Prediction Accuracy On Cancer Classification

Different Models Comparison

Dataset: Classification Of Cancer Type

Yukuan Zou 1008808024
Yue Zhang 1009086481
Ziming Wang 1008475115
Vicky Lai 1008961731

Kaggle team name: Group 1
Final ranking and prediction score: 7 and 1.00

University of Toronto

Dec, 2024

1 Introduction

Cancer, a disease that causes morbidity and mortality, has been difficult to conquer in the past and is still difficult today. The prediction performance of cancer type is important because it guarantees that patients receive the best care possible by directly influencing diagnosis, prognosis, and treatment planning. Analyzing gene expression data has become a key approach to distinguishing cancer types. Misclassification of cancer type may cause severe issues, such as further deterioration of cancer due to complications from the use of inappropriate medication [**2**], it emphasizes how such errors can distort treatment decisions and impact patient safety. Inaccuracies in classification or diagnosis can significantly alter treatment effect estimates. This highlights why accurate prediction performance is important for our medication progress. Based on the study of Lu and Han's study, accurate cancer classification is the key to enhancing diagnostic precision and improving therapeutic efficacy. They highlight the key role of cancer classification in differentiating cancer types using gene expression data and also emphasize how to use statistical learning methods to solve their potential problems, which is the main problem we aim to solve in this report [**1**].

Large-scale initiatives such as the Cancer Genome Atlas (TCGA), a collaboration between the NCI and NHGRI which covers genomic data from more than 11,000 patients across 33 cancers [**7**], have offered valuable resources, providing gene expression profiles for various cancers, such as adrenocortical carcinoma (ACC), Bladder urothelial carcinoma (BLCA), Breast invasive carcinoma (BRCA), and so on [**8**]. Among these, glioblastoma multiforme (GBM), lung squamous cell carcinoma (LUSC), and ovarian cancer (OV) are notable for their distinct molecular features which they reflect differences in cellular origins, genetic mutations, and epigenetic modifications. These distinctions are critical for designing targeted therapies and improving diagnostics, as they influence each cancer type's biological behaviour, response to treatment, and prognosis [**7**].

By using TCGA's gene expression data, we aim to examines the prediction accuracy and comparison between different statistical methods and machine learning models: Lasso, Linear Discriminant Analyst, Decision tree, and XG boost in classifying cancer types. Moreover, throughout the article, statistical methods like Principal Component Analysis (PCA) or t-distributed Stochastic Neighbor Embedding(t-SNE) could help in dealing with a large number of variables. They are powerful tools for visualizing high-dimensional data. Its primary purpose is to reduce the dimensionality of data while preserving the relationships between data points. We would then explore the models' ability to generalize to unseen and testing data and their robustness when subjected to noise or the removal of features. Therefore, we could aim to provide a comprehensive understanding of the performance of each model and analyze the process of how we conduct them.

2 Method

2.1 Data Prepartion

The dataset was initially divided into train and test sets, where train set contains 886 observations and 12042 predictors, and test set contains 379 observations and 12042 predictors. The ID columns in both datasets were removed to ensure they did not interfere with the analysis. The feature variables were then reordered alphabetically, with the target variable – `"cancer"`, positioned as the first column to maintain consistency across datasets.
To facilitate model evaluation, the train set was further split into an 80:20 ratio randomly. This resulted in a sub-training set of 709 samples and a sub-validation set of 177 samples. During all statistical approaches, we use the sub-training set to fit the model and the sub-validation set to simulate the unseen data and thus verify the model's generalization ability.

2.2 Lasso

Since the dataset contains 12042 genes but only 886 observations, it is insufficient to fit an effective logistic regression that can predict the types of cancer. Too many predictors will cause a huge variance in the predicted classes, while the bias will be much smaller because the model has more training data. The core idea of using the Lasso model is to relax the requirement for bias to significantly reduce variance when building models with strong generalization ability.

2.2.1 Mathematical Formula

Suppose that $\boldsymbol{\beta}$ is the vector of coefficients of multinomial logistic regression; $\mathscr{L}(\boldsymbol{\beta})$ is the loss function of multinomial logistic Lasso regression; $L(\boldsymbol{\beta})$ is the likelihood function of multinomial logistic regression. The loss function for multinomial logistic lasso regression is defined by:

$$\mathscr{L}(\boldsymbol{\beta}) = -\log(L(\boldsymbol{\beta})) + \lambda \|\boldsymbol{\beta}\|_1$$

Here $\lambda > 0$ is a tuning parameter determined by cross-validation.

2.2.2 Model Training

We use the `cv.glmnet()` function to obtain candidate lambda values by setting parameter $\alpha = 1$ and `family = "multinomial"` to specify the lasso penalty applied to the multilateral logistic regression. Then we choose the minimum value among the candidate lambda values as the optimal lambda value. We fit the lasso model with the `glmnet()` function and the optimal lambda value on the sub-training set, and then use the `predict()` function and the model to obtain the predicted classes and probabilities of the model on the sub-validation set by setting `type = "class"` and `type = "response"`, respectively. If the result from the sub-validation set is satisfying(with high prediction accuracy), we apply a similar procedure to the entire train set. By cross-validation of lambda on the train set, we can get another optimal lambda value, and then use it to make predictions for unseen test sets.

2.3 Linear Discriminant Analysis(LDA)

LDA is effective for addressing the challenge of high dimensionality relative to the number of samples, particularly when classifying data into two or more categories. Moreover, to increase the the efficiency of the model we would use dimensionality reduction through Principal Component Analysis (PCA) before we generate the model. PCA reduces the feature set by putting the data into a lower-dimensional space that retains the most significant variance, enabling the LDA classifier to focus on the most informative features. This approach not only helps to increase efficiency but also to avoid a problem that many models would face which is referred to as the `"curse of dimensionality"`.

2.3.1 Model Training

We first applied PCA to the sub-training set. This would select the principal components that cumulatively explain at least 95% of the variance in the given dataset. We also use the center to adjust the data so each feature has a mean of zero, which is essential for PCA and improves numerical stability. Then, we fit a LDA model based on PCA principal components. After that, we use a sub-validation set to test the model whether it meets the expected accuracy. If not, We would modify the variance level which could be at least 75% of the variance instead of 95%. Once we have satisfactory prediction accuracy on the sub-validation set, we will use the same procedure again with the full train set.

2.4 Decision Tree

Decision tree can be used to analyze this study due to its ability to classify and graphical representation. We use recursive binary splitting at each node representing one predictor to grow a classification tree. It automatically identifies the most important features for prediction, this may decrease the probability of misclassification, and is easier for high-dimensional features to be viewed directly. Moreover, it can model nonlinear relationships between features and the target variable without requiring transformations of the data.

2.4.1 Model Training

As mentioned in the data preparation section, we split the train set into sub-training set and sub-validation set. Based on the sub-training data, we first construct a full tree model to see how all features contribute to the classification of cancer type, then we construct a pruned tree to see what the key features are that dominant the contribution.
Since the data of predictors is continuous, we need to select a splitting criterion such as `Gini index` (a way to determine which splitting tree is better if the misclassification rate being the same) to divide the data at each node [**10**]. Then, we use iterative splitting to see whether the sub-training data is split into branches based on the feature that provides the best reduction in impurity at each step, where the feature occurs first being the most important. In addition, we use `"rpart.plot()"` to visualize the decision tree structure and assign the classification for each observation we predicted to be the cancer classification with the highest frequency of occurrence in that region. We use pruned tree to see if there is little contribution features that lie in the full tree, the goal is to simplify the tree by removing unnecessary splits to improve generalization on unseen data. We find the minimum complexity parameter values corresponding to the minimum cross-validation error, then construct the visualization of pruned tree by using `"prune()"` function and see if the tree structure has been simplified.

2.5 eXtreme Gradient Boosting(XGBoost)

XGBoost is a machine-learning approach based on gradient tree boosting, it was proposed by Tianqi Chen in 2014 and it applies second-order Taylor expansion to achieve parallel computing and faster convergence than other traditional models [**3**]. This approach is widely used in high-dimensional structured data to effectively deal with complex model feature relationships, and missing data and prevent overfitting through regularization techniques. Due to its balance of computational speed, model complexity, and scalability [**11**], it is suitable for classification tasks on GBM, LUSC, and OV.

2.5.1 Model Training

Based on the segmentation of the overall data, XGBoost transformed the target variable `"cancer"` into a classification factor and converted the feature variables into numeric matrices to ensure compatibility with XGBoost, which was ultimately used to classify the three cancer types, with key hyper-parameters such as a maximum depth of 6 (for control tree complexity), a learning rate of 0.1 (for stepwise optimization), and the use of 2 threads for efficient computation. The model was trained with 100 rounds of enhancement and successfully balanced performance and computational efficiency.

2.6 Model Evaluation Criterion

For the above four models, we all use the sub-validation set to examine the training model performance on prediction of cancer classification. To quantify this performance, we focus on prediction accuracy. Confusion matrix is an important tool for calculating accuracy, which is a table that visualizes and summarizes the performance of a classification algorithm [**9**]. It helps us to identify the correctness of classification for each of the three cancer categories. During our report, we will mainly use the confusion matrix to analyze the performance of four models. In addition, for XG boost, we use learning curves to examine the error rate on the sub-validation set during each round of training, thus observing the convergence of the model as training progresses, further validating the stability and effectiveness of the model.

3 Results

3.1 Accuracy Comparison Among Different Models

Table 1: Prediction Accuracy Among Different Models

Methods	Overall Training Accuracy	Precision	Recall	F1-score	Accuracy of Testing data on Kaggle
XGBoost	99.4%	GBM(1): 100%	GBM(1): 100%	GBM(1): 1.00	100%
		OV(2): 100%	OV(2): 93.33%	OV(2): 0.97	
		LUSC(3): 98.82%	LUSC(3): 100%	LUSC(3): 0.99	
Lasso	100%	GBM(1): 100%	GBM(1): 100%	GBM(1): 1.00	100%
		OV(2): 100%	OV(2): 100%	OV(2): 1.00	
		LUSC(3): 100%	LUSC(3): 100%	LUSC(3): 1.00	
LDA	100%	GBM(1): 100%	GBM(1): 100%	GBM(1): 1.00	100%
		OV(2): 100%	OV(2): 100%	OV(2): 1.00	
		LUSC(3): 100%	LUSC(3): 100%	LUSC(3): 1.00	
Decision Tree (Full and Pruned)	98.9%	GBM(1): 100%	GBM(1): 100%	GBM(1): 1.00	99.462%
		OV(2): 93.33%	OV(2): 93.33%	OV(2): 0.93	
		LUSC(3): 98.81%	LUSC(3): 98.81%	LUSC(3): 0.99	

This table represents the result of prediction accuracy comparison of four models on classification metrics for three classes (GBM(1), OV(2), LUSC(3)). Lasso and LDA consistently achieved 100% accuracy, precision, recall, and F1-scores among three classes, suggesting these models are highly effective at identifying patterns in the dataset. XGBoost performed slightly worse, with lower recall (93.33%) and F1-score (0.97) for OV(2), potentially due to its sensitivity to imbalanced data or reliance on boosting weaker learners. The Decision Tree model had the lowest performance (overall accuracy of 98.9%), with a drop in recall and F1-score for OV(2) (93.33%). The differences in performance are likely due to variations in model complexity, regularization techniques (e.g., Lasso's penalization), and the ability of ensemble methods like XGBoost to handle more nuanced patterns compared to a single Decision Tree.

3.2 Lasso

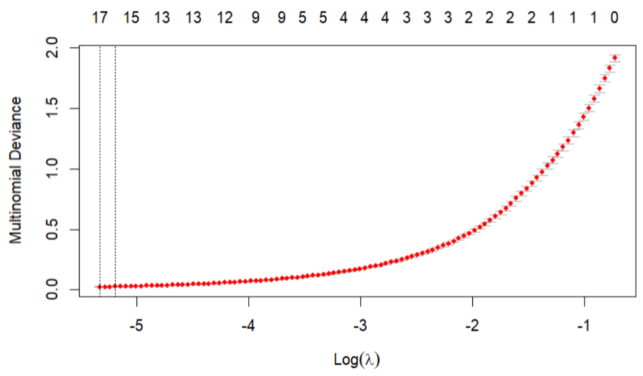

Figure 1: Regularization Path for Lasso regression

Figure 1 illustrates the impact of different regularization strengths (lambda) in the model on the error (Multinomial Deviance). Deviance is $\frac{1}{n}$ of the negative log-likelihood of the multinomial logistic regression, which measures the difference between the predicted probability distribution and the actual data distribution. Compared to the misclassification rate, Multinomial Deviance places more emphasis on the probability output of each category. It is a continuously differentiable function providing finer-grained feedback to help select more accurate lambda values.

3.3 LDA

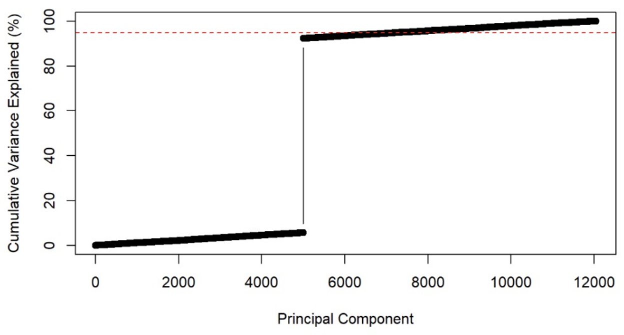

Figure 2: Variance Explained by PCA

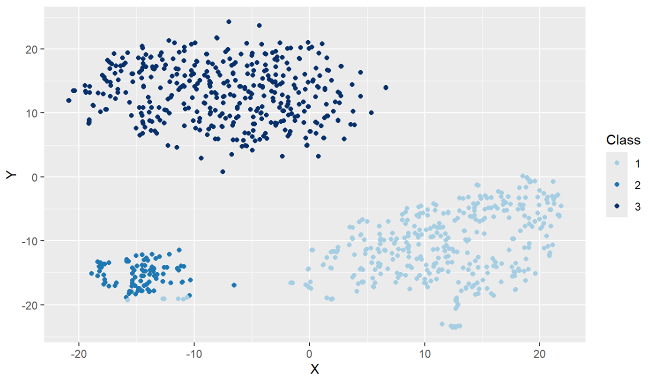

Figure 3: t-SNE Visualization of Class Separation

Due to the incredible number of predictors, using the PCA approach is necessary; by doing so, we are left with almost half of the components needed to explain 95% of the variance (Figure 2). Those data that go beyond the threshold wouldn't be considered in our case since it only contributes a little to the total variance but may bring dimensionality problems and high computational costs. From the graph (Figure 2), we can also see the generation of principle components stops increasing at a level of 5000 until the cumulative variance explained goes beyond about 90%. This means that almost 90% of the total variance could be explained by only 5000 principle components, the model only need to consider about 5000 new predictors. Furthermore, we used a t-NSE plot to check how well the classes in the dataset are separated in a reduced two-dimensional space (Figure 3). The distinct and well-separated clusters suggest that a classification model trained on this data would likely perform well, as the classes are distinguishable in the feature space. The clarity of separation also indicates that the PCA preprocessing steps have been effective in reducing dimensionality while retaining important class-specific information.

3.4 Decision Tree

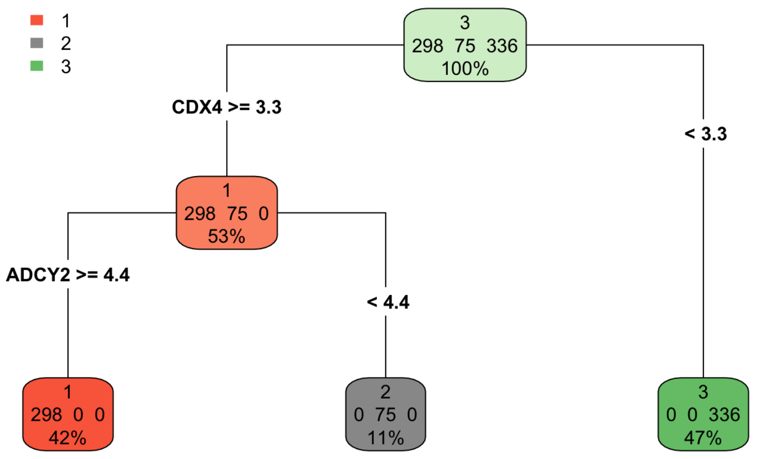

Figure 4: Full & Pruned Decision Tree

This is the model visualization for both the full tree and the pruned tree. It visualizes the classification process using two features, `CDX4` and `ADCY2`. The full tree includes only these two features due to their dominant predictive power, making them the two most significant features in the dataset. With `CDX4` being the most important feature at the root, effectively dividing the data. Once split by `CDX4`, `ADCY2` further refines the predictions for subsets of the data. At the root node, the data is split based on "`CDX4` ≥ 3.3". If true, the tree further splits on "`ADCY2` ≥ 4.4". Observations satisfying both conditions ("`CDX4` ≥ 3.3" and "`ADCY2` ≥ 4.4") are classified as class 1 (42% of the data). If "`CDX4` ≥ 3.3" but "`ADCY2` $<$ 4.4", the prediction is class 2 (11% of the data). For observations where "`CDX4` < 3.3", the tree predicts class 3 with 100% certainty (47% of the data). This streamlined structure highlights the simplicity of the dataset, as the majority of class separations are effectively captured by just these two variables. Consequently, the pruned tree remains identical to the full tree, since incorporating additional features or splits would neither enhance classification accuracy nor add value, thereby preventing overfitting and preserving the model's interpretability.

3.5 XGBoost

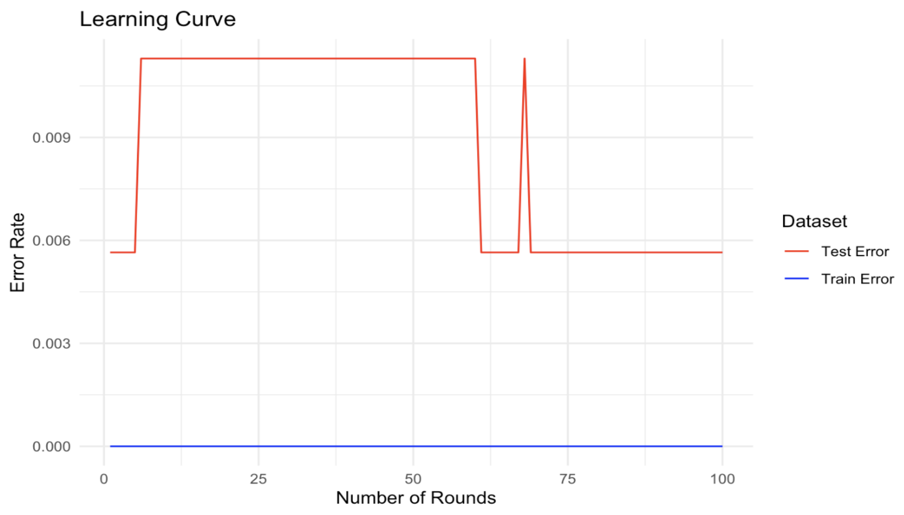

Figure 5: Learning curve of XGBoost

In 100 iterations of XGBoost, the model always fits the sub-training data perfectly which leads to the error rate for the sub-training set is always 0. As shown in Figure 5, the validation error fluctuated considerably between the 10th and 70th iterations, possibly due to parameter tuning, data imbalance, or sensitivity of the features to the target variable. After about 70 iterations, the validation error rate decreased to 0.006 and stabilized on the rest training. Overall, the error rate is quite low, which may lead to overfitting risk. To reduce the risk and increase the effectiveness, we may add standardization, change the learning rate, or stop training after the validation error stabilizes.

4 Conclusion

Our analysis and results suggest that Multinomial Logistic Lasso Regression has the highest accuracy and best generalization ability on unknown cancer classification datasets. The limited sample size of the cancer dataset used in our study may lead to bias in model performance evaluation. The interpretability of the final model obtained by Lasso needs to be supported by a professional genetic background. In practical applications, researchers can consider nonlinear relationships and combine multiple models to form a hybrid model for complicated data.

5 Discussion

First, our research has some strengths. Through the results of the whole research, four models with different perspectives and approaches represent high predictive accuracy, which leads us to evaluate the data more precisely and systematically. Meanwhile, due to its diversity, we can choose a better model to fit the data, further enhancing the conclusions' practical significance and application value. In addition, compared to high-cost algorithms from deep learning and reinforcement learning in other studies, our models are more straightforward and can be manipulated more simply in a realistic sense [**4**].

Second, different approaches have different limitations.

1. Lasso regression
   Lasso multinomial logistic regression can apply the L1 penalty to shrink the coefficients of some predictors with insignificant statistical significance to precisely 0. However, in biological genetic data, many gene predictors coordinate in the human body. Like the MTA family, the gene expression of MTA1 and MTA2 is accompanied by an increase in mRNA or protein, but the Lasso model only leaves MTA1 gene predictors [**5**]. Therefore, in prediction tasks, we cannot rely solely on the L1 regularization applied by Lasso for fitting. A more reasonable approach is to use an appropriate grouping of genes that interact with each other to apply regularization or consider nonlinear relationships.

2. LDA
   For LDA, the PCA dimensionality reduction technique used in the data preprocessing stage will remove principal components that do not contribute significantly to the overall variance of the predictors' matrix based on the contribution of components to the total variance, rather than the model's predictive ability. It cannot serve as an effective variable filter and may unconsciously exclude some predictors from the model. If there are genes with significant statistical significance in the excluded predictors, the accuracy of the entire model will be limited. The assumptions of LDA, which require the data to follow a normal distribution and each predictor to have the same variance, are difficult to satisfy. This may lead to issues such as decision boundary shifts in LDA models. Considering that the data represents the expression level of genes (all values are positive), we can use Box-Cox to adjust the data distribution to be closer to a normal distribution. To free up the assumption of sharing the same covariance matrix between categories, we can try QDA to make the decision boundary nonlinear and improve accuracy.

3. Decision Tree and XGBoost
   For decision trees and XGBoost, the tree model itself is extremely sensitive to outliers, and changes in data near the decision boundary may seriously affect the model's fit. Due to the total number of 12042 predictors in the data, the resulting Tree Model can achieve extremely high accuracy with only two nodes. A dataset that is too simple or has too much redundancy is a potential problem. Xgboost trades a large amount of computing resources for excellent predictive performance. People must be careful when adjusting hyperparameters such as regularization coefficients, learning rates, and iteration times to avoid wasting computing resources and time [**6**].

In addition, to improve the prediction accuracy, we can also consider introducing other models (such as the genetic algorithm which will process feature selection and choose the best model [**12**]) and comparing them with the currently available models to select the best one. Meanwhile, to better investigate the relevance between predictors and cancer-specific determinants, we can refer to other datasets besides the current one and combine them. Since the current dataset has only 886 observations but contains more than 12,000 predictors, the small number of observations limits our ability to draw relevant conclusions about practical implications such as cancer predisposing factors. By expanding the dataset, the robustness and generalizability of the findings can be enhanced.

References

[1] Lu, Y., & Han, J. (2003). Cancer classification using gene expression data. Information Systems, 28(4), 243-268.

[2] Jonsson Funk, M., & Landi, S. N. (2014). Misclassification in administrative claims data: quantifying the impact on treatment effect estimates. Current epidemiology reports, 1, 175-185.

[3] Chen T, Guestrin C (2016) XGBoost: a scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Dara Mining. ACM, San Francisco California USA 785-794.

[4] Li, X. (2024). From simple to complex: A complete overview of reinforcement learning. Medium. https://medium.com/@sergioli/from-simple-to-complex-a-complete-overview-of-reinforcement-learning-599a8c1ea689

[5] Kumar, R., & Wang, R. A. (2016). Structure, expression and functions of MTA genes. Gene, 582(2), 112–121. https://doi.org/10.1016/j.gene.2016.02.012

[6] Brownlee, J. (2019, June 16). What is the difference between a parameter and a hyperparameter?. MachineLearningMastery.com. https://machinelearningmastery.com/difference-between-a-parameter-and-a-hyperparameter/

[7] Weinstein, J. N., Collisson, E. A., Mills, G. B., Shaw, K. R., Ozenberger, B. A., Ellrott, K., ... & Stuart, J. M. (2013). The cancer genome atlas pan-cancer analysis project. Nature genetics, 45(10), 1113-1120.

[8] The cancer genome atlas program (TCGA). NCI.
https://www.cancer.gov/ccg/research/genome-sequencing/tcga

[9] Confusion Matrix - an overview | ScienceDirect Topics.
https://www.sciencedirect.com/topics/engineering/confusion-matrix

[10] Quantitative Finance & Algo Trading Blog by QuantInsti. (2024, September 25). Gini index: Decision tree, formula, calculator, Gini coefficient in machine learning. Quantitative Finance & Algo Trading Blog by QuantInsti.
https://blog.quantinsti.com/gini-index/

[11] What is XGBoost?. NVIDIA Data Science Glossary.
https://www.nvidia.com/en-us/glossary/xgboost/

[12] What is the genetic algorithm?.
https://www.mathworks.com/help/gads/what-is-the-genetic-algorithm.html

[13] Lu, T., Silveira, P. P., & Greenwood, C. M. T. (2023, July 3). Development of risk prediction models for depression combining genetic and early life risk factors. Frontiers.
https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2023.1143496/full

[14] Li, W., Yin, Y., Quan, X., & Zhang, H. (2019, October 9). Gene expression value prediction based on XGBoost algorithm. Frontiers.
https://www.frontiersin.org/journals/genetics/articles/10.3389/fgene.2019.01077/full

[15] Chen, S., Zhou, W., Tu, J., Li, J., Wang, B., Mo, X., Tian, G., Lv, K., & Huang, Z. (2021, January 6). A novel XGBoost method to infer the primary lesion of 20 solid tumor types from gene expression data. Frontiers.
https://www.frontiersin.org/journals/genetics/articles/10.3389/fgene.2021.632761/full

6 Code

```
install.packages("xgboost")
library(xgboost)
library(gglasso)
library(Metrics)
library(tidyverse)
library(caret)
library(dplyr)
library(car)
library(stringr)
library(glmnet)
library(grpreg)
library(MASS)
library(e1071)

# data extract
train_data_address = "train.csv"
train_data = read.csv(train_data_address)
pred_address = "test.csv"
pred_data = read.csv(pred_address)

# data cleaning
train_data = train_data %>% select(-id)
pred_data = pred_data %>% select(-id)

# prep
train _data <- train_data |> dplyr::select(sort(names(train_data)))
train_data <- train_data |> dplyr::select(cancer, everything())

set.seed(123)
n <- nrow(train_data)
test_indices <- sample(1:n, size = round(0.2 * n))
test_set <- train_data[test_indices, ]
train_set <- train_data[-test_indices, ]

train_set_y = train_set[1]
train_set_x = train_set[-1]

test_set_y = test_set[1]
test_set_x = test_set[-1]

# original training data
train_y = train_data[1]
train_x = train_data[-1]
```

6.1 Lasso

```
# Validation
test_set_x_mat=as.matrix(test_set_x)
test_set_y_val=test_set_y$cancer
```

```
train_set_x_mat=as.matrix(train_set_x)
train_set_y_fac=as.factor(train_set_y$cancer)
# Get cv object
cv_obj_val<-cv.glmnet(train_set_x_mat,train_set_y_fac,alpha=1,family="multinomial")

# Find the best lambda
best_v_lambda<-cv_obj_val$lambda.min
cat("Best lambda:",best_v_lambda,"\n")
plot(cv_obj_val)
# Apply to the eln model
eln_val_model<-glmnet(train_set_x_mat,train_set_y_fac,alpha = 1,lambda =best_v_lambda,
family = "multinomial")
# Predict for the validation set
eln_val_pred<-predict(eln_val_model,newx=test_set_x_mat,s=best_v_lambda,type="class")
eln_val_prob<-predict(eln_val_model,newx=test_set_x_mat,s=best_v_lambda,type="response")
eln_acc<-mean(eln_val_pred==test_set_y_val)
eln_v_probs_df<-as.data.frame(eln_val_prob)
colnames(eln_v_probs_df)<-levels(train_set_y)

# Print The accuracy
print(eln_acc)

## Print The Confusion Matrix
true_label=test_set_y_val
pred_class=as.numeric(eln_val_pred)
confusion_matrix=confusionMatrix(factor(as.vector(eln_val_pred)),factor(true_label))
print(confusion_matrix)
#Full data
train_x_mat=as.matrix(train_x)
train_y_fac=factor(train_y$cancer)
pred_x_mat=as.matrix(pred_data)
cv_obj<-cv.glmnet(train_x_mat,train_y_fac,alpha=1,nfolds=10,family="multinomial")

# Find the best lambda
best_lambda<-cv_obj$lambda.min
cat("Best lambda:",best_lambda,"\n")
plot(cv_obj)
eln_model<-glmnet(train_x_mat,train_y_fac,alpha=1,lambda=best_lambda,
family="multinomial")
eln_pred<-predict(eln_model,newx=pred_x_mat,s=best_lambda,type="class")
eln_prob<-predict(eln_model,newx=pred_x_mat,s=best_lambda,type="response")
eln_probs_df<-as.data.frame(eln_prob)
colnames(eln_probs_df)<-levels(train_y_fac)

# Check For The Coefficients
eln_model$a0
eln_model$df
eln_model$lambda
coefficients<-coef(eln_model)
coeff1<-coefficients[[1]]
coeff1<-as.matrix(coeff1)
nonzero_coeffs1<-coeff1[coeff1!=0]
```

```
# print(nonzero_coeffs1)
coeff2<-coefficients[[2]]
coeff2<-as.matrix(coeff2)
nonzero_coeffs2<-coeff2[coeff2!=0]
# print(nonzero_coeffs2)
coeff3<-coefficients[[3]]
coeff3<-as.matrix(coeff3)
nonzero_coeffs3<-coeff3[coeff3!=0]
# print(nonzero_coeffs3)
nonzero_coefficients<-lapply(coefficients, function(coef_matrix) {
  dense_matrix<-as.matrix(coef_matrix)
  nonzero_indices<-which(dense_matrix!=0)
  nonzero_values<-dense_matrix[nonzero_indices]
  nonzero_names<-rownames(dense_matrix)[nonzero_indices]
  data.frame(Variable=nonzero_names, Coefficient=nonzero_values)
})
print(nonzero_coefficients)

data=read.csv("result_lasso.csv")
data$cancer<-eln_pred
write.csv(data, "result_lasso.csv", row.names = FALSE)
```

6.2 LDA with PCA

```
preProcessObj <- preProcess(train_data[-1], method = c("center", "scale", "pca"))
trainDataProcessed <- predict(preProcessObj, train_data)
testDataProcessed <- predict(preProcessObj, pred_data)

ldaModel <- lda(cancer ~., data = trainDataProcessed)

ldaPred <- predict(ldaModel, testDataProcessed)

# Extract the posterior probabilities for each type (1, 2, and 3)
posteriorProbabilities <- ldaPred$posterior

# Add row and column names for clarity
colnames(posteriorProbabilities) <- c("Type 1", "Type 2", "Type 3")
mean_probabilities <- colMeans(posteriorProbabilities)

# Print the mean probabilities
print("Mean Probabilities for Each Type:")
print(mean_probabilities)

# Bar plot of mean probabilities
barplot(mean_probabilities, main = "Mean Posterior Probabilities for Each Type", xlab = "Cancer Types",
ylab = "Mean Probability", col = c("lightblue", "lightblue", "lightblue"), names.arg = c("Type 1", "Type 2",
"Type 3"))

confusionMatrix <- table(Predicted = ldaPred$class, Actual = test_set$cancer)
print(confusionMatrix)
```

```
accuracy <- sum(diag(confusionMatrix)) / sum(confusionMatrix)
print(paste("Accuracy:", accuracy))

library(Rtsne)

set.seed(123)
# Perform tSNE on PCAtransformed data
tsne <- Rtsne(trainDataProcessed[, -1], dims = 2, perplexity = 30)
tsne_df <- data.frame(X = tsne$Y[, 1], Y = tsne$Y[, 2], Class = as.factor(train_data$cancer))

# Plot t-SNE
ggplot(tsne_df, aes(x = X, y = Y, color = Class)) + geom_point() + scale_color_manual(values = c("#a6cee3",
"#1f78b4", "#08306b")) + ggtitle("t-SNE Visualization of Class Separation")

# Extract standard deviations from preProcessObj
sdev <- preProcessObj$std
if (!is.null(sdev) && all(is.finite(sdev))) { variance_explained <- sdev^2 / sum(sdev^2)
cumulative_variance <- cumsum(variance_explained) * 100

# Plot cumulative variance plot(cumulative_variance, type = "b", xlab = "Principal Component", ylab
= "Cumulative Variance Explained (%)", main = "Variance Explained by PCA")
abline(h = 95, col = "red", lty = 2) } else { print("Standard deviations for PCA are missing or invalid.") }
test_data_processed <- predict(preProcessObj, pred_data)

# Make predictions using the trained LDA model lda_predictions <- predict(ldaModel, test_data_processed)

# Combine IDs with predictions output <- data.frame(id = pred_data$id, cancer = lda_predictions$class)

# Write the predictions to a CSV file
output_file <- "predictions.csv"
write.csv(output, file = output_file, row.names = FALSE)

# Confirmation message
cat("Predictions saved to", output_file, "\n")

# Train SVM with linear kernel (hinge loss penalty)
svm_model <- svm( cancer ~., data = train_data, kernel = "linear", cost = 2 )

# View model summary
svm_predictions <- predict(svm_model, pred_data)
svm_predictions <- as.integer(as.character(svm_predictions))

# Combine IDs with predictions
output <- data.frame(id = pred_data$id, cancer = svm_predictions)

# Write the predictions to a CSV file
output_file <- "svm_predictions.csv"
write.csv(output, file = output_file, row.names = FALSE)

# Confirmation message cat("Predictions saved to", output_file, "\n")
```

6.3 Decision Tree

```
library(rpart)
library(rpart.plot)

# Build the model
tree_model <- rpart(cancer ~., data = train_set, method = "class")

# View the tree structure
print(tree_model)

# Plot the decision tree
rpart.plot(tree_model, type = 4, extra = 101, fallen.leaves = TRUE)

# Create a confusion matrix
library(caret)
confusion_matrix = confusionMatrix(factor(as.vector(predictions)), factor(test_set$cancer))
print(confusion_matrix)

# Pruned Tree
printcp(tree_model)

# Prune the tree using the optimal cp
optimal_cp <-tree_model$cptable[which.min(tree_model$cptable[, "xerror"]), "CP"]
pruned_tree <- prune(tree_model, cp = optimal_cp)

# Visualize the pruned tree
rpart.plot(pruned_tree, type = 4, extra = 101, fallen.leaves = TRUE)

confusion_matrix = confusionMatrix(factor(as.vector(predictions)), factor(test_set$cancer))
print(confusion_matrix)

prediction_full <- predict(tree_model, pred_data , type = "class")
pred_result_f_final <- prediction_full

# Save the results to a CSV file
result = unlist(pred_result_f_final)
data = read.csv("sub_py_2.csv")
data$cancer <- result
write.csv(data, "sub_py_2.csv", row.names = FALSE)
```

6.4 XGBoost

```
train_set$cancer <- as.factor(train_set$cancer)
test_set$cancer <- as.factor(test_set$cancer)

# Convert the predictor variables to a numeric matrix
train_matrix <- as.matrix(train_set[,-1])
test_matrix <- as.matrix(test_set[,-1])

# Convert the target variable to an integer (for multi-class classification)
```

```
train_label <- as.integer(train_set$cancer) - 1
test_label <- as.integer(test_set$cancer) - 1

# Create xgboost DMatrix objects
dtrain <- xgb.DMatrix(data = train_matrix, label = train_label)
dtest <- xgb.DMatrix(data = test_matrix, label = test_label)

# Train the XGBoost model
params <- list(objective = "multi:softmax", num_class = length(unique(train_label)), max_depth = 6, eta
= 0.1, nthread = 2)

# Train the model (adjust number of rounds as needed)
xgb_model <- xgboost(params = params, data = dtrain, nrounds = 100, verbose = 1)

# Predict for the test set
pred_test <- predict(xgb_model, newdata = dtest)

# Convert the predicted values back to the original class labels
pred_test_class <- as.factor(pred_test + 1)

# Evaluate performance (Confusion Matrix)
confusion_matrix_1 <- confusionMatrix(factor(as.factor(pred_test_class)), factor(test_set$cancer))
print(confusion_matrix_1)
confusion_matrix <- table(Predicted = pred_test_class, Actual = test_set$cancer)
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
print(confusion_matrix)
print(accuracy)
confusion_matrix_1$byClass

# Ensure both train_set and pred_data have the same columns (excluding target variable 'cancer')
train_columns <- colnames(train_set)[-1]
pred_data <- pred_data[, train_columns]

# Convert pred_data to matrix form
pred_data_matrix <- as.matrix(pred_data)

# Create DMatrix for prediction
dpred <- xgb.DMatrix(data = pred_data_matrix)

# Make predictions using the trained xgboost model
pred_result <- predict(xgb_model, newdata = dpred)

# Convert the numeric predictions back to factor
pred_result_class <- as.factor(pred_result + 1)

# Train model with evaluation log
watchlist <- list(train = dtrain, test = dtest)
xgb_model <- xgb.train(params = params, data = dtrain, nrounds = 100, watchlist = watchlist, eval_metric
= "merror", verbose = 1)

# Extract evaluation metrics
eval_log <- xgb_model$evaluation_log
```

```
# Plot learning curve
library(ggplot2)
ggplot(data = eval_log, aes(x = iter)) + geom_line(aes(y = train_merror, color = "Train Error")) + geom_line(aes(y
= test_merror, color = "Test Error")) + theme_minimal() + labs(title = "Learning Curve", x = "Number of
Rounds", y = "Error Rate") + scale_color_manual(name = "Dataset", values = c("Train Error" = "blue",
"Test Error" = "red"))

# Save the results to a CSV file
data <- read.csv("sub_py_3.csv")
data$cancer <- pred_result_class
write.csv(data, "sub_py_3.csv", row.names = FALSE)
```