCA Reduction",X\_pca.shape) |

Figure 5: Below is an Explained Variance of the above code snippet for PCA

A graph of a graph

Description automatically generated with medium confidence

Here is a 2 component PCA with output for the same

|  |
| --- |
| # Conduct PCA with 2 components  pca = PCA(n\_components=2)  X\_pca = pca.fit\_transform(X\_scaled)  # Display the results  print(f"Explained variance ratio for each component: {pca.explained\_variance\_ratio\_}")  print(f"Number of components selected: {pca.n\_components\_}")  print(f"Shape of the PCA-transformed data: {X\_pca.shape}")  plt.figure(figsize=(5, 3))  plt.bar(['PC1', 'PC2'], pca.explained\_variance\_ratio\_, color='skyblue', alpha=0.7)  plt.xlabel('Principal Components')  plt.ylabel('Variance Ratio')  plt.title('Explained Variance Ratio by PCA Components')  plt.show() |

Figure 6 : Explained Variance for PCA with 2 Components

A graph with blue squares

Description automatically generated

**CLUSTERING TECHINIQUE**

After PCA, we apply below techniques:

* **Silhouette Analysis and KMeans Clustering:** Utilizing silhouette analysis to ascertain the optimal number of clusters,[5] we then apply KMeans clustering to categorize the data into coherent groups based on their gene expression similarities.

|  |
| --- |
| # Define the range of clusters to analyze  range\_n\_clusters = [2, 3, 4, 5, 6]  # Perform silhouette analysis for each cluster count  for n\_clusters in range\_n\_clusters:      # Initialize KMeans and compute cluster labels      clusterer = KMeans(n\_clusters=n\_clusters, random\_state=10)      cluster\_labels = clusterer.fit\_predict(X\_pca)        # Calculate the average silhouette score      silhouette\_avg = silhouette\_score(X\_pca, cluster\_labels)      print(f"For n\_clusters = {n\_clusters}, the average silhouette\_score is : {silhouette\_avg}")        # Set up the figure for silhouette plot and clustered data visualization      fig, (ax1, ax2) = plt.subplots(1, 2)      fig.set\_size\_inches(8, 4)        # Silhouette plot      ax1.set\_xlim([-0.1, 1])      ax1.set\_ylim([0, len(X\_pca) + (n\_clusters + 1) \* 10])      silhouette\_values = silhouette\_samples(X\_pca, cluster\_labels)        y\_lower = 10      for i in range(n\_clusters):          # Sort silhouette values and plot          ith\_values = np.sort(silhouette\_values[cluster\_labels == i])          y\_upper = y\_lower + ith\_values.shape[0]          color = cm.nipy\_spectral(float(i) / n\_clusters)          ax1.fill\_betweenx(np.arange(y\_lower, y\_upper), 0, ith\_values, facecolor=color, edgecolor=color, alpha=0.7)          y\_lower = y\_upper + 10        # Clustered data visualization      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')        # Plot cluster centers      centers = clusterer.cluster\_centers\_      ax2.scatter(centers[:, 0], centers[:, 1], marker='o', c='white', alpha=1, s=200, edgecolor='k')      for i, c in enumerate(centers):          ax2.scatter(c[0], c[1], marker='$%d$' % i, alpha=1, s=50, edgecolor='k')        # Enhance plot aesthetics      ax1.set\_title("Silhouette Plot", fontsize=14)      ax1.set\_xlabel("Silhouette Coefficient", fontsize=12)      ax1.set\_ylabel("Cluster", fontsize=12)      ax2.set\_title("Clustered Data", fontsize=14)      ax2.set\_xlabel("Feature Space for 1st Feature", fontsize=12)      ax2.set\_ylabel("Feature Space for 2nd Feature", fontsize=12)      plt.suptitle(f"Silhouette Analysis (n\_clusters = {n\_clusters})", fontsize=16, fontweight='bold')        plt.show() |

Figure 7 : Silhouette analysis for K\_Means Clusters ranging from 2 – 6

A graph of a graph of a graph

Description automatically generated with medium confidenceA screen shot of a graph

Description automatically generated A screen shot of a graph

Description automatically generated A graph of different colored shapes

Description automatically generated with medium confidence A graph of different colors and numbers

Description automatically generated with medium confidence

1. **Uniform Manifold Approximation and Projection (UMAP) Analysis**: Following Principal Component Analysis (PCA), we proceed with UMAP analysis to further reduce the dimensionality of the gene expression data.[6] UMAP provides a nonlinear dimensionality reduction, allowing us to uncover intricate structures within the dataset. By implementing UMAP, we transform the data into lower-dimensional representations, facilitating a deeper understanding of underlying patterns and relationships.

|  |
| --- |
| import umap.umap\_ as umap  from sklearn.cluster import AgglomerativeClustering  # Perform UMAP dimensionality reduction  reducer = umap.UMAP(n\_neighbors=15, min\_dist=0.1, n\_components=2, random\_state=42)  X\_umap = reducer.fit\_transform(X\_scaled)  # Perform Hierarchical clustering on the reduced data  hierarchical\_cluster = AgglomerativeClustering(n\_clusters=5, affinity='euclidean', linkage='ward')  hierarchical\_cluster.fit(X\_umap)  hierarchical\_labels = hierarchical\_cluster.labels\_  # Create a DataFrame for visualization  embedding\_df = pd.DataFrame(X\_umap, columns=['UMAP1', 'UMAP2'])  embedding\_df['label'] = labelset['Class']  embedding\_df['hierarchical\_labels'] = hierarchical\_labels  # Create subplots for side by side visualization  fig, axes = plt.subplots(1, 2, figsize=(24, 11))  # Plot UMAP results  sns.scatterplot(      x='UMAP1', y='UMAP2',      hue='label',      palette=sns.color\_palette("hsv", 5),      data=embedding\_df,      legend="full",      alpha=0.8,      ax=axes[0]  )  axes[0].set\_title('UMAP projection of the dataset')  # Plot UMAP results with Hierarchical clustering  scatter = axes[1].scatter(embedding\_df['UMAP1'], embedding\_df['UMAP2'], c=embedding\_df['hierarchical\_labels'], cmap='plasma', s=50)  axes[1].set\_title('UMAP with Hierarchical Clustering')  axes[1].set\_xlabel('UMAP Component 1')  axes[1].set\_ylabel('UMAP Component 2')  plt.colorbar(scatter, ax=axes[1], label='Cluster Label')  # Adjust layout and show plot  plt.tight\_layout()  plt.show() |

Figure 8: Visualized map of UMAP algorithm

A screenshot of a graph

Description automatically generated

**CLUSTERING TECHINIQUE**

In addition to UMAP, apply below techniques :

* **Davies Bouldin analysis and hierarchical clustering:** To gain additional insights into the dataset's structure. Davies Bouldin analysis aids in determining the optimal number of clusters, while hierarchical clustering unveils hierarchical relationships among data points based on their gene expression profiles[6]. By integrating these techniques, we effectively segment the data into meaningful clusters, enabling a comprehensive exploration of tumor subtypes and classifications.

|  |
| --- |
| from sklearn.metrics import davies\_bouldin\_score  from scipy.cluster.hierarchy import dendrogram, linkage  # Calculate the Davies-Bouldin Index  db\_index = davies\_bouldin\_score(X\_umap, hierarchical\_labels)  print(f"Davies-Bouldin Index: {db\_index}")  # Visualization of Hierarchical Clustering Dendrogram  linked = linkage(X\_umap, 'ward')  plt.figure(figsize=(10, 7))  dendrogram(linked, orientation='top', distance\_sort='descending', show\_leaf\_counts=True, truncate\_mode = "lastp")  plt.title('Hierarchical Clustering Dendrogram')  plt.show() |

Figure 9: Explained UMAP with Davies Bouldin analysis and Hierarchical Cluster

A diagram of a clustering diagram

Description automatically generated with medium confidence

* 1. **SUPERVISED LEARNING APPROACH:**

In the supervised learning phase of our project, we focus on developing predictive models that can accurately classify tumor types based on gene expression data. This approach involves training models on a labeled dataset, where the true tumor types are known, allowing the models to learn and make predictions on new, unseen data.

**MODEL DEVELOPMENT**

We have selected four state-of-the-art machine learning algorithms known for their robustness and effectiveness in classification tasks.

|  |
| --- |
| X = dataset  Y = labelset['Class']  # Split the dataset into training and testing sets with a test size of 25%  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.25, random\_state=42)  # Initialize PCA to retain 95% of the variance  pca = PCA(n\_components=0.95)  # Fit PCA on the training data and transform both training and testing data  X\_train\_pca = pca.fit\_transform(X\_train)  X\_test\_pca = pca.transform(X\_test)  # The shapes of the transformed data can be printed to verify the reduction  print(f"Original training data shape: {X\_train.shape}")  print(f"Transformed training data shape: {X\_train\_pca.shape}")  print(f"Original testing data shape: {X\_test.shape}")  print(f"Transformed testing data shape: {X\_test\_pca.shape}") |

* + 1. **RandomForest Classifier**: A versatile ensemble method that builds multiple decision trees and merges them to get a more accurate and stable prediction.

|  |
| --- |
| # Initialize the RandomForestClassifier with a random state for reproducibility for "PCA"  rfc\_clf = RandomForestClassifier(random\_state=42)  # Fit the classifier on the PCA-transformed training data  rfc\_clf.fit(X\_train\_pca, y\_train)  # Predict on the PCA-transformed testing data  y\_pred = rfc\_clf.predict(X\_test\_pca)  # Define target names for the classification report  target\_names = ['BRCA', 'COAD', 'KIRC', 'LUAD', 'PRAD']  # Calculate metrics  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 metrics  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))  # Compute the confusion matrix  conf\_matrix = confusion\_matrix(y\_true=y\_test, y\_pred=y\_pred)  classes = np.unique(y\_test)  # Create a figure with subplots for the confusion matrix and class prediction error  fig, axes = plt.subplots(1, 2, figsize=(10, 4))  # Plotting the confusion matrix on the first subplot  axes[0].imshow(conf\_matrix, cmap=plt.cm.Greys)  axes[0].set\_title('Random Forest Confusion Matrix')  axes[0].set\_xticks(np.arange(len(target\_names)))  axes[0].set\_yticks(np.arange(len(target\_names)))  axes[0].set\_xticklabels(target\_names)  axes[0].set\_yticklabels(target\_names)  plt.setp(axes[0].get\_xticklabels(), rotation=45, ha="right", rotation\_mode="anchor")  plt.setp(axes[0].get\_yticklabels(), rotation=45, ha="right", rotation\_mode="anchor")  thresh = conf\_matrix.max() / 2.  for i in range(conf\_matrix.shape[0]):      for j in range(conf\_matrix.shape[1]):          axes[0].text(j, i, format(conf\_matrix[i, j], 'd'), ha="center", va="center",                       color="white" if conf\_matrix[i, j] > thresh else "black")  # Plotting Class Prediction Error on the second subplot  class\_totals = conf\_matrix.sum(axis=1)  class\_correct = np.diag(conf\_matrix)  class\_error = class\_totals - class\_correct  indices = np.arange(len(classes))  bar\_width = 0.35  axes[1].bar(indices, class\_correct / class\_totals, bar\_width, label='Correct', color='Green')  axes[1].bar(indices + bar\_width, class\_error / class\_totals, bar\_width, label='Incorrect', color='red')  axes[1].set\_title('Class Prediction Error')  axes[1].set\_xlabel('Class')  axes[1].set\_ylabel('Proportion')  axes[1].set\_xticks(indices + bar\_width / 2)  axes[1].set\_xticklabels(classes)  axes[1].legend()  # Adjust layout and show plot  plt.tight\_layout()  plt.show() |

Figure 10: Output for the RandomForest Classifier model using PCA

A screenshot of a computer

Description automatically generated

* + 1. **XGBoost Classifier:** An implementation of gradient boosted decision trees designed for speed and performance.

|  |
| --- |
| import xgboost as xgb  from sklearn.preprocessing import LabelEncoder  # Data preparation  label\_encoder = LabelEncoder()  y\_encoded = label\_encoder.fit\_transform(Y)  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_encoded, test\_size=0.25, random\_state=42)  # PCA transformation  pca = PCA(n\_components=0.95)  X\_train\_pca = pca.fit\_transform(X\_train)  X\_test\_pca = pca.transform(X\_test)  # XGBoost classifier for "PCA"  xgb\_clf = xgb.XGBClassifier(objective='multi:softprob', num\_class=len(set(y\_encoded)), seed=42)  xgb\_clf.fit(X\_train\_pca, y\_train)  y\_pred = xgb\_clf.predict(X\_test\_pca)  # Calculate 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 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}%")  # Classification report  print("\nClassification Report:")  print(classification\_report(y\_test, y\_pred, target\_names=label\_encoder.classes\_))  # Confusion matrix  conf\_matrix = confusion\_matrix(y\_test, y\_pred)  # Create a figure with subplots for the confusion matrix and class prediction error  fig, axes = plt.subplots(1, 2, figsize=(10, 4))  # Plotting the confusion matrix on the first subplot  axes[0].imshow(conf\_matrix, cmap=plt.cm.Greys)  axes[0].set\_title('XGBoost Confusion Matrix')  axes[0].set\_xticks(np.arange(len(target\_names)))  axes[0].set\_yticks(np.arange(len(target\_names)))  axes[0].set\_xticklabels(target\_names)  axes[0].set\_yticklabels(target\_names)  plt.setp(axes[0].get\_xticklabels(), rotation=45, ha="right", rotation\_mode="anchor")  plt.setp(axes[0].get\_yticklabels(), rotation=45, ha="right", rotation\_mode="anchor")  thresh = conf\_matrix.max() / 2.  for i in range(conf\_matrix.shape[0]):      for j in range(conf\_matrix.shape[1]):          axes[0].text(j, i, format(conf\_matrix[i, j], 'd'), ha="center", va="center",                       color="white" if conf\_matrix[i, j] > thresh else "black")  # Plotting Class Prediction Error on the second subplot  class\_totals = conf\_matrix.sum(axis=1)  class\_correct = np.diag(conf\_matrix)  class\_error = class\_totals - class\_correct  indices = np.arange(len(classes))  bar\_width = 0.35  axes[1].bar(indices, class\_correct / class\_totals, bar\_width, label='Correct', color='Green')  axes[1].bar(indices + bar\_width, class\_error / class\_totals, bar\_width, label='Incorrect', color='red')  axes[1].set\_title('Class Prediction Error')  axes[1].set\_xlabel('Class')  axes[1].set\_ylabel('Proportion')  axes[1].set\_xticks(indices + bar\_width / 2)  axes[1].set\_xticklabels(classes)  axes[1].legend()  # Adjust layout and show plot  plt.tight\_layout()  plt.show() |

Figure 11: Output of XGBoost Classifier with PCA

A screenshot of a computer

Description automatically generated

* + 1. **Neural Network**: A deep learning algorithm that models complex relationships between inputs and outputs through layers of interconnected nodes.

|  |
| --- |
| from sklearn.neural\_network import MLPClassifier  # Initialize the Neural Network classifier with one hidden layer of 150 neurons  nn\_clf = MLPClassifier(hidden\_layer\_sizes=(150,), max\_iter=200, activation='relu', solver='adam', random\_state=1)  # Train the model and predict on the test set  nn\_clf.fit(X\_train\_pca, y\_train)  y\_pred = nn\_clf.predict(X\_test\_pca)  # Define target names for the classes  target\_names = ['BRCA', 'COAD', 'KIRC', 'LUAD', 'PRAD']  # Calculate 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 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))  # Compute the confusion matrix  conf\_matrix = confusion\_matrix(y\_true=y\_test, y\_pred=y\_pred)  classes = np.unique(y\_test)  # Create a figure with subplots for the confusion matrix and class prediction error  fig, axes = plt.subplots(1, 2, figsize=(10, 4))  # Plotting the confusion matrix on the first subplot  axes[0].imshow(conf\_matrix, cmap=plt.cm.Greys)  axes[0].set\_title('Neural Network Confusion Matrix')  axes[0].set\_xticks(np.arange(len(target\_names)))  axes[0].set\_yticks(np.arange(len(target\_names)))  axes[0].set\_xticklabels(target\_names)  axes[0].set\_yticklabels(target\_names)  plt.setp(axes[0].get\_xticklabels(), rotation=45, ha="right", rotation\_mode="anchor")  plt.setp(axes[0].get\_yticklabels(), rotation=45, ha="right", rotation\_mode="anchor")  thresh = conf\_matrix.max() / 2.  for i in range(conf\_matrix.shape[0]):      for j in range(conf\_matrix.shape[1]):          axes[0].text(j, i, format(conf\_matrix[i, j], 'd'), ha="center", va="center",                       color="white" if conf\_matrix[i, j] > thresh else "black")  # Plotting Class Prediction Error on the second subplot  class\_totals = conf\_matrix.sum(axis=1)  class\_correct = np.diag(conf\_matrix)  class\_error = class\_totals - class\_correct  indices = np.arange(len(classes))  bar\_width = 0.35  axes[1].bar(indices, class\_correct / class\_totals, bar\_width, label='Correct', color='Green')  axes[1].bar(indices + bar\_width, class\_error / class\_totals, bar\_width, label='Incorrect', color='red')  axes[1].set\_title('Class Prediction Error')  axes[1].set\_xlabel('Class')  axes[1].set\_ylabel('Proportion')  axes[1].set\_xticks(indices + bar\_width / 2)  axes[1].set\_xticklabels(classes)  axes[1].legend()  # Adjust layout and show plot  plt.tight\_layout()  plt.show() |

Figure 12: Output of Neural Network Model with PCA A screenshot of a computer

Description automatically generated

* + 1. **Gradient Boosting**: Another ensemble technique that builds the model in a stage-wise fashion and generalizes them by allowing optimization of an arbitrary differentiable loss function.

|  |
| --- |
| from sklearn.ensemble import GradientBoostingClassifier  # Initialize the Gradient Boosting Classifier  gb\_clf = GradientBoostingClassifier()  # Fit the model  gb\_clf.fit(X\_train\_pca, y\_train)  y\_pred = gb\_clf.predict(X\_test\_pca)  # Define target names for the classes  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 Boosting 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}%")  # Generate and display the classification report  print("\nClassification Report:")  print(classification\_report(y\_test, y\_pred, target\_names=target\_names))  # Compute the confusion matrix  conf\_matrix = confusion\_matrix(y\_true=y\_test, y\_pred=y\_pred)  classes = np.unique(y\_test)  # Create a figure with subplots for the confusion matrix and class prediction error  fig, axes = plt.subplots(1, 2, figsize=(10, 4))  # Plotting the confusion matrix on the first subplot  axes[0].imshow(conf\_matrix, cmap=plt.cm.Greys)  axes[0].set\_title('Gradient Boosting Confusion Matrix')  axes[0].set\_xticks(np.arange(len(target\_names)))  axes[0].set\_yticks(np.arange(len(target\_names)))  axes[0].set\_xticklabels(target\_names)  axes[0].set\_yticklabels(target\_names)  plt.setp(axes[0].get\_xticklabels(), rotation=45, ha="right", rotation\_mode="anchor")  plt.setp(axes[0].get\_yticklabels(), rotation=45, ha="right", rotation\_mode="anchor")  thresh = conf\_matrix.max() / 2.  for i in range(conf\_matrix.shape[0]):      for j in range(conf\_matrix.shape[1]):          axes[0].text(j, i, format(conf\_matrix[i, j], 'd'), ha="center", va="center",                       color="white" if conf\_matrix[i, j] > thresh else "black")  # Plotting Class Prediction Error on the second subplot  class\_totals = conf\_matrix.sum(axis=1)  class\_correct = np.diag(conf\_matrix)  class\_error = class\_totals - class\_correct  indices = np.arange(len(classes))  bar\_width = 0.35  axes[1].bar(indices, class\_correct / class\_totals, bar\_width, label='Correct', color='Green')  axes[1].bar(indices + bar\_width, class\_error / class\_totals, bar\_width, label='Incorrect', color='red')  axes[1].set\_title('Class Prediction Error')  axes[1].set\_xlabel('Class')  axes[1].set\_ylabel('Proportion')  axes[1].set\_xticks(indices + bar\_width / 2)  axes[1].set\_xticklabels(classes)  axes[1].legend()  # Adjust layout and show plot  plt.tight\_layout()  plt.show() |

Figure 13: Output of GradientBoosting Classifier with PCA

A screenshot of a computer

Description automatically generated

* 1. **Supervised Learning Approach using UMAP as feature reduction Technique:**

In the Supervised Learning Approach, we have incorporated UMAP as a feature reduction technique, which, unlike PCA[7], is particularly adept at preserving both the local and global structure of high-dimensional data. This advanced method has been applied to refine our feature set before training our models.

* + 1. **Model Overfitting Consideration**

Upon analyzing the performance metrics of our classifiers, we observe a pattern indicative of model overfitting. Overfitting occurs when a model learns the training data too well, capturing noise along with the underlying pattern. It is characterized by perfect or near-perfect performance metrics on the training data, which are not replicated on unseen data.

A paper with numbers and words

Description automatically generated

Table 1: Table of Performance Metrics of Supervised models using UMAP

As seen, the Random Forest, SVM, Gradient Boosting, and Neural Network models exhibit perfect scores across all metrics, suggesting that they may have overfitted to the training data. The XGBoost model, while showing slightly less than perfect scores, also indicates a high likelihood of overfitting given its near-perfect metrics.

# **EVALUATION AND ANALYSIS**:

The evaluation and analysis phase is a critical component of our project, serving as the juncture where the performance of our machine learning models is rigorously assessed and interpreted. This phase is meticulously designed to ensure that the models we have developed not only exhibit high accuracy but also maintain precision, recall, and F1-score metrics that are paramount in the context of medical diagnostics.

* 1. **Performance Metrics**

To evaluate the effectiveness of our models[8], we employ a comprehensive set of performance metrics:

* **Accuracy:** This metric provides a general indication of the model’s ability to correctly classify tumor types.
* **Precision:** Precision measures the proportion of true positive predictions in the total predicted positives, highlighting the model’s ability to minimize false positives.
* **Recall (Sensitivity):** Recall assesses the model’s capability to identify all actual positives, emphasizing its sensitivity to detecting tumor types.
* **F1-Score:** The F1-score is the harmonic mean of precision and recall, offering a balance between the two and serving as a robust indicator of the model’s overall performance.
  1. **Comparative Analysis and Interpretation of Results**

The Comparative Analysis and Interpretation of Results section is the culmination of our project’s analytical efforts, where we juxtapose the performance of our machine learning models and delve into the implications of their predictive capabilities.

* + 1. **Comparative Analysis**

We conducted a comparative analysis of the RandomForest, XGBoost, Neural Network, and Gradient Boosting models, utilizing a series of performance metrics accuracy, precision, recall, and F1-score to evaluate each model’s efficacy in classifying tumor types from PCA-transformed gene expression data. By graphically representing these metrics, we gained a clear visual perspective of each model’s performance, allowing us to discern the most effective algorithm for our specific dataset.

A paper with numbers and words

Description automatically generated

Table 2: Table of Performance Metrics

* + 1. **Interpretation of Results**

The interpretation of these results extends beyond numerical comparisons; it involves a thoughtful examination of what the metrics reveal about the models’ abilities to generalize from the training data to make accurate predictions on new data. We considered the balance between precision and recall, particularly the F1-score, to ensure our model not only identifies tumor types accurately but also minimizes false positives and negatives critical factors in a medical diagnostic setting.

A diagram of different colors and numbers

Description automatically generated with medium confidence

Figure 14: Comparative Analysis of the Performance Metrics

# **CONCLUSION:**

In this project, we embarked on a comprehensive journey to classify tumor types based on gene expression data. Through the application of advanced machine learning techniques, we developed models that demonstrated a high degree of accuracy in predicting tumor classifications. The RandomForest, XGBoost, Neural Network, and Gradient Boosting algorithms each played a pivotal role in our analysis, showcasing their respective strengths in handling complex biological data.

Our findings indicate that with the right feature reduction techniques and model configurations, it is possible to achieve near-perfect classification results. However, the perfect scores across our models also prompted a critical evaluation of potential overfitting, a common pitfall in machine learning. We addressed this by considering strategies such as cross-validation and regularization to ensure our models’ robustness and generalizability. The implications of our work are significant, offering potential pathways for enhancing precision medicine and providing insights that could lead to more targeted cancer treatments. By accurately classifying tumors, clinicians can better tailor therapies to individual patients, improving outcomes and personalizing the healthcare experience.

# **FUTURE WORK:**

In conclusion, our project has taken significant strides in the classification of tumor types through gene expression data, utilizing a suite of sophisticated machine learning models. The accuracy of the RandomForest, XGBoost, Neural Network, and Gradient Boosting algorithms in our study suggests that such techniques can be highly effective when paired with appropriate feature reduction methods. Despite achieving near-perfect classification results, we have critically assessed the potential for overfitting and have implemented strategies to enhance the robustness and generalizability of our models. The potential impact of our work on precision medicine is substantial, offering insights that could lead to more personalized and effective cancer treatments.

As we look to the future, several research paths beckon. Data augmentation could provide a bulwark against overfitting, while model ensembles may offer improved performance and reliability. Collaborating with clinical experts for model validation could bridge the gap between computational predictions and clinical application. Furthermore, enhancing model interpretability will be crucial for clinician acceptance and understanding. Lastly, exploring new machine learning algorithms and feature reduction techniques may yield even more powerful tools for tumor classification. Our ongoing efforts aim to refine these models and contribute to the evolving landscape of cancer diagnosis and treatment, leveraging the transformative potential of genomics and machine learning.

# **APPENDIX:**

**A. CODE SNIPPETS**

Throughout the report, we have referenced various machine learning models and analyses performed. For the sake of brevity and readability, the full code has not been included within the main body of the text. However, detailed code snippets for each significant step of our methodology are provided in this appendix. These snippets include the initialization and training of models, performance evaluation, and data visualization techniques.

|  |
| --- |
| dataset=pd.read\_csv("data.csv")  labelset = pd.read\_csv("labels.csv")  # Drop unnamed columns from the dataset  dataset.drop(dataset.filter(regex="Unnamed").columns, axis=1, inplace=True)  # Basic statistics for the dataset  # dataset\_stats = dataset.apply(calculate\_stats)  # Number of columns with missing values  num\_columns\_with\_missing\_values = dataset.isnull().any().sum()  # Visualization: Pie chart for class distribution  labelset['Class'].value\_counts().plot.pie(autopct='%1.1f%%', startangle=140, figsize=(6, 6))  plt.title('Distribution of Labels')  plt.show()  # Class distribution  class\_distribution = labelset['Class'].value\_counts()  # Print outputs  print("\nNumber of columns with missing values:", num\_columns\_with\_missing\_values)  print("\nClass Distribution:")  print(class\_distribution)  print("\nDataset Information:")  dataset.info()  print("\nLabelset Information:")  labelset.info()  #Using StandardScalar module for Normalization  X = dataset  scaler = StandardScaler()  X\_scaled = scaler.fit\_transform(X)  X\_scaled  # Perform PCA to retain 95% of the variance  pca = PCA(n\_components=0.95)  X\_pca = pca.fit\_transform(X\_scaled)  # Plotting the explained variance  plt.figure(figsize=(8, 5))  plt.bar(range(1, len(pca.explained\_variance\_ratio\_) + 1), pca.explained\_variance\_ratio\_, alpha=0.5, align='center', label='Individual Explained Variance')  plt.step(range(1, len(pca.explained\_variance\_ratio\_.cumsum()) + 1), pca.explained\_variance\_ratio\_.cumsum(), where='mid', label='Cumulative Explained Variance')  plt.axvline(x=pca.n\_components\_, color='r', linestyle='--', label=f'95% variance at {pca.n\_components\_} components')  plt.ylabel('Explained Variance Ratio')  plt.xlabel('Principal Component Index')  plt.title('Explained Variance by PCA Components')  plt.legend(loc='best')  plt.show()  # Print the number of components  print(f"Number of components to retain 95% variance: {pca.n\_components\_}")  #Print the comparision of number of Feature reduces through PCA  print("=============================================\nOriginal Number of Features",X.shape)  print("Reduced Number of Features after PCA Reduction",X\_pca.shape)  # Initialize KMeans with 5 clusters  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)  # Retrieve the cluster labels  clusters = kmeans.labels\_  # Set up the plot  plt.figure(figsize=(6, 6))  # Plot each cluster  for cluster in np.unique(clusters):      # Create a mask for the cluster      mask = clusters == cluster      # Scatter plot for each cluster      plt.scatter(X\_pca[mask, 0], X\_pca[mask, 1], s=50, label=f"Cluster {cluster}", cmap='plasma')  # Plot the centroids  centroids = kmeans.cluster\_centers\_  plt.scatter(centroids[:, 0], centroids[:, 1], c='black', marker='X', s=200, label="Centroids")  # Labeling the plot  plt.xlabel('Principal Component 1', fontsize=12)  plt.ylabel('Principal Component 2', fontsize=12)  plt.title('KMeans Clustering Results', fontsize=14)  # Add a legend  plt.legend(title="Cluster Legend", bbox\_to\_anchor=(1.05, 1), loc='upper left')  # Show grid and plot  plt.grid(True)  plt.show()  import umap.umap\_ as umap  from sklearn.cluster import AgglomerativeClustering  # Perform UMAP dimensionality reduction  reducer = umap.UMAP(n\_neighbors=15, min\_dist=0.1, n\_components=2, random\_state=42)  X\_umap = reducer.fit\_transform(X\_scaled)  # Perform Hierarchical clustering on the reduced data  hierarchical\_cluster = AgglomerativeClustering(n\_clusters=5, affinity='euclidean', linkage='ward')  hierarchical\_cluster.fit(X\_umap)  hierarchical\_labels = hierarchical\_cluster.labels\_  # Create a DataFrame for visualization  embedding\_df = pd.DataFrame(X\_umap, columns=['UMAP1', 'UMAP2'])  embedding\_df['label'] = labelset['Class']  embedding\_df['hierarchical\_labels'] = hierarchical\_labels  # Create subplots for side by side visualization  fig, axes = plt.subplots(1, 2, figsize=(24, 11))  # Plot UMAP results  sns.scatterplot(      x='UMAP1', y='UMAP2',      hue='label',      palette=sns.color\_palette("hsv", 5),      data=embedding\_df,      legend="full",      alpha=0.8,      ax=axes[0]  )  axes[0].set\_title('UMAP projection of the dataset')  # Plot UMAP results with Hierarchical clustering  scatter = axes[1].scatter(embedding\_df['UMAP1'], embedding\_df['UMAP2'], c=embedding\_df['hierarchical\_labels'], cmap='plasma', s=50)  axes[1].set\_title('UMAP with Hierarchical Clustering')  axes[1].set\_xlabel('UMAP Component 1')  axes[1].set\_ylabel('UMAP Component 2')  plt.colorbar(scatter, ax=axes[1], label='Cluster Label')  # Adjust layout and show plot  plt.tight\_layout()  plt.show()  from sklearn.metrics import davies\_bouldin\_score  from scipy.cluster.hierarchy import dendrogram, linkage  # Calculate the Davies-Bouldin Index  db\_index = davies\_bouldin\_score(X\_umap, hierarchical\_labels)  print(f"Davies-Bouldin Index: {db\_index}")  # Visualization of Hierarchical Clustering Dendrogram  linked = linkage(X\_umap, 'ward')  plt.figure(figsize=(10, 7))  dendrogram(linked, orientation='top', distance\_sort='descending', show\_leaf\_counts=True, truncate\_mode = "lastp")  plt.title('Hierarchical Clustering Dendrogram')  plt.show()  X = dataset  Y = labelset['Class']  # Split the dataset into training and testing sets with a test size of 25%  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.25, random\_state=42)  # Initialize PCA to retain 95% of the variance  pca = PCA(n\_components=0.95)  # Fit PCA on the training data and transform both training and testing data  X\_train\_pca = pca.fit\_transform(X\_train)  X\_test\_pca = pca.transform(X\_test)  # The shapes of the transformed data can be printed to verify the reduction  print(f"Original training data shape: {X\_train.shape}")  print(f"Transformed training data shape: {X\_train\_pca.shape}")  print(f"Original testing data shape: {X\_test.shape}")  print(f"Transformed testing data shape: {X\_test\_pca.shape}")  # Initialize the RandomForestClassifier with a random state for reproducibility for "PCA"  rfc\_clf = RandomForestClassifier(random\_state=42)  # Fit the classifier on the PCA-transformed training data  rfc\_clf.fit(X\_train\_pca, y\_train)  # Predict on the PCA-transformed testing data  y\_pred = rfc\_clf.predict(X\_test\_pca)  # Define target names for the classification report  target\_names = ['BRCA', 'COAD', 'KIRC', 'LUAD', 'PRAD']  # Calculate metrics  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 metrics  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))  # Compute the confusion matrix  conf\_matrix = confusion\_matrix(y\_true=y\_test, y\_pred=y\_pred)  classes = np.unique(y\_test)  # Create a figure with subplots for the confusion matrix and class prediction error  fig, axes = plt.subplots(1, 2, figsize=(10, 4))  # Plotting the confusion matrix on the first subplot  axes[0].imshow(conf\_matrix, cmap=plt.cm.Greys)  axes[0].set\_title('Random Forest Confusion Matrix')  axes[0].set\_xticks(np.arange(len(target\_names)))  axes[0].set\_yticks(np.arange(len(target\_names)))  axes[0].set\_xticklabels(target\_names)  axes[0].set\_yticklabels(target\_names)  plt.setp(axes[0].get\_xticklabels(), rotation=45, ha="right", rotation\_mode="anchor")  plt.setp(axes[0].get\_yticklabels(), rotation=45, ha="right", rotation\_mode="anchor")  thresh = conf\_matrix.max() / 2.  for i in range(conf\_matrix.shape[0]):      for j in range(conf\_matrix.shape[1]):          axes[0].text(j, i, format(conf\_matrix[i, j], 'd'), ha="center", va="center",                       color="white" if conf\_matrix[i, j] > thresh else "black")  # Plotting Class Prediction Error on the second subplot  class\_totals = conf\_matrix.sum(axis=1)  class\_correct = np.diag(conf\_matrix)  class\_error = class\_totals - class\_correct  indices = np.arange(len(classes))  bar\_width = 0.35  axes[1].bar(indices, class\_correct / class\_totals, bar\_width, label='Correct', color='Green')  axes[1].bar(indices + bar\_width, class\_error / class\_totals, bar\_width, label='Incorrect', color='red')  axes[1].set\_title('Class Prediction Error')  axes[1].set\_xlabel('Class')  axes[1].set\_ylabel('Proportion')  axes[1].set\_xticks(indices + bar\_width / 2)  axes[1].set\_xticklabels(classes)  axes[1].legend()  # Adjust layout and show plot  plt.tight\_layout()  plt.show()  import xgboost as xgb  from sklearn.preprocessing import LabelEncoder  # Data preparation  label\_encoder = LabelEncoder()  y\_encoded = label\_encoder.fit\_transform(Y)  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_encoded, test\_size=0.25, random\_state=42)  # PCA transformation  pca = PCA(n\_components=0.95)  X\_train\_pca = pca.fit\_transform(X\_train)  X\_test\_pca = pca.transform(X\_test)  # XGBoost classifier for "PCA"  xgb\_clf = xgb.XGBClassifier(objective='multi:softprob', num\_class=len(set(y\_encoded)), seed=42)  xgb\_clf.fit(X\_train\_pca, y\_train)  y\_pred = xgb\_clf.predict(X\_test\_pca)  # Calculate 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 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}%")  # Classification report  print("\nClassification Report:")  print(classification\_report(y\_test, y\_pred, target\_names=label\_encoder.classes\_))  # Confusion matrix  conf\_matrix = confusion\_matrix(y\_test, y\_pred)  # Create a figure with subplots for the confusion matrix and class prediction error  fig, axes = plt.subplots(1, 2, figsize=(10, 4))  # Plotting the confusion matrix on the first subplot  axes[0].imshow(conf\_matrix, cmap=plt.cm.Greys)  axes[0].set\_title('XGBoost Confusion Matrix')  axes[0].set\_xticks(np.arange(len(target\_names)))  axes[0].set\_yticks(np.arange(len(target\_names)))  axes[0].set\_xticklabels(target\_names)  axes[0].set\_yticklabels(target\_names)  plt.setp(axes[0].get\_xticklabels(), rotation=45, ha="right", rotation\_mode="anchor")  plt.setp(axes[0].get\_yticklabels(), rotation=45, ha="right", rotation\_mode="anchor")  thresh = conf\_matrix.max() / 2.  for i in range(conf\_matrix.shape[0]):      for j in range(conf\_matrix.shape[1]):          axes[0].text(j, i, format(conf\_matrix[i, j], 'd'), ha="center", va="center",                       color="white" if conf\_matrix[i, j] > thresh else "black")  # Plotting Class Prediction Error on the second subplot  class\_totals = conf\_matrix.sum(axis=1)  class\_correct = np.diag(conf\_matrix)  class\_error = class\_totals - class\_correct  indices = np.arange(len(classes))  bar\_width = 0.35  axes[1].bar(indices, class\_correct / class\_totals, bar\_width, label='Correct', color='Green')  axes[1].bar(indices + bar\_width, class\_error / class\_totals, bar\_width, label='Incorrect', color='red')  axes[1].set\_title('Class Prediction Error')  axes[1].set\_xlabel('Class')  axes[1].set\_ylabel('Proportion')  axes[1].set\_xticks(indices + bar\_width / 2)  axes[1].set\_xticklabels(classes)  axes[1].legend()  # Adjust layout and show plot  plt.tight\_layout()  plt.show() |

**B. GITHUB REPOSITORY**

To facilitate reproducibility and further research, the complete source code, along with additional resources used in this project, is available in our GitHub repository. The repository includes:

* Jupyter notebooks containing the executed code and output for each model.
* Scripts for data preprocessing and analysis.
* Requirements file specifying the necessary libraries and their versions.
* README file with instructions on how to set up the environment and run the code.

The repository can be accessed at the following link:

**GitHub Repository:**

<https://github.com/vishnugupthaa/Cancer-Type-Prediction-and-Exploration-of-TCGA-Data.git>

# **REFERENCES:**

|  |  |
| --- | --- |
| [1] | Chang, K., Creighton, C. J., Davis, C., Donehower, L., Drummond, J., Wheeler, D., Ally, A., et al. (2013). The Cancer Genome Atlas Pan-Cancer analysis project. *Nature Genetics, 45*(10), 1113-1120. http://dx.doi.org/10.1038/ng.2764. |
| [2] | Furat, F., GEMC.I., & Ibrikci, T. (2019). Exploring tumor type detection through the Naïve Bayes algorithm on RNA-Seq gene expression data. Lung Cancer, 10, 13. |
| [3] | Retinraj, P. D. (2023, June 10). Machine Learning System Design: Template. AI Science. “https://medium.com/ai-science/machine-learning-system-design-template-2f296622aec1” |
| [4] | Shlens, J. (2003, March 25). A Tutorial on Principal Component Analysis: Derivation, Discussion and Singular Value Decomposition. Version 1. |
| [5] | UBT Knowledge Center. (2019). An Application of PCA Based K-Means Clustering for Customer Segmentation in One Luxury Goods Company. In UBT International Conference 2019 Proceedings (pp. 2362). University for Business and Technology in Kosovo. Retrieved from “https://knowledgecenter.ubt-uni.net/cgi/viewcontent.cgi?article=2362&context=conference”. |
| [6] | UMAP-learn Contributors. (n.d.). UMAP Documentation. Retrieved from” https://umap-learn.readthedocs.io/en/latest/”. |
| [7] | F. L. G. R. F. H. F. D. A. Gewers, N. Filipi, H. Cesar and Comin, "Principal component analysis: A natural approach to data exploration.," *ACM Computing Surveys (CSUR),* vol. 54, pp. 1-34, 2021. |
| [8] | Ž. Vujović, "Classification model evaluation metrics.," *International Journal of Advanced Computer Science and Applications,* 202. |