Comparative Analysis of Machine Learning Models for Thyroid Cancer Recurrence Prediction

(Mentor: Dr. Marly Gotti)

Anay Aggarwal, Ekam Kaur, and Susie Lu

MIT PRIMES-USA

August 12, 2024

Table of Contents

Introduction

- Cancer ranks as one of the top causes of death globally. Approximately 1 in 3 people will be diagnosed with cancer during their lifetime, and of those, 1 in 6 will succumb to the disease.
- Thyroid cancer is among the most prevalent endocrine malignancies worldwide.
- Differentiated Thyroid Cancer (DTC) represents the majority of cases of thyroid cancer. Our project aims to compare the performance of six machine learning models for predicting DTC recurrence.
- Traditional approaches for predicting DTC recurrence often lack the precision for personalized treatment planning, so there has been interest in leveraging machine learning instead.

Machine Learning Algorithms Used

Algorithm	A Key Advantage
Support Vector Machines (SVM)	Effective for high-dimensional data
Random Forests (RF)	Ensemble method via bagging
Extreme Gradient Boosting (XGBoost)	Ensemble method via boosting
Artificial Neural Networks (ANN)	Model complex non-linear relationships
K-Nearest Neighbors (KNN)	Efficient, based on space proximity
Logistic Regression (LR)	High interpretability, outputs predicted probabilities

We implemented these algorithms in R, and we will show some code in the presentation for interest.

Our Github Repository

- Our code and the details of the models are in a GitHub repository.
- `https://marlycormar.github.io/primes-research-project-2024/`
- For reproducibility, we use
  - Quarto manuscript structure
  - `renv` environment
  - GitHub project management

Overview of Dataset

- Our study focuses on the "Differentiated Thyroid Cancer Recurrence" dataset from the UCI Machine Learning Repository.
- For each of the 383 patients, 17 variables pertinent to thyroid cancer are recorded.
  - Age, Gender, Smoking, History of smoking, History of radiotherapy
  - Thyroid function, Physical examination, Adenopathy, Pathology, Focality
  - Risk Assessment, Cancer Stage, T, N, M, Response

  Aim to predict: whether DTC recurrence occurred.

Data Preprocessing

The dataset has 16 categorical variables and 1 numerical variable.

Data Preprocessing

The dataset has 16 categorical variables and 1 numerical variable.

Remove duplicate observations

Change categorical variables into factors

```
#' Load raw data.
cleaned_data <-
  readr::read_csv(here::here('data/raw-data.csv')) |>
  dplyr::distinct() |>
  dplyr::rename(`Hx Radiotherapy` = 'Hx Radiothreapy') |>
  dplyr::mutate(Gender = ifelse(Gender == 'F', 'Female', 'Male')) |>
  dplyr::mutate(
    Gender = factor(Gender, levels = c('Female', 'Male')),
    Smoking = factor(Smoking, levels = c('Yes', 'No')),
    `Hx Smoking` = factor(`Hx Smoking`, levels = c('Yes', 'No')),
    `Hx Radiotherapy` = factor(`Hx Radiotherapy`, levels = c('Yes', 'No')),
    `Thyroid Function` = factor(
      `Thyroid Function`,
      levels = c('Euthyroid', 'Clinical Hyperthyroidism',
                 'Subclinical Hyperthyroidism', 'Clinical Hypothyroidism',
                 'Subclinical Hypothyroidism')),
    `Physical Examination` = factor(`Physical Examination`,
                                    levels = c('Normal', 'Diffuse goiter',
                                               'Single nodular goiter-right',
                                               'Single nodular goiter-left',
                                               'Multinodular goiter')),
    Adenopathy = factor(Adenopathy,
                        levels = c('No', 'Right', 'Left', 'Bilateral',
                                   'Posterior', 'Extensive')),
    Pathology = factor(
      Pathology,
      levels = c('Papillary', 'Micropapillary', 'Follicular',
                 'Hurthel cell')),
    Focality = factor(Focality, levels = c('Uni-Focal', 'Multi-Focal')),
    `T` = factor(`T`, levels = c('T1a', 'T1b', 'T2', 'T3a', 'T3b', 'T4a',
                                 'T4b')),
    N = factor(N, levels = c('N0', 'N1b', 'N1a')),
    M = factor(M, levels = c('M0', 'M1')),
    Stage = factor(Stage, levels = c('I', 'II', 'III', 'IVA', 'IVB')),
    Response = factor(
      Response,
      levels = c('Excellent', 'Biochemical Incomplete',
                 'Structural Incomplete', 'Indeterminate')),
    Risk = factor(Risk, levels = c('Low', 'Intermediate', 'High')),
    Recurred = factor(Recurred, levels = c('Yes', 'No'))
  )
```

Exploratory Data Analysis

- After removing duplicates, the dataset has 364 observations.
- Figure 1: 80.5% of the patients are female, while 19.5% are male. Males are more likely to have DTC recurrence.
- Figure 2: In general, older patients are more likely to have DTC recurrence.

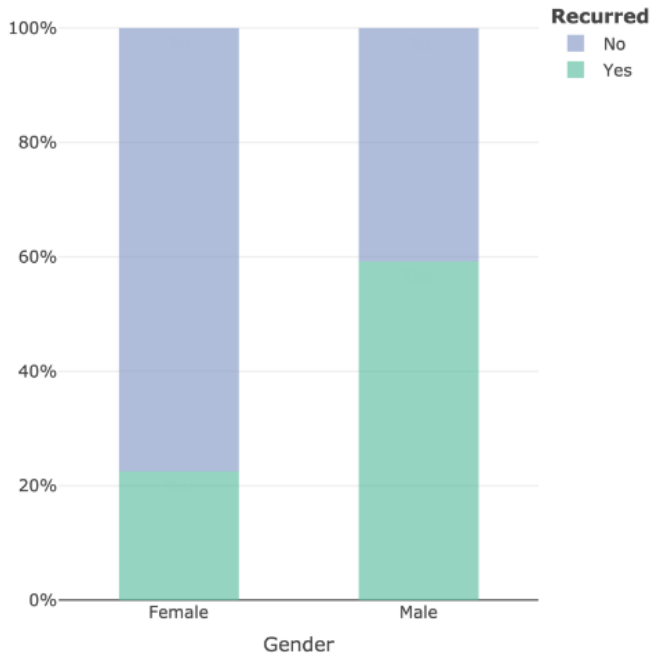


Figure 1: Gender Distribution by Cancer Recurrence

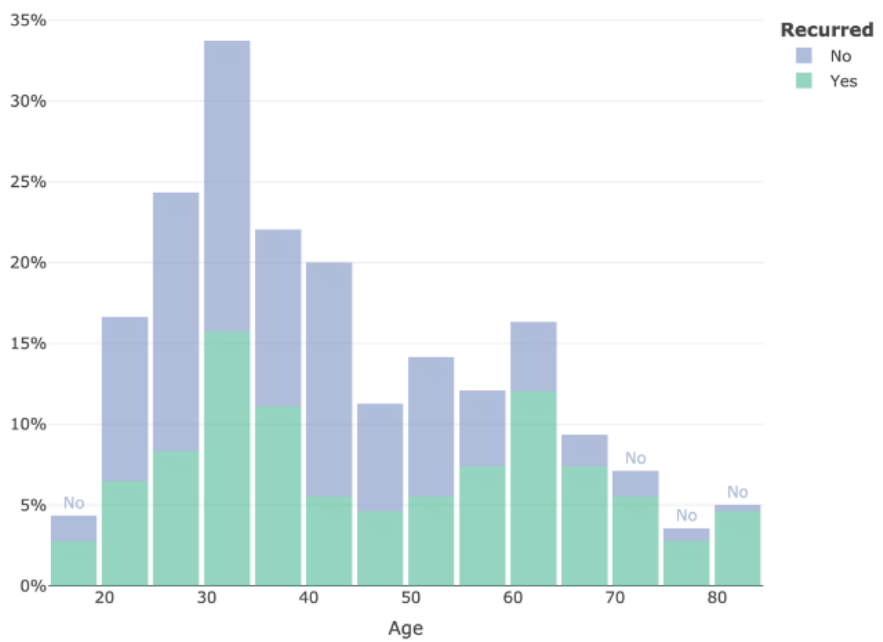


Figure 2: Age Distribution by Cancer Recurrence

Exploratory Data Analysis

- Besides age, the rest of the features are categorical.
- Adenopathy: the presence of swollen lymph nodes during physical examination.

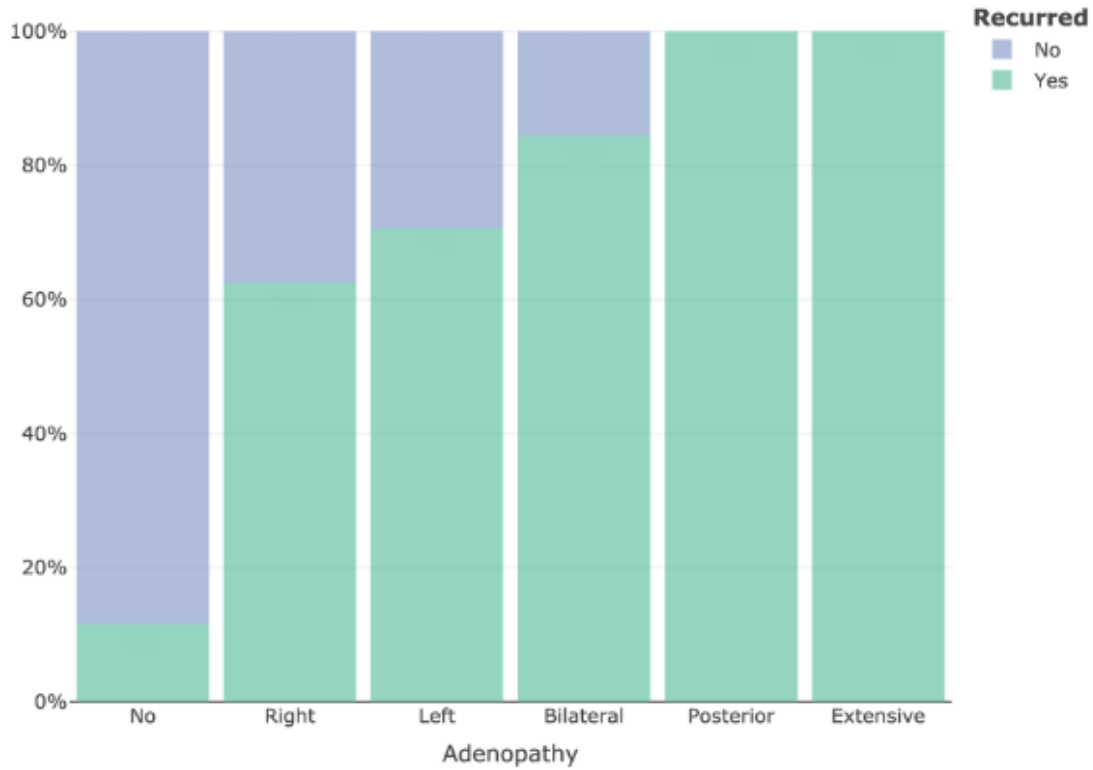


Figure 3: Adenopathy Distribution by Cancer Recurrence

Model Training & Testing

What is the process of creating and testing a machine learning model?

- Train set: this is the part of the dataset used for training/fitting the model. The goal is to use the training set (the predictors and response variable) and determine a model in order to minimize some *cost/error function*.
- Test set: once the model parameters are finalized, its performance is computed on the test set. This is done by computing several different performance metrics, which we will discuss below.

The goal is to train a model that isn't too specific to the training set (i.e. minimizes *overfitting*) while maintaining high accuracy (i.e. minimizes *underfitting*).

Model Training: Cross Validation

A useful way to better test and understand a model's fit is to use *k-fold cross validation*.

k-fold cross validation

k-fold cross validation is done as follows.

- We split the training set into k parts (or "folds").
- We run the training process using first $k-1$ folds as the training set and the last fold as the test set.
- Repeat using each fold as the test set.

This runs the training process k times, obtaining k different model parameters and performances.

- Applying k-cross validation is useful to evaluate the model's performance in a more robust way – and also helps detect overfitting.

Model Training: Hyperparameters and Regularization

Models usually have *hyperparameters*, which are model parameters that need to be set beforehand.
Finding the optimal hyperparameters is done through *tuning* which refers to testing multiple options for the hyperparameters.
An efficient way to tune hyperparameters is using the k-fold cross validation method described above.

- For each combination of hyperparameters, we run through the k-fold cross validation process. The combination with the highest average metric across the k-folds is chosen.

Regularization is another tool to combat overfitting

- The idea is that while learning from training set, we have another goal of keeping the parameters small.
- The weightage which we place on the "keep parameters small" term is referred to as *penalty*.

Model Training

For the DTC dataset:

- We split the dataset into a training (75%) set and a test (25%) set using a random generator.
- The training data is further separated into 10 folds for cross-validation.

```
# Split data into training and test sets.
set.seed(314)
data_split <- rsample::initial_split(cleaned_data)
train_data <- rsample::training(data_split)
test_data <- rsample::testing(data_split)

# Split the training data into 10-folds for cross-validation.
set.seed(3145)
data_cross_val <- rsample::vfold_cv(train_data)

# Set aside the outcome column of the sample test data.
test_outcome <- factor(test_data$Recurred)
```

Model Training: Pre-processing Steps

Additionally, we apply the following pre-processing steps to allow for better training.

Correlation

Roughly speaking, the correlation between two variables refers to the degree in which they change together.

- A high correlation indicates that fluctuation in one variable implies consistent fluctuation in the other.
- A low correlation means that fluctuation in one variable does not really affect the other variable.

This is measured on an index from -1 to 1, indicating the strength and direction of the correlation.

- remove the minimum number of features having very strong correlations – this means removing redundant information.
- the numeric features should be normalized – this means to transform the mean and standard deviation of each column to 0 and 1, respectively.
- the categorical variables should be transformed into numerical variables.

Metrics for Model Performance

Once the finalized models are applied to the test set, we look at comparison table of the actual truth and predicted values.

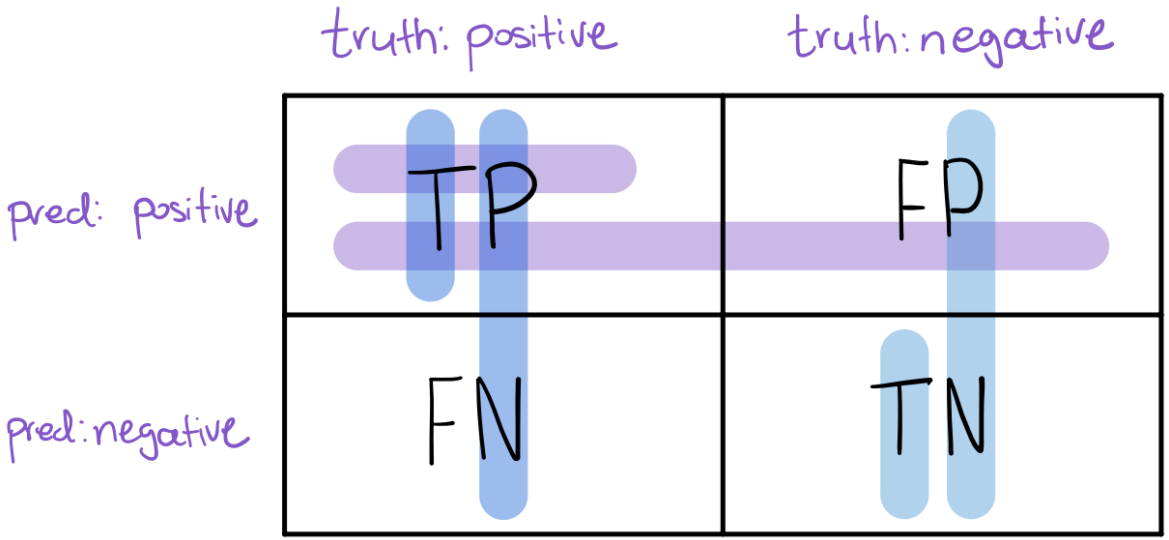
	Truth: Positive	Truth: Negative
Prediction: Positive	*TP*	*FP*
Prediction: Negative	*FN*	*TN*

- True Positive (TP): actually positive and model predicts positive.
- True Negative (TN): actually negative and model predicts negative.
- False Positive (FP): actually negative but model falsely predicts positive.
- False Negative (FN): actually positive but model falsely predicts negative.

Metrics for Model Performance

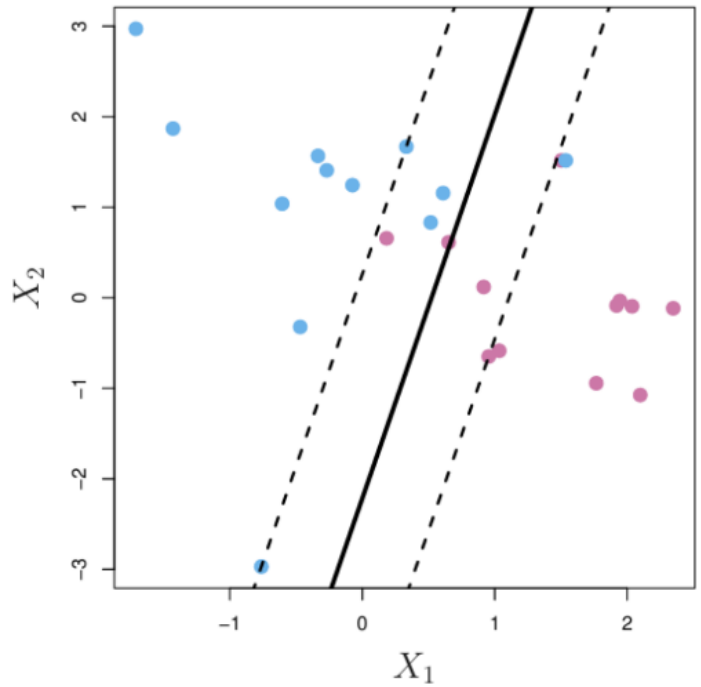
For each model, after we train it and apply it to make predictions on the dataset, we evaluate its performance using the following four metrics.

- Accuracy: proportion of correct predictions, or $\frac{TN+TP}{TN+TP+FN+FP}$.
- Precision: proportion of positive classified observations that are actually positive, or $\frac{TP}{TP+FP}$.
- Recall: proportion of actual positive observations correctly classified as positive, or $\frac{TP}{TP+FN}$.
- Specificity: proportion of actual negative observations correctly classified as negative, or $\frac{TN}{TN+FP}$.



Support Vector Machine (SVM)

- Powerful supervised learning algorithm: we have a response variable of interest.
- SVM finds the hyperplane that best separates data points of different classes (blue and purple) in a high-dimensional space.
- The optimal hyperplane is determined by maximizing the margin between the two classes.



Support Vector Machine (SVM)

The data can have a linear decision boundary or a non-linear one.

For linear decision boundaries

Given n training observations $X_1, \ldots, X_n$, the best hyperplane $\beta_0 + \beta_1 X_{i1} + \cdots + \beta_p X_{ip} = 0$ is the solution of an optimization problem.

Maximize the margin M subject to the conditions:

1. $\sum_{j=1}^{p} \beta_j^2 = 1$
2. $y_i(\beta_0 + \beta_1 X_{i1} + \cdots + \beta_p X_{ip}) \geq M(1 - \varepsilon_i)$, where the ε_i's are small error terms.
3. $\varepsilon_i \geq 0$ and $\sum_{i=1}^{n} \varepsilon_i \leq C$, where C is a user-chosen parameter called the cost.

The larger the cost C, the higher tolerance we have for observations that are on the wrong side of the margin.

Support Vector Machine (SVM)

For linear decision boundaries

Denote the inner product of two p-vectors a and b by $\langle a, b\rangle = \sum_{i=1}^{p} a_i b_i$. The classification produced by SVM can also be written as

$$f(X) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle X, X_i\rangle.$$

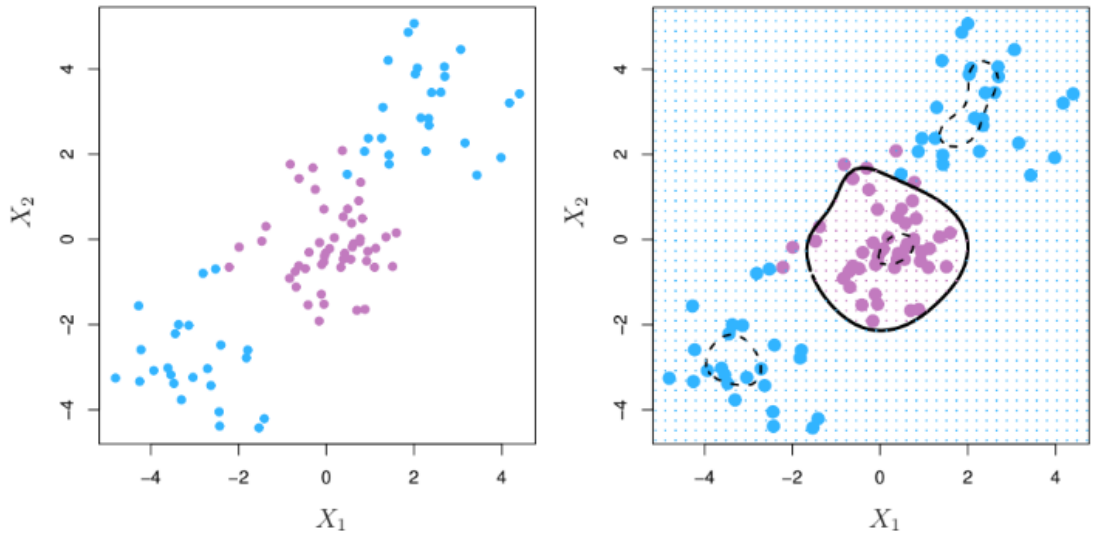
- The test observation X is put in one category if $f(X) \geq 0$ and put in the other category if $f(X) < 0$.
- The coefficients $\beta_0, \alpha_1, \ldots, \alpha_n$ can be written in terms of the $\binom{n}{2}$ inner products $\langle X_i, X_j\rangle$.

Support Vector Machine (SVM)

For non-linear decision boundaries

SVM uses kernels, which are generalizations of the inner product:
$f(X) = \beta_0 + \sum_{i=1}^{n} \alpha_i K(X, X_i)$.

- A kernel maps input features into a high-dimensional space.
- Radial basis function kernel: $K(X, X_i) = e^{-\frac{||X - X_i||^2}{2\sigma^2}}$, where σ is a parameter that controls the influence of individual training samples.



Support Vector Machine (SVM)

- We create a SVM model specification and workflow.
- Hyperparameters: σ (or `rbf_sigma`) and C (or `cost`).
- We use the `tune::tune()` function to identify the optimal values of these parameters in terms of model accuracy.

Support Vector Machine (SVM)

- We create a SVM model specification and workflow.
- Hyperparameters: σ (or `rbf_sigma`) and C (or `cost`).
- We use the `tune::tune()` function to identify the optimal values of these parameters in terms of model accuracy.

```
# Create model specification.
svm_model_spec <-
  parsnip::svm_rbf(
    cost = tune::tune(),
    rbf_sigma = tune::tune()
  ) |>
  parsnip::set_engine('kernlab') |>
  parsnip::set_mode('classification')

# Create model workflow.
svm_workflow <- workflows::workflow() |>
  workflows::add_model(svm_model_spec) |>
  workflows::add_recipe(data_rec)
```

Support Vector Machine (SVM)

We apply our selected model to the test set.

Table 3: SVM Performance Metrics: Accuracy, Precision, Recall, and Specificity.

Metric	**Value**
Accuracy	78%
Precision	23.1%
Recall	100%
Specificity	76.5%

SVM has perfect recall of 100%, so it is exceptionally effective at capturing all actual positive cases, which is important when the primary goal is to ensure that no positive cases are missed.

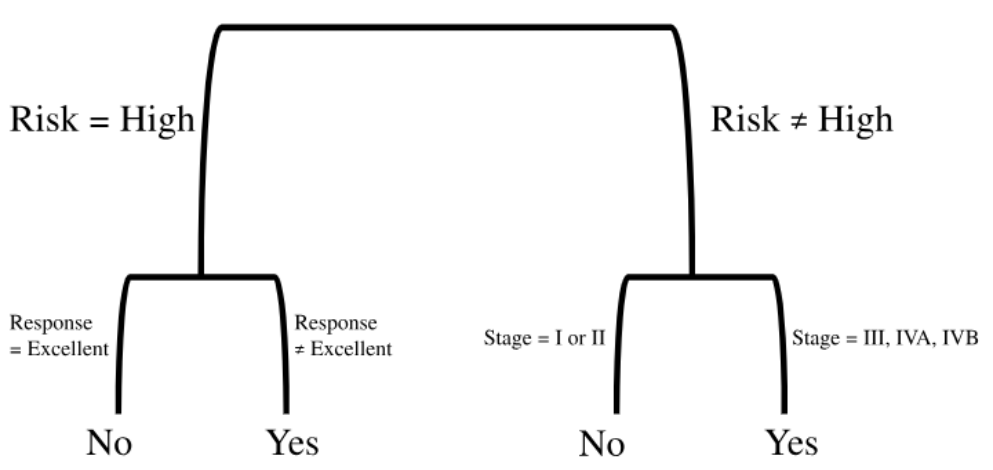
Random Forest (RF)

An ensemble learning method is an approach that combines several simple models to obtain a single powerful model. RF is an ensemble learning method that

- constructs multiple decision trees
- then outputs the mode of the classifications given by the individual trees.

Decision Tree

- Each decision tree splits the predictor space using recursive binary splits.
- Splits are chosen to minimize an error criterion, such as the Gini index (which is small if the observations in a branch are mostly from the same class).



Random Forest (RF)

Individual decision trees can have high variance, so small changes in the training data can lead to very different tree structures. RF decreases the variance through two techniques.

RF: Bagging

Intuition:

- Given n independent observations, each with variance σ^2, the variance of the average of the observations is σ^2/n.
- Averaging multiple observations reduces variance.

The approach in RF:

- Multiple trees are trained on different samples of the data.
- The final prediction is made by taking the mode of the predictions of all trees.

RF: Random selection of features for each split

- Approximately $\sqrt{p}$ features are used, where p is the total number of features.
- This reduces the correlation between the predictions of different trees.

Random Forest (RF)

- Hyperparameters: `trees` (number of trees) and `min_n` (minimum number of observations required in a terminal node)
- We tune the parameters and then apply the selected model to the test set.

Table 7: Random Forest Performance Metrics: Accuracy, Precision, Recall, and Specificity.

Metric	Value
Accuracy	94.5%
Precision	84.6%
Recall	95.7%
Specificity	94.1%

- RF has higher accuracy, precision, and specificity than SVM.
- RF has slightly lower recall than SVM.

Extreme Gradient Boosting (XGBoost)

- XGBoost, similar to RF, is an ensemble learning technique involving decision trees.
- It sequentially builds decision trees to improve predictions. Each new tree is a refinement of the previous tree, with the goal to optimize a loss function.
- It is known for high performance, scalability, and accuracy.

Extreme Gradient Boosting (XGBoost)

XGBoost uses multiple hyperparameters.

- `tree_depth`: Controls the maximum depth of each tree, impacting the model's complexity.

Extreme Gradient Boosting (XGBoost)

XGBoost uses multiple hyperparameters.

- `tree_depth`: Controls the maximum depth of each tree, impacting the model's complexity.
- `min_n`: Specifies the minimum number of observations that must exist in a terminal node (as with RF).

Extreme Gradient Boosting (XGBoost)

XGBoost uses multiple hyperparameters.

- `tree_depth`: Controls the maximum depth of each tree, impacting the model's complexity.
- `min_n`: Specifies the minimum number of observations that must exist in a terminal node (as with RF).
- `loss_reduction`: Sets the minimum improvement in the loss function required to create a new split in a tree, acting as a control mechanism to prevent overfitting.

Extreme Gradient Boosting (XGBoost)

- `sample_size`: Determines the fraction of the training data used for fitting each individual tree, introducing randomness and preventing overfitting.

Extreme Gradient Boosting (XGBoost)

- `sample_size`: Determines the fraction of the training data used for fitting each individual tree, introducing randomness and preventing overfitting.
- `mtry`: Sets the number of features considered when looking for the best split, adding variability to enhance generalization.

Extreme Gradient Boosting (XGBoost)

- `sample_size`: Determines the fraction of the training data used for fitting each individual tree, introducing randomness and preventing overfitting.
- `mtry`: Sets the number of features considered when looking for the best split, adding variability to enhance generalization.
- `learn_rate`: Also known as the shrinkage parameter, controls the rate at which the model learns. Smaller learning rates can lead to better performance by allowing the model to learn more slowly and avoid overfitting.

Extreme Gradient Boosting (XGBoost)

We set up and specify our model with the aforementioned hyperparameters:

Extreme Gradient Boosting (XGBoost)

We set up and specify our model with the aforementioned hyperparameters:

```
# Create model specification.
xgboost_model_spec <-
  boost_tree(
    trees = 1000,
    tree_depth = tune(),
    min_n = tune(),
    loss_reduction = tune(),
    sample_size = tune(),
    mtry = tune(),
    learn_rate = tune()
  ) |>
  set_engine('xgboost') |>
  set_mode('classification')

# Create model workflow.
xgboost_workflow <- workflows::workflow() |>
  workflows::add_model(xgboost_model_spec) |>
  workflows::add_recipe(data_rec)
```

Extreme Gradient Boosting (XGBoost)

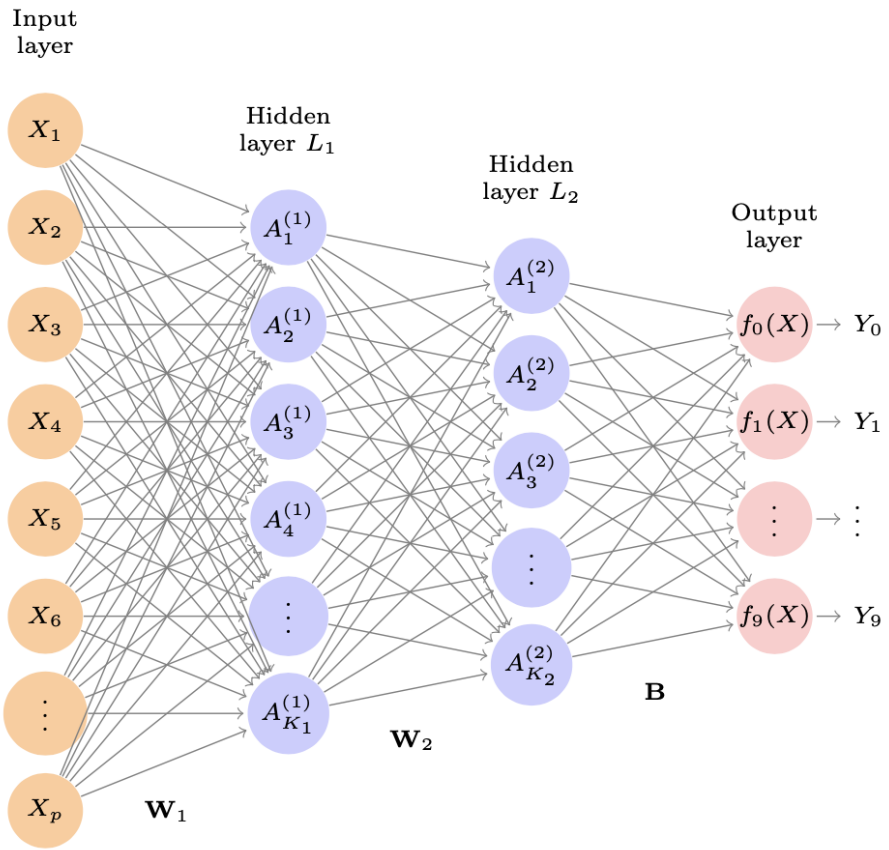
Finally, we assess our model on the test data:

Metric	Value
Accuracy	94.5%
Precision	84.6%
Recall	95.7%
Specificity	94.1%

The model has high recall, meaning that it accurately identifies true positives.

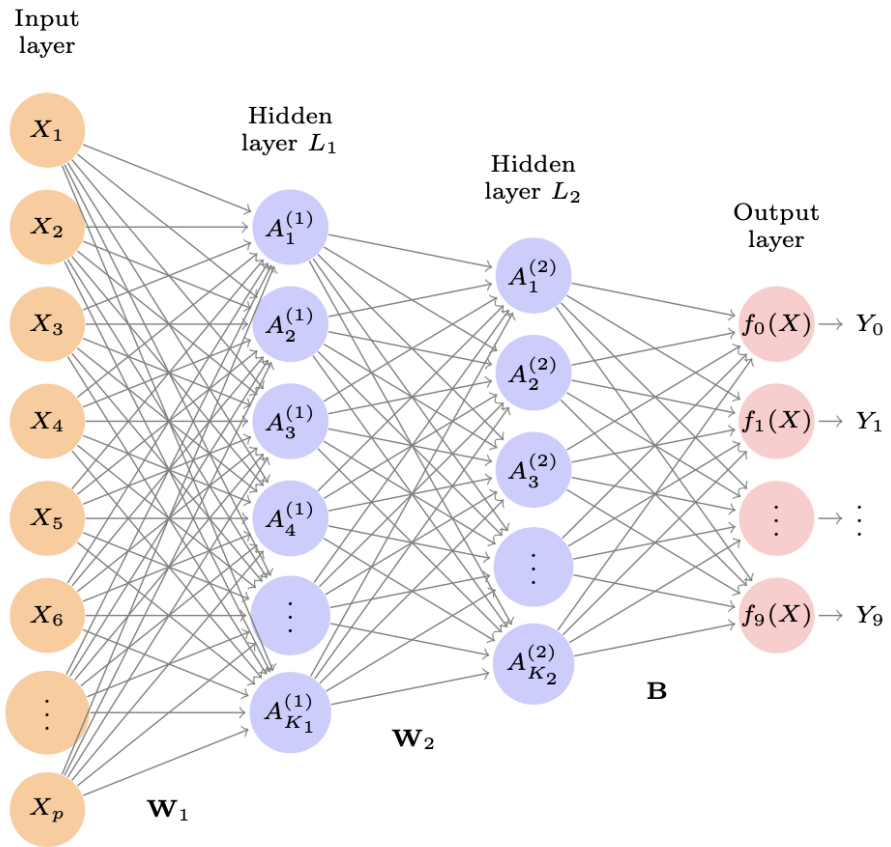
Artificial Neural Network (ANN)

- ANNs are computational models inspired by the brain, composed of interconnected nodes (neurons) in layers.
- An ANN is composed of multiple layers, including an input layer, one or more hidden layers, and an output layer.



Artificial Neural Network (ANN)

- The input layer receives the raw data, the hidden layers process the data through various transformations, and the output layer produces the final prediction.
- Each connection between neurons has an associated weight, and each neuron has a bias term. These parameters are optimized during training.



Artificial Neural Network (ANN)

- The training process of an ANN involves forward propagation, where input data is passed through the network layer by layer.
- Each neuron applies an activation function to compute its output, introducing non-linearity to help the network learn complex patterns.
- The loss, or error, between the network's output and the true target values is calculated using a loss function.
- Through backpropagation, the loss is propagated backward through the network, and the weights and biases are adjusted using an optimization algorithm like gradient descent.

Artificial Neural Network (ANN)

- We create an ANN model specification and workflow with hyperparameters `epochs`, `hidden_units`, and `penalty`. We optimize these parameters by tuning.

Artificial Neural Network (ANN)

- We create an ANN model specification and workflow with hyperparameters `epochs`, `hidden_units`, and `penalty`. We optimize these parameters by tuning.

```
# Create model specification.
ann_model_spec <-
  parsnip::mlp(
    epochs = tune::tune(),
    hidden_units = tune::tune(),
    penalty = tune::tune(),
    learn_rate = 0.1
  ) |>
  parsnip::set_engine('nnet') |>
  parsnip::set_mode('classification')

# Create model workflow.
ann_workflow <- workflows::workflow() |>
  workflows::add_model(ann_model_spec) |>
  workflows::add_recipe(data_rec)
```

Artificial Neural Network (ANN)

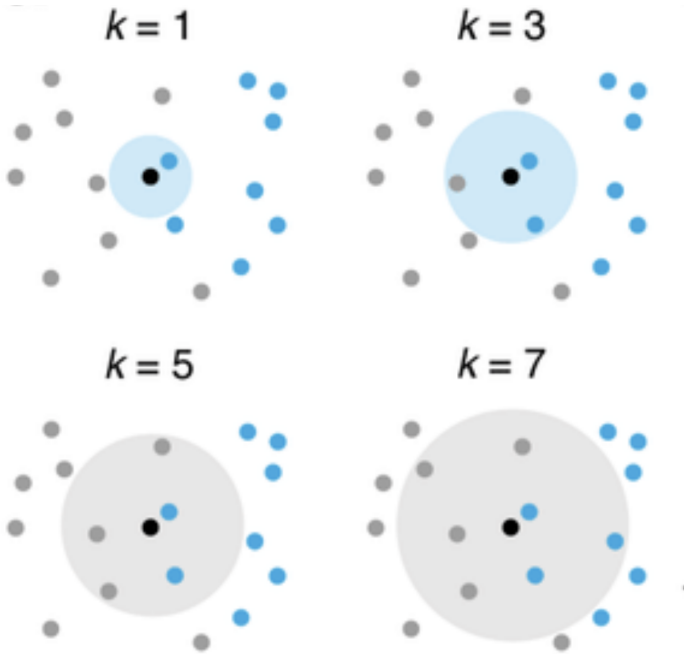
We apply our selected model to the test data.

Metric	Value
Accuracy	92.3%
Precision	84.6%
Recall	88%
Specificity	93.9%

It has a 93.9% specificity, meaning that it can accurately predict true negatives. Additionally, the ANN is more consistent than the previous models.

K-Nearest Neighbors (KNN)

- Simple algorithm that classifies a given sample based on the proximity to the training data.
- A point X is classified as the most common class label among its closest k neighbors, for some fixed k and fixed distance measure.
- We can also choose how to weigh each of the k neighbors, (inversely) based on the distance from X.
- Hence the primary hyperparameters of the KNN model are the value of k, the distance measure, and the weight function.



K-Nearest Neighbors (KNN)

The value of k is arguably the most important hyperparameter of the model, and the optimal value of k usually depends on what's called the *bias-variance tradeoff*.

- Bias refers to the error from the incorrect assumptions a model makes about the data which causes it to miss important connections in the data.
- Variance is the error caused by fluctuations in the particular training set.

The tradeoff is because small values of k tend to have a high variance while large values of k tend to have a high bias.

K-Nearest Neighbors (KNN)

- For (an extreme) example, if $k = 1$, then the algorithm is using the class label of the single closest training data point. Hence, the model would be entirely dependent on that individual data point and highly sensitive to changes in the nearest neighbor.
- As k increases, the model averages over more neighbors, which "smoothens out" the predictions and reduces the model's sensitivity to individual points.
- However, a large value of k implies that the model may be too simplistic (think averaging over almost all the data points), and may not capture the idea of "averaging over a point's close neighbors".

K-Nearest Neighbors (KNN)

Tuning all of these hyperparameters (value of k, distance function, weight function) is done via the `tune::tune()` function.

```
# Create model specification.
knn_model_spec <-
  parsnip::nearest_neighbor(
    neighbors = tune::tune(),
    dist_power = tune::tune(),
    weight_func = tune::tune()
  ) |>
  parsnip::set_mode('classification') |>
  parsnip::set_engine('kknn')

# Create model workflow.
knn_workflow <- workflows::workflow() |>
  workflows::add_model(knn_model_spec) |>
  workflows::add_recipe(data_rec)
```

K-Nearest Neighbors (KNN)

Our final KNN model on the test set yields the following results.

Metric	Value
Accuracy	90.1%
Precision	76.9%
Recall	87%
Specificity	91.2%

KNN has a high specificity of 91.2% indicating it's ability to capture the true negatives.

Logistic Regression (LR)

- Supervised learning algorithm widely used for classification problems.
- For binary classification, the algorithm is an extension of the linear regression algorithm, but instead of estimating the output, the algorithm estimates the *log odds*.

Odds and Log Odds

The odds denote the probability of "success" to that of "failure". Let $p(X)$ denote the probability of some event X happening. Then its odds are the quantity $\frac{p(X)}{1-p(X)}$. The log odds denote the quantity $\log \frac{p(X)}{1-p(X)}$.

LR Model

For a given point X with predictors $X_1, \ldots, X_p$, LR aims to fit the model

$$\log \text{odds}_X = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p,$$

where the $\beta_0, \beta_1, \ldots, \beta_p$ are parameters to be estimated.

Logistic Regression (LR)

The model equation can be rearranged to model the probability instead of log odds.

Sigmoid function

The sigmoid function σ is defined as

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

LR Model as Probability

Let $p(X)$ denote the probability a given point X is classified as the positive class. Then LR estimates $p(X)$ via

$$p(X) = \sigma(\beta_0 + \beta_1 \cdot X_1 + \cdots + \beta_p \cdot X_p).$$

Logistic Regression (LR)

These parameters are estimated using the method of maximum likelihood estimation (MLE), which maximizes the likelihood of the observed data. The basic method is as follows.

LR Parameter Estimation

For a given training set of points $x_1, \ldots, x_n$, we are trying to maximize the likelihood function

$$L(\beta) = \prod_{y_i=1} P(x_i) \prod_{y_i=0} (1 - P(x_i)).$$

We can write this as a single product

$$L(\beta) = \prod_{i=1}^{n} P(x_1)^{y_i} \cdot (1 - P(x_i))^{(1-y_i)}.$$

This is equivalent to maximizing the log-likelihood function

$$\ell(\beta) = \sum_{i=1}^{n} [y_i \log P(x_i) + (1 - y_i) \log(1 - P(x_i))].$$

Logistic Regression (LR)

The primary advantage of LR is its interpretability. Each coefficient represents the change in the log-odds for a one unit change in the corresponding predictor variable, providing clear insight into the influence of each predictor on the final probability.

Logistic Regression (LR)

- In the implementation, the key hyperparameter we optimize is the penalty value, which refers to the coefficient of the regularization term added to the loss function to prevent overfitting.

```
# Create model specification.
lr_model_spec <-
  parsnip::logistic_reg(
    penalty = tune(),
    mixture = 1) |>
  parsnip::set_mode('classification') |>
  parsnip::set_engine('glmnet')

# Create model workflow.
lr_workflow <- workflows::workflow() |>
  workflows::add_model(lr_model_spec) |>
  workflows::add_recipe(data_rec)
```

Logistic Regression (LR)

Here are the final metrics for LR on the test set.

Metric	Value
Accuracy	94.5%
Precision	84.6%
Recall	95.7%
Specificity	94.1%

Logistic Regression has a high specificity and comparatively high precision, highlighting its ability to capture both the true negatives and be correct about its positive predictions.

Conclusion

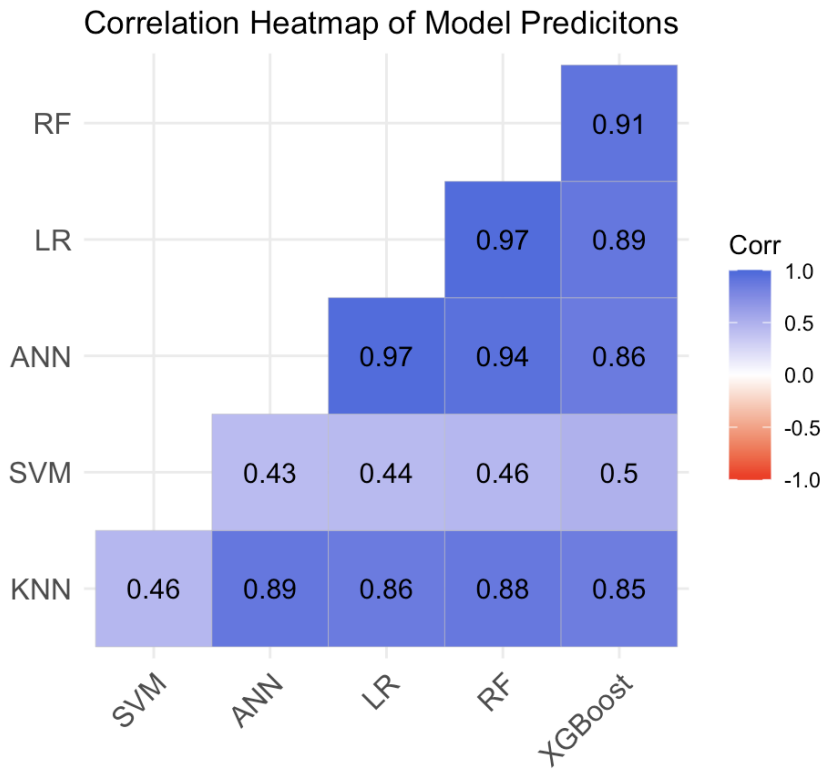
- RF is the most robust and balanced classifier for predicting DTC recurrence.
- RF achieved the highest accuracy and specificity rates, both at 94%, demonstrating its reliability in correctly identifying both positive and negative cases.
- ANN, LR, and RF all achieve 85% precision, so they are equally competent at minimizing false positives.
- SVM excels at recall, so it is the most effective at identifying all actual positive cases.

Table 8: Performance Comparison: Accuracy, Precision, Recall, and Specificity.

Metric	Model	Value
Accuracy	Random Forest	94%
Precision	ANN, Logistic Regression, Random Forest	85%
Recall	SVM	100%
Specificity	Random Forest	94%

Correlation Matrix of Model Predictions

To better understand how the models compare to one another, we can look at how similar the predictions are. Here is a correlation heatmap pairwise comparing the predictions.



Correlation Matrix of Model Predictions

- The correlation between SVM predictions and the rest of the models is much lower than the other correlations. The pairwise correlations between the non-SVM models are all at least 0.85, while those between SVM and the other models are all at most 0.5.
- This suggests that SVM predicts DTC in a slightly different manner than the rest.
- This is closely related to SVM having a recall of 100.0% but an extremely low precision of 23.1%.
- There is not a single test case in which SVM predicts negative, but the other model predicts positive.
- For each model, there are between 14 and 20 test cases (among the 91 total) for which SVM predicts positive but the other model predicts negative.

Future Directions

To improve model performance and pinpoint the critical predictors of DTC:

- Factor Analysis for Mixed Data (FAMD)
- Feature Importance Analysis (FIA)

To conduct a more in-depth comparison of model performance:

- Bayesian model comparison

References

- Pellegriti, G., Frasca, F., Regalbuto, C., Squatrito, S. and Vigneri, R. (2013). *Worldwide increasing incidence of thyroid cancer: Update on epidemiology and risk factors.* Journal of Cancer Epidemiology 2013 1–10.
- Bhattacharya, S., Mahato, R. K., Singh, S., Bhatti, G. K., Mastana, S. S. and Bhatti, J. S. (2023). *Advances and challenges in thyroid cancer: The interplay of genetic modulators, targeted therapies, and AI-driven approaches.* Life Sciences 332 122110.
- NM, X., L, W. and C., Y. (2022). *Improving the diagnosis of thyroid cancer by machine learning and clinical data.* Scientific report 12 11143.
- Abiodun, O. I., Jantan, A., Omolara, A. E., Dada, K. V., Mohamed, N. A. and Arshad, H. (2018). *State-of-the-art in artificial neural network applications: A survey.* Heliyon 4 e00938.
- Cervantes, J., Garcia-Lamont, F., Rodríguez-Mazahua, L. and Lopez, A. (2020). *A comprehensive survey on support vector machine classification: Applications, challenges and trends.* Neurocomputing 408 189–215.

References

- Syriopoulos, P. K., Kotsiantis, S. B. and Vrahatis, M. N. (2022). *Survey on KNN methods in data science.* In Learning and intelligent optimization (D. E. Simos, V. A. Rasskazova, F. Archetti, I. S. Kotsireas and P. M. Pardalos, ed) pp 379–93. Springer International Publishing, Cham.
- Maalouf, M. (2011). *Logistic regression in data analysis: An overview.* International Journal of Data Analysis Techniques and Strategies 3 281–99.
- Genuer, R., Poggi, J.-M. and Tuleau-Malot, C. (2010). *Variable selection using random forests.* Pattern Recognition Letters 31 2225–36.
- Anju and Sharma, N. (2018). *Survey of boosting algorithms for big data applications.* International Journal of Engineering Research and Technology (IJERT) 5.
- Kourou, K., Exarchos, T. P., Exarchos, K. P., Karamouzis, M. V. and Fotiadis, D. I. (2015). *Machine learning applications in cancer prognosis and prediction.* Computational and Structural Biotechnology Journal 13 8–17.
- G. James, T. H., D. Witten and Tibshirani, R. (2021). *An introduction to statistical learning.* Springer Texts in Statistics.

Thank you!