K – Nearest Neighbor, Random Forests, and Support Vector Machines

Handout

Submitted by:

Almer John B. Sta. Ines

Cloyd Eduard B. Sancio

Ma. Angela C. Gabayeron

Phia Ashley Roc

College of Science, Department of Mathematics and Statistics

Polytechnic University of the Philippines

STAT-MSE3 Introduction to Data Mining

Katrina Elizon

July 2023

Table of Contents

Data Description

The dataset comprises patient records gathered during a trial conducted by the German Breast Cancer Study Group (GBSG) between 1984 and 1989, involving 720 patients diagnosed with node-positive breast cancer. The dataset was refined to include 686 patients with complete data on prognostic variables. These datasets were utilized in Royston and Altman's paper from 2013.

Columns	Description
pid	Patient Identifier
age	Age (in years)
meno	Menopausal Status (0= premenopausal, 1= postmenopausal)
size	Tumor Size (in mm)
grade	Tumor Grade (levels: 3)
nodes	number of positive lymph nodes
pgr	progesterone receptors (fmol/l)
er	estrogen receptors (fmol/l)
hormon	hormonal therapy (0= no, 1= yes)
rfstime	recurrence free survival time; days to first of recurrence, death or last follow-up
status	0= alive without recurrence, 1= recurrence or death

Random Forest

The Random Forest algorithm is a powerful tree-based technique in supervised machine learning. It functions by creating multiple decision trees during the training phase. For instance, a random

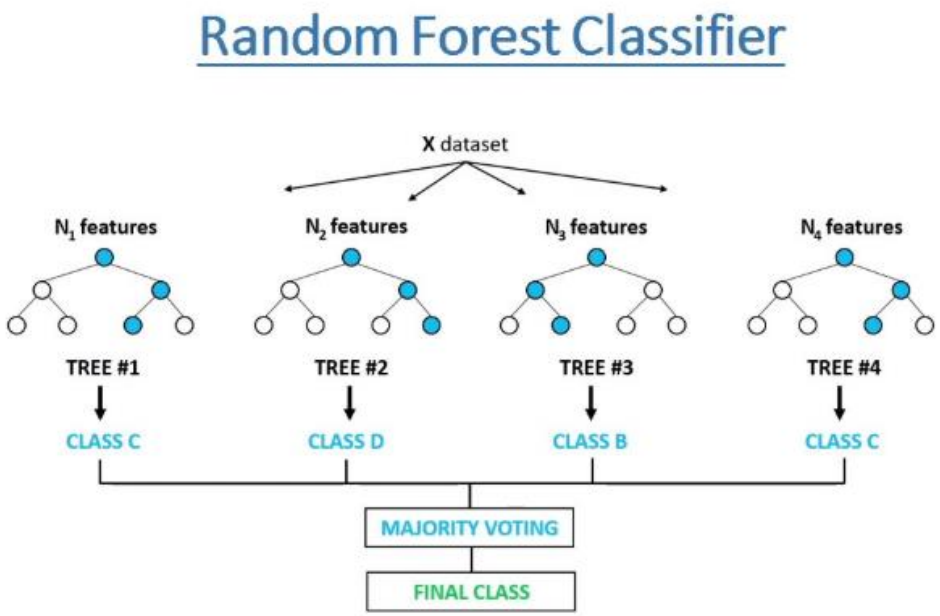
forest can predict factors that aid in the efficient operation of various industries, such as customer behavior, patient history, and safety measures. Data scientists apply random forests in numerous sectors, including banking, stock trading, healthcare, and e-commerce.

Definition

Random forests are frequently employed for classification and regression tasks because they excel at handling complex data, reducing overfitting, and providing accurate predictions in various contexts. The core concept of Random Forest involves constructing numerous decision trees through random sampling with replacement, with the final prediction being determined by the aggregate of these trees.

Fig. 1.1

Random Forest Classifier



Assumptions

1. There should be some actual values in the feature variable of the dataset, which will give the classifier a better chance to predict accurate results, rather than an estimation. Missing values should be handled from the training model.

Applications of Random Forests

Random Forest is used in various sectors such as banking, medicine, land use, spam email detection, sentiment analysis, customer churn prediction, forecasting behavior and outcome in banking and finance, e-commerce, and healthcare.

- In banking random forest is used to identify loan risk
- While in medicine, it is used to identify illness trends and risks
- In land-use, random forest is used to classify lands with similar use patterns.
- In market trends, random forest is used to determine market trends.

Versus Decision Trees

While both random forests and decision trees are both guided machine learning (ML) algorithms, the modeling structure is entirely different. Random Forest is a combination of numerous decision trees making it complex to interpret. Thus, random forest excels in application that requires high accuracy, robustness, and high-dimensionality of data sets such us its utilization on genomics, market basket analysis, bioinformatics, and anomaly detection.

Advantages of using Random Forests

- It is highly versatile, with easily understandable hyperparameters.
- Classifier does not overfit when enough trees are used.
- It efficiently manages large datasets.
- It achieves higher accuracy compared to decision trees.
- It is user-friendly and more accessible for beginners.
- It can perform both regression and classification tasks effectively.
- It offers superior accuracy compared to the decision tree algorithm.
- Random Forest provides an effective mechanism for managing missing data.
- It can generate reasonably accurate predictions even without hyperparameter tuning.
- Random Forest addresses the overfitting problem inherent in decision trees.
- In every random forest tree, a subset of features is selected randomly at the node's splitting point.

Versus Decision Trees

Random Forest are advantageous for resolving more complex problems with multiple features and higher accuracy requirements. Overfitting is less likely to occur due to the ensemble structure of random forests. However, in terms of sizes, decision trees are better with fewer features and simple interpretability.

Limitations of using Random Forests

- Achieving higher accuracy demands a larger number of trees.
- Excessive decision trees can significantly slow down the model.
- It lacks the ability to explicitly describe relationships within the data.
- Computationally, it demands more resources.
- Compared to a single decision tree algorithm, it typically requires more time for training and prediction.

Versus Decision Trees

In terms of limitations, random forest is less reliable with linear techniques as compared to decision trees. Due to high-dimensionality in nature, random forests is relatively slow and inefficient for real-time predictions due to the existence of large number of trees; comparably to decision trees, it offers faster prediction because it only accommodates the features with highest importance. Consequently, decision trees are susceptible for overfitting, thus, will require complex pruning procedures.

Coding Process

Step 1: Required Packages and Data Preparation

Patient identifier was excluded from the features since it is only an identifying mechanism to identify breast cancer patients from the dataset.

```r
# Data Preparation
# Required Packages
library(randomForest)
library(caret)
library(ggplot2)
library(dplyr)

# Data Cleaning
breastc_cleaned <- breastc %>%
  mutate(breastc,
         age = as.numeric(age),
         meno = as.factor(meno),
         size = as.numeric(size),
         grade = as.numeric(grade),
         nodes = as.numeric(nodes),
         pgr = as.numeric(pgr),
         er = as.numeric(er),
         hormon = as.numeric(hormon),
         rfstime = as.numeric(rfstime),
         status = as.factor(status)) %>%
  select( age, meno, size, grade, nodes, pgr,
          er, hormon, rfstime, status)
levels(breastc_cleaned$status)[levels(breastc_cleaned$status)=="0"] <- "alive"
levels(breastc_cleaned$status)[levels(breastc_cleaned$status)=="1"] <- "expired"
View(breastc_cleaned)
```

Step 2: Creating a training and testing sets.

The ratio of training to testing sets is 7:3. 70% of the data were used as a training sets and 30% were used as a testing set.

```r
# Training and Testing Data Sets
set.seed(123)
rftrainIndex <- createDataPartition(y = breastc_cleaned$status,
                                    p = 0.7, list  = FALSE)
rftrain <- breastc_cleaned[trainIndex, ]
rftest <- breastc_cleaned[-trainIndex, ]
```

Step 3: Fitting the Random Forest Model

```r
# Model Fit
forest <- randomForest(status~., data = rftrain)
forest
```

Step 4: Assessment for Optimal Parameters

Based on the model, 73.80% of the Out-of-the -Bag (OOB) samples were correctly classified by the model.

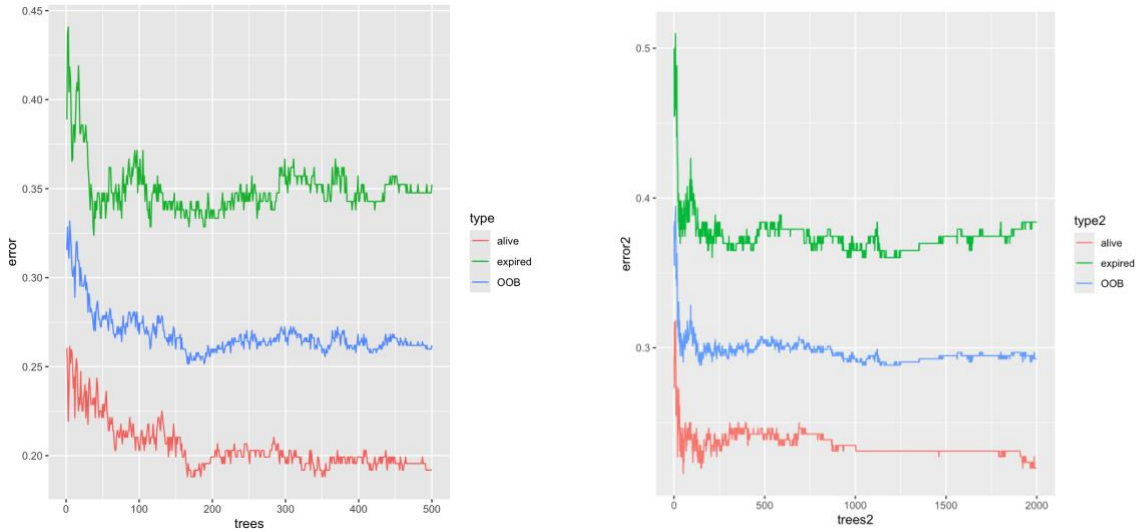
```
Call:
 randomForest(formula = status ~ ., data = rftrain)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 3

        OOB estimate of  error rate: 26.2%
Confusion matrix:
        alive expired class.error
alive     219      52   0.1918819
expired    74     136   0.3523810
>
```

Thus, plotting the OOB samples:

```
oob.error.data <- data.frame(
  trees = rep(1:nrow(forest$err.rate), times = 3),
  type = rep(c("OOB", "alive", "expired"), each = nrow(forest$err.rate)),
  error = c(forest$err.rate[,"OOB"],
            forest$err.rate[,"alive"],
            forest$err.rate[,"expired"]))

ggplot(data = oob.error.data, aes(x = Trees, y = Error)) +
  geom_line(aes(color = Type))

forest_2 <- randomForest(status~., data = train, ntree = 2000)

oob.error.data2 <- data.frame(
  trees2 = rep(1:nrow(forest_2$err.rate), times = 3),
  type2 = rep(c("OOB", "alive", "expired"), each = nrow(forest_2$err.rate)),
  error2 = c(forest_2$err.rate[,"OOB"],
            forest_2$err.rate[,"alive"],
            forest_2$err.rate[,"expired"]))
ggplot(data = oob.error.data2, aes(x = trees2, y = error2)) +
  geom_line(aes(color = type2))
```

Fig. 1.2.

Comparison of OOB plots by the Number of Trees



This reveals that error rate decreases when the random forest has more trees. Moreover, OOB rates stabilizes as the number of tree increases with ntree = 2000.

Moreover, to determine the optimal number of splits,

```
# Optimal splits
oob.optimal <- vector(length = 15)
for(i in 1:15) {
  temp.model <- randomForest(status~., data = rftrain,
                             mtry = i, ntree = 2000)
  oob.optimal[i] <- temp.model$err.rate[nrow(temp.model$err.rate),1]
}
oob.optimal
```

```
> oob.optimal
 [1] 0.3035343 0.2785863 0.2598753 0.2681913 0.2681913 0.2806653 0.2723493 0.2723493 0.2640333
[10] 0.2681913 0.2723493 0.2661123 0.2744283 0.2723493 0.2681913
```

This confirms that the default value of three (mtry = 3) as the number of splits is the most optimal value for number of variables tried in each split.

To determine the variable with the highest predictive power, variable importance will be employed. Higher values indicate more importance.

```
# Variable Importance
varImp(forest)
```

```
> varImp(forest)
          Overall
age     28.849164
meno     3.083608
size    24.947859
grade    9.744112
nodes   24.539471
pgr     35.438374
er      28.527871
hormon   5.028780
rfstime 75.611816
```

Step 5: Assumption Checks

Since the only assumption should be met is the presence of missing values, this can be detected with the following code.

```
sum(is.na(breastc))
```

```
> sum(is.na(breastc))
[1] 0
```

Since there are no missing values detected, no corrective measures should be employed.

Step 6: Making Predictions

```
#Prediction
predict_rf <- predict(forest_2, rftest)
```

Step 7: Model Diagnostics

```
# Model Diagnostics
rfcm <- confusionMatrix(predict_rf, rftest$status)
rfcm
rfcm$byClass
rfcm$overall
```

```
Confusion Matrix and Statistics

          Reference
Prediction alive expired
   alive      98      32
   expired    18      57

               Accuracy : 0.7561
                 95% CI : (0.6914, 0.8132)
    No Information Rate : 0.5659
    P-Value [Acc > NIR] : 0.00000001126

                  Kappa : 0.4943

 Mcnemar's Test P-Value : 0.06599

            Sensitivity : 0.8448
            Specificity : 0.6404
         Pos Pred Value : 0.7538
         Neg Pred Value : 0.7600
             Prevalence : 0.5659
         Detection Rate : 0.4780
   Detection Prevalence : 0.6341
      Balanced Accuracy : 0.7426

       'Positive' Class : alive

> rfcm$byClass
         Sensitivity          Specificity       Pos Pred Value       Neg Pred Value
           0.8448276            0.6404494            0.7538462            0.7600000
           Precision               Recall                   F1           Prevalence
           0.7538462            0.8448276            0.7967480            0.5658537
      Detection Rate Detection Prevalence    Balanced Accuracy
           0.4780488            0.6341463            0.7426385
> rfcm$overall
        Accuracy            Kappa    AccuracyLower    AccuracyUpper     AccuracyNull
0.75609756097561 0.49432659102121 0.69140521607175 0.81322457849008 0.56585365853659
  AccuracyPValue    McnemarPValue
0.00000001125554 0.06599205505935
```

The following summarizes the result of random forest.

Confusion Matrix

		Reference	
		Alive	***Expired***
Predictions	***Alive***	98	32
	Expired	18	57

	Accuracy	***Precision***	***Recall***	***F1 Score***
Random Forest	75.61%	64.04%	84.48%	79.67%

Accuracy measures the proportion of correct predictions made by the model. Precision, on the other hand, measures the proportion of true positive predictions among all positive predictions. Recall pertains to the sensitivity rate. F1 Score is defined to be the harmonic mean of precision and recall.

Support Vector Machines (SVM)

The Support Vector Machine is a powerful supervised algorithm that performs well on both simpler and more complex datasets. During the 1990s, these algorithms gained significant popularity and have remained a preferred choice for high-performing algorithms with few adjustments. You can apply SVMs to various tasks such as text classification, image classification, spam detection, handwriting identification, gene expression analysis, and face detection. SVM algorithms are highly efficient in identifying the largest separating hyperplane among the several classes present in the target feature.

Definition

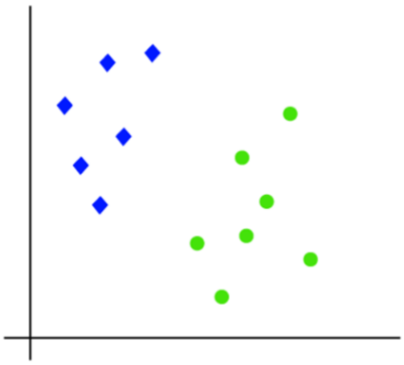
Support Vector Machines (SVM), which can be utilized in regression and classification applications, generally perform well, particularly in classification problems. The primary goal of

the SVM algorithm is to identify the most optimal hyperplane in an N-dimensional space that can accurately divide the data points into distinct classes within feature space. The hyperplane aims to maximize the distance between the nearest points of various classes, known as the margin. The number of features determines the size of the hyperplane. When there are only two input characteristics, the hyperplane is simply a line. Three input characteristics reduce the hyperplane to a two-dimensional plane. It becomes harder to imagine as the number of features surpasses three.

Let us examine an illustrative example to gain a comprehensive understanding of the functioning of support vector machines (SVM). Let's consider a dataset containing two distinct classifications, namely green and blue. We aim to categorize the new data point as either blue or green.

Fig. 2.1.

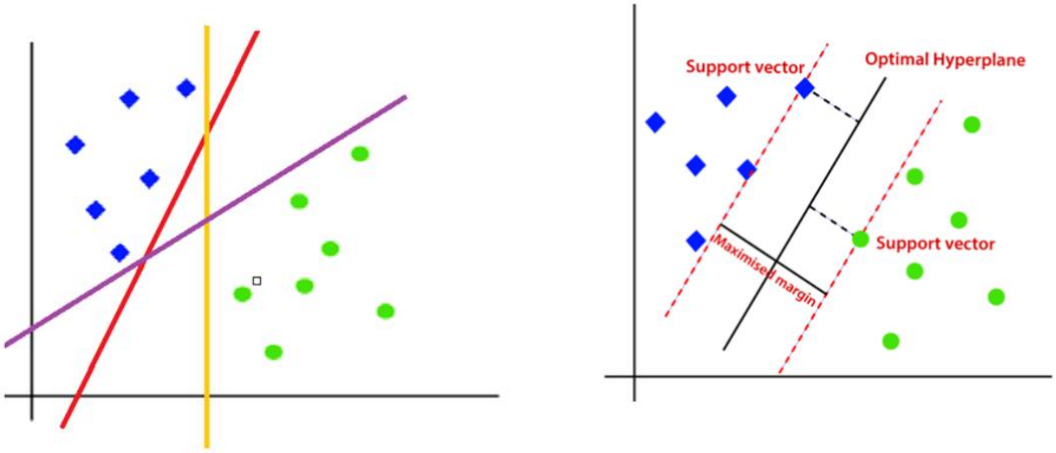
Illustrative Example of SVM: Data Points with two Distinct Classifications



To categorize these points, there are numerous possible decision boundaries. However, the most important question is: which one is the most ideal, and how can we determine it? It is important to note that when we plot the data points on a 2-dimensional graph, the decision boundary is referred to as a straight line. However, if we have additional dimensions, we refer to the decision boundary as a "hyperplane."

Fig. 2.2.

Illustrative Example of SVM: Optimal Hyperplane



The optimal hyperplane maximizes distance from both classes, SVM's main goal. This is achieved by identifying distinct hyperplanes that effectively categorize the labels and subsequently selecting the one that is farthest from the data points or possesses the maximum margin.

Assumptions of SVM

Support vector machines contain specific assumptions that are necessary for understanding when using them.

1. Linear Separability: The basic idea of SVM is that the data either already exists in a linearly separable space or can be converted into one. In short, there is a hyperplane that can clearly distinguish between the classes.
2. Margin Maximization: SVM seeks to identify the hyperplane that maximizes the distance between classes, known as the margin. This implies that an increased margin leads to better generalization and performance.

Application of SVM

Support vector machines are frequently utilized in several sectors because of their wide range and application. SVM are commonly used in several applications, such as:

1. Face detection – The SVM model recognizes the image's facial and non-facial components and generates a square boundary around the face.

2. Text and hypertext categorization – SVM allows for the categorization of text and hypertext using both inductive and transudative models. Training data enables the classification of documents into distinct categories. The system classifies data based on the generated score and subsequently compares it to the threshold value.
3. Classification of images – The utilization of SVM enhances the precision of picture classification in search tasks. It offers superior precision compared to conventional query-based search methods.
4. Bioinformatics – The content includes both protein classification and cancer classification. We use SVM for gene classification, patient classification based on genes, and other medical issues.
5. Protein fold and remote homology detection – Utilize SVM algorithms to find remote homology in proteins.
6. Handwriting recognition – SVMs are being extensively used for the recognition of handwritten characters.
7. Generalized predictive control (GPC) – Using relevant parameters, apply SVM-based GPC to regulate chaotic dynamics.

Versus Decision Trees

Like random forests, SVM excels in datasets with high-dimensionality and complexity; compared to decision trees which prioritizes simplicity. Thus, SVM offers higher accuracy and precision for text classifications, image classifications, and time-series forecasting. Studies o handwritten equation recognition, raisin seed classification, and rule extraction in power systems demonstrates superiority of SVM over decision trees.

Advantages of using SVM

1. Effective in High-Dimensional Spaces. SVMs exhibit strong performance in high-dimensional space, making them well-suited for tasks that include a significant number of features.

2. Robust to Overfitting: SVMs are less prone to overfitting, particularly in scenarios involving a large number of dimensions, because they include a margin that penalizes data points located within the margin.
3. Effective in Cases with Clear Margin of Separation: SVMs are effective in scenarios where there is a visible gap between different classes, making them appropriate for tasks with well-defined and well-distinguished classes.
4. Kernel Trick for Non-Linear Data: SVMs utilize the kernel method to effectively handle non-linear decision boundaries by implicitly transforming the data into higher-dimensional spaces.
5. Versatile Kernels: SVMs have the ability to use multiple kernel functions, allowing for the capture of various relationships within the data.
6. Memory Efficiency: SVMs employ a limited number of training points, known as support vectors, to make decisions. This characteristic makes SVMs cheap in terms of memory usage, particularly when handling large datasets.

Versus Decision Trees

Studies involving the comparison of SVM and Decision Trees shows that SVM outperforms decision trees in terms of accuracy and precision. Decision trees struggles with complex continuous dynamics; while, SVM provides a more comprehensive approach.

Limitations of using SVM

1. Sensitivity to Noise and Outliers: These factors can influence the location and direction of the decision boundary, making SVMs sensitive to noise and outliers.
2. Difficulty in Handling Large Datasets: SVMs can be computationally demanding and require a significant amount of memory, especially when dealing with huge datasets.
3. Kernel and Parameter Choice: Choosing a suitable kernel and fine-tuning hyperparameters can be difficult, and these decisions can have a significant impact on performance.
4. Limited Interpretability: The decision function of SVMs lacks simplicity, making it difficult to comprehend the individual impact of each feature on the final conclusion.

5. Not Suitable for Imbalanced Datasets: SVMs may exhibit poor performance when used with imbalanced datasets that have significant differences in class frequencies.
6. Binary Classification: SVMs are classifiers specifically designed to operate with two classes. We may use additional algorithms, such as one-vs.-one or one-vs.-all, to tackle multiclass situations.

Versus Decision Trees

In terms of limitation, decision trees are effective in handling linear relationships; while SVM expands to more complex relationships such as those with polynomial relationships. Decision trees are susceptible to overfitting which is an advantage of SVM due to the margin maximization principle to generalize unseen data. However, SVM are more complex in computation compared to decision tree.

Coding Process

Step 1: Required Packages and Data Preparation

Patient identifier was excluded from the features since it is only an identifying mechanism to identify breast cancer patients from the dataset.

```
# Required Package
library(e1071)
library(caret)
library(dplyr)

breastc_cleaned <- breastc %>%
  mutate(breastc,
         age = as.numeric(age),
         meno = as.factor(meno),
         size = as.numeric(size),
         grade = as.numeric(grade),
         nodes = as.numeric(nodes),
         pgr = as.numeric(pgr),
         er = as.numeric(er),
         hormon = as.numeric(hormon),
         rfstime = as.numeric(rfstime),
         status = as.factor(status)) %>%
  select( age, meno, size, grade, nodes, pgr,
          er, hormon, rfstime, status)
levels(breastc_clean$status)[levels(breastc_cleaned$status)=="0"] <- "alive"
levels(breastc_clean$status)[levels(breastc_cleaned$status)=="1"] <- "expired"
```

Step 2: Creating Training and Testing Sets

The ratio of training to testing sets is 7:3. 70% of the data were used as a training sets and 30% were used as a testing set.

```
# Training and Testing Data Sets
set.seed(123)
svmtrainIndex <- createDataPartition(y = breastc_cleaned$status,
                                     p = 0.7, list  = FALSE)
svmtrain <- breastc_cleaned[trainIndex, ]
svmtest <- breastc_cleaned[-trainIndex, ]
```

Step 3: Fitting the SVM Model

```
# Model fitting
svm_model <- svm(status~., data = svmtrain, type = "C-classification",
                 kernel = "radial")
```

Step 4: Making Predictions

```
# Prediction
prediction <- predict(svm_model, newdata = svmtest)
```

Step 5: Model Diagnostics

```
# Model Diagnostics
conf_matrix <- confusionMatrix(prediction, svmtest$status)
conf_matrix
conf_matrix$byClass
conf_matrix$overall
```

```
Confusion Matrix and Statistics

          Reference
Prediction alive expired
   alive      89      28
   expired    27      61

               Accuracy : 0.7317
                 95% CI : (0.6655, 0.791)
    No Information Rate : 0.5659
    P-Value [Acc > NIR] : 0.0000006622

                  Kappa : 0.4532

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.7672
            Specificity : 0.6854
         Pos Pred Value : 0.7607
         Neg Pred Value : 0.6932
             Prevalence : 0.5659
         Detection Rate : 0.4341
   Detection Prevalence : 0.5707
      Balanced Accuracy : 0.7263

       'Positive' Class : alive

> conf_matrix$byClass
         Sensitivity          Specificity       Pos Pred Value       Neg Pred Value
           0.7672414            0.6853933            0.7606838            0.6931818
           Precision               Recall                   F1           Prevalence
           0.7606838            0.7672414            0.7639485            0.5658537
      Detection Rate Detection Prevalence    Balanced Accuracy
           0.4341463            0.5707317            0.7263173
> conf_matrix$overall
       Accuracy          Kappa   AccuracyLower   AccuracyUpper    AccuracyNull
0.7317073170732 0.4532272925658 0.6655382217393 0.7910419546837 0.5658536585366
 AccuracyPValue   McnemarPValue
0.0000006622447 1.000000000000
```

The following summarizes the result of SVM.

Confusion Matrix

		Reference	
		Alive	***Expired***
Predictions	***Alive***	89	28
	Expired	27	61

	Accuracy	*Precision*	*Recall*	*F1 Score*
SVM	73.17%	68.54%	76.72%	76.39%

K – Nearest Neighbor

K-Nearest Neighbor (K-NN) is a simple, yet powerful, supervised machine learning algorithm used primarily for classification and regression tasks. The core idea behind K-NN is to predict the value of a new data point based on the 'k' most similar data points in the training set. It operates on the assumption that similar data points exist in close proximity. The algorithm does not make any assumptions about the underlying data distribution, making it a non-parametric and instance-based learning technique. This characteristic allows K-NN to be highly flexible and effective across a variety of datasets, especially those where the relationship between variables is complex and non-linear.

Definition

K-Nearest Neighbor (K-NN) is a type of instance-based learning, or lazy learning, where the function is only approximated locally, and all computation is deferred until function evaluation. It is a non-parametric algorithm, meaning it does not make any underlying assumptions about the distribution of the data.

In K-NN, the 'K' signifies the number of nearest neighbors that are considered when making a prediction for a new data point. For classification tasks, the algorithm assigns the class that is most common among the K nearest neighbors. For regression tasks, it takes the average of the values of the K nearest neighbors.

The process involves:

1. Choosing the number of K and a distance metric: Common distance metrics include Euclidean, Manhattan, and Minkowski distances.
2. Calculating the distance: For each data point in the training set, the distance from the new data point is computed.

3. Sorting and selecting K nearest neighbors: The K data points with the smallest distances are selected.
4. Predicting the output: For classification, the most frequent class among the K neighbors is chosen. For regression, the mean or median of the K neighbors' values is used.

Assumptions of K-NN

K-NN algorithm makes several assumptions about the data and its structure to function effectively:

1. Similarity Assumption

   K-NN assumes that similar data points are close to each other in the feature space. This means that data points that are near each other are likely to belong to the same class (for classification tasks) or have similar output values (for regression tasks).
2. Feature Relevance

   The algorithm assumes that all features contribute equally to the distance calculation. If certain features are more important than others, they may need to be weighted accordingly, or irrelevant features should be removed to avoid misleading distance computations.
3. Local Homogeneity

   K-NN assumes that the local region (defined by the K nearest neighbor) is homogenous and representative of the point being predicted. This means that the data is expected to be locally smooth, with no abrupt changes in the region around the query point.
4. Sufficient and Representative Training Data

   K-NN relies on having a sufficient amount of training data to accurately represent the feature space. The training data should be comprehensive enough to cover the various aspects of the data distribution and ensure that the nearest neighbors are meaningful.
5. Balance of Classes

   For classification tasks, K-NN assumes that the classes are balanced. If the classes are imbalanced, the majority class can dominate the prediction, leading to biased results. Techniques such as class weighting or resampling may be necessary to address class imbalance.
6. Independent and Identically Distributed (IID) Data

K-NN assumes that the training and test data are drawn from the same distribution and are independently and identically distributed. This ensures that the nearest neighbor found during training are relevant for predicting the test data.

Applications of K-NN

K-Nearest Neighbor (K-NN) can be applied to a variety of practical problems in both classification and regression contexts. Some common applications include:

Image Recognition

K-NN is widely used in image classification tasks. For instance, it can classify an image based on its similarity to other images in the training set. Each image is represented as a feature vector, and K-NN can determine the class of a new image by finding the K most similar images from the training set.

Handwriting Recognition

In optical character recognition (OCR), K-NN can classify handwritten characters by comparing them to a database of known characters. The algorithm matches the unknown character to its nearest neighbors in terms of pixel intensity and spatial distribution.

Recommendation Systems

K-NN can be used to build recommendation systems by finding users with similar preferences. For example, in a movie recommendation system, K-NN can suggest movies to a user based on the preferences of similar users (nearest neighbors) in terms of movie ratings.

Medical Diagnosis

K-NN can assist in diagnosing medical conditions by comparing a patient's medical data (e.g., symptoms, test results) to that of other patients. The algorithm identifies the most similar past cases and uses their outcomes to predict the diagnosis or treatment for the new patient.

Finance

In the finance sector, K-NN can be used for credit scoring, fraud detection, and stock price prediction. For credit scoring, it compares the financial data of a new applicant with that of previous applicants to determine the likelihood of default. For fraud detection, it identifies transactions that are significantly different from typical user behavior.

Anomaly Detection

K-NN is effective in identifying anomalies or outliers in a dataset. In network security, it can detect unusual patterns in network traffic that may indicate a potential security threat.

Text Classification

In natural language processing (NLP), K-NN can classify text documents based on their content. For example, it can categorize emails into spam or non-spam by finding the most similar emails in the training set.

Customer Segmentation

Businesses use K-NN to segment customers into different groups based on purchasing behavior, demographics, or other attributes. This helps in targeting marketing efforts more effectively.

Advantages of using K-NN

K-Nearest Neighbor (K-NN) offers several advantages that make it a popular choice for many machine learning tasks:

1. Simplicity. K-NN is easy to understand and implement. The algorithm does not require any complex parameter tuning or extensive pre-processing of data, making it accessible for beginners and straightforward to apply in practice.
2. Versatility. K-NN can be used for both classification and regression tasks. It is flexible and can handle various types of data, including numerical, categorical, and mixed data types.
3. No Training Phase. K-NN is a lazy learning algorithm, meaning there is no explicit training phase. The entire dataset is used at the time of prediction, which can be beneficial in

scenarios where the training data is updated frequently or where computational resources are limited.

4. Adaptability to Multiclass Problems. K-NN can naturally handle multiclass classification problems without the need for complex extensions. The algorithm can classify data points into multiple categories by simply considering the majority vote among the nearest neighbors.
5. Non-parametric Nature. K-NN does not assume any underlying distribution of the data. This non-parametric characteristic allows it to be effective in scenarios where the relationship between features and the target variable is complex and non-linear.
6. Performance with Large Datasets. With a sufficient amount of representative data, K-NN can perform well on large datasets. The algorithm can leverage the richness of the dataset to make accurate predictions by finding the most similar instances.
7. Robustness to Noise. K-NN can be robust to noisy data if an appropriate value of K is chosen. A higher value of K can smooth out the effects of noise, leading to more stable and reliable predictions.
8. Scalability with Distance Metrics. K-NN can be adapted to different types of distance metrics (e.g., Euclidean, Manhattan, Minkowski) depending on the nature of the data and the problem at hand. This adaptability allows for customization to achieve better performance.
9. No Need for Model Training and Maintenance. Since K-NN does not involve a training phase, there is no need for model training and retraining. This characteristic is useful in dynamic environments where new data continuously becomes available.

Limitations of using K-NN

Versus Decision Trees

K-NN is intuitive and easy to implement, but it can be computationally intensive for large datasets since it requires calculating the distance from the query point to all points in the training set. Additionally, K-NN is sensitive to the local structure of the data and can be significantly affected by the choice of K and the distance metric used. Moreover, K-NN has several limitations:

1. K-NN does not learn the model during the training phase; instead, it stores ll the training data which leads to a faster training but slower classication modeling.
2. K-NN is sensitive to feature scaling. Features with larger ranges will dominate the distance metric more. Thus, data normalization shall be employed.
3. K-NN is also sensitive to outlier since that can disproportionately affect the nearest neighbors' determination.
4. Lacks generalizability and Interpretability.

Coding Process

Step 1: Required Packages and Data Preparation

Patient identifier was excluded from the features since it is only an identifying mechanism to identify breast cancer patients from the dataset.

```
# Required Packages and Data Preparation
library(class)
library(dplyr)

# Data cleaning
breastc_cleaned <- breastc %>%
  mutate(breastc,
         age = as.numeric(age),
         meno = as.factor(meno),
         size = as.numeric(size),
         grade = as.numeric(grade),
         nodes = as.numeric(nodes),
         pgr = as.numeric(pgr),
         er = as.numeric(er),
         hormon = as.numeric(hormon),
         rfstime = as.numeric(rfstime),
         status = as.factor(status)) %>%
  select( age, meno, size, grade, nodes, pgr,
          er, hormon, rfstime, status)
levels(breastc_clean$status)[levels(breastc_cleaned$status)=="0"] <- "alive"
levels(breastc_clean$status)[levels(breastc_cleaned$status)=="1"] <- "expired"
```

Step 2: Creating Training and Testing Sets

The ratio of training to testing sets is 7:3. 70% of the data were used as a training sets and 30% were used as a testing set.

```
# Testing and Training Sets
set.seed(123)
knntrainIndex <- createDataPartition(y = breastc_cleaned$status,
                                     p = 0.7, list  = FALSE)
knntrain <- breastc_cleaned[trainIndex, ]
knntest <- breastc_cleaned[-trainIndex, ]
View(knntrain)
```

Step 3: Optimal K – Values

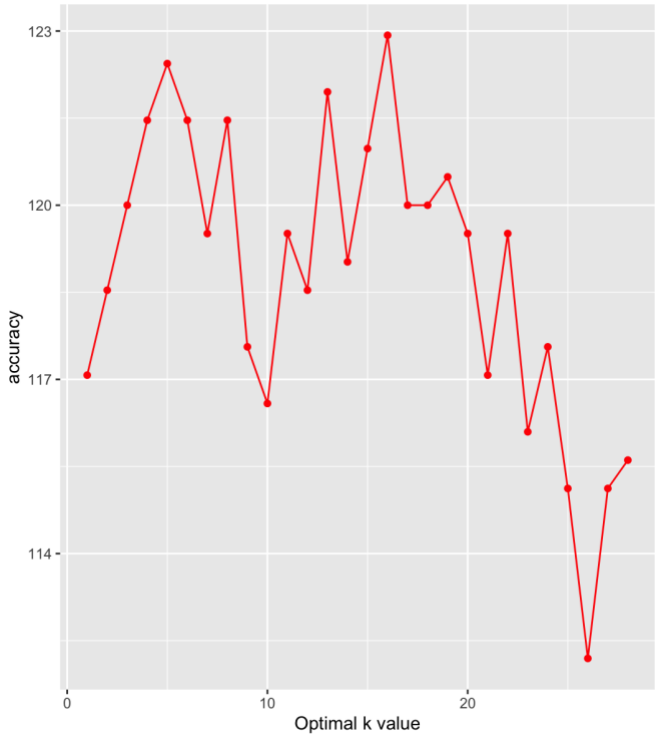
The optimal initial k-value can be determined by

```
# Optimal K Values
set.seed(123)
i = 1
k.optm = 1
for (i in 1:28){
  knn.mod <- knn(train = n.knntrain[, -10],
                 test = n.knntest[, -10],
                 cl = n.knntrain.labels,
                 k = i)
  k.optm[i] <- 100 * sum(n.knntrain.labels == knn.mod) / NROW(n.knntest.labels)
  k = i
  cat(k, '=', k.optm[i], '\n')
}
k_optimal<- data.frame(1:28, k.optm)
k_optimal
ggplot(data = k_optimal, aes(x = 1:28, y = k.optm)) +
  geom_line(color = "red") + geom_point(color = "red") +
  labs(x = "Optimal k value", y = "accuracy")
```

The code creates an iteration of variable of possible k values 28 times from the sequence of 1 to 28. Thus, the k-value with the highest accuracy level should be the best parameter to be used in the model.

```
> k_optimal
   X1.28   k.optm
1      1 117.0732
2      2 118.5366
3      3 120.0000
4      4 121.4634
5      5 122.4390
6      6 121.4634
7      7 119.5122
8      8 121.4634
9      9 117.5610
10    10 116.5854
11    11 119.5122
12    12 118.5366
13    13 121.9512
14    14 119.0244
15    15 120.9756
16    16 122.9268
17    17 120.0000
18    18 120.0000
19    19 120.4878
20    20 119.5122
21    21 117.0732
22    22 119.5122
23    23 116.0976
24    24 117.5610
25    25 115.1220
26    26 112.1951
27    27 115.1220
28    28 115.6098
```

Fig. 3.1. Optimal K-Parameter for the Model



The graph shows that a *k* of 16 will produce the highest accuracy result. Thus, it will be employed in the model.

Step 4: Making Predictions

```
# Model Prediction
knn_model <- knn(train = n.knntrain[, -10],
                 test = n.knntest[, -10],
                 cl = n.knntrain.labels,
                 k = 16)
```

Step 5: Model Diagnostics

```
# Model Prediction
knn_model <- knn(train = n.knntrain[, -10],
                 test = n.knntest[, -10],
                 cl = n.knntrain.labels,
                 k = 16)

knncm <- confusionMatrix(knn_model, n.knntest.labels)
knncm
knncm$byClass
knncm$overall
```

```
          Reference
Prediction  0  1
         0 82 40
         1 34 49

               Accuracy : 0.639
                 95% CI : (0.5692, 0.7048)
    No Information Rate : 0.5659
    P-Value [Acc > NIR] : 0.01989

                  Kappa : 0.2595

 Mcnemar's Test P-Value : 0.56108

            Sensitivity : 0.7069
            Specificity : 0.5506
         Pos Pred Value : 0.6721
         Neg Pred Value : 0.5904
             Prevalence : 0.5659
         Detection Rate : 0.4000
   Detection Prevalence : 0.5951
      Balanced Accuracy : 0.6287

       'Positive' Class : 0

> knncm$byClass
        Sensitivity          Specificity       Pos Pred Value       Neg Pred Value
          0.7068966            0.5505618            0.6721311            0.5903614
          Precision               Recall                   F1           Prevalence
          0.6721311            0.7068966            0.6890756            0.5658537
     Detection Rate Detection Prevalence    Balanced Accuracy
          0.4000000            0.5951220            0.6287292
> knncm$overall
      Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull AccuracyPValue
    0.63902439     0.25949429     0.56920036     0.70476297     0.56585366     0.01988998
 McnemarPValue
    0.56107993
```

The following summarizes the result of K-NN.

Confusion Matrix

		Reference	
		Alive	***Expired***
Predictions	***Alive***	89	28
	Expired	27	61

	Accuracy	***Precision***	***Recall***	***F1 Score***
K-NN	63.90%	55.06%%	70.69%	68.91%

Comparison to a Decision Tree Model

A decision tree has a hierarchical tree structure, which consists of a root node, internal nodes, and leaf nodes. The paths from root to leaf represent classification rules. In decision analysis, a decision tree and the closely related influence diagram are used as a visual and analytical decision support tool, where the expected values (or expected utility) of competing alternatives are calculated.

In machine learning, a decision tree is a non-parametric supervised learning algorithm used for both classification and regression tasks. Regression algorithms are used to predict the continuous values such as price, salary, age, etc., while Classification algorithms are used to predict or classify discrete values, such as sex, true or false, spam or not spam, etc.

Modeling of Decision Tree

Step 1: Required Packages and Data Preparation

```
# Decision Tree
library(rpart)
library(caret)
```

Step 2: Data Cleaning and Preparation

```
# Data Cleaning and Preparation
breastc_cleaned <- breastc %>%
  mutate(breastc,
         age = as.numeric(age),
         meno = as.factor(meno),
         size = as.numeric(size),
         grade = as.numeric(grade),
         nodes = as.numeric(nodes),
         pgr = as.numeric(pgr),
         er = as.numeric(er),
         hormon = as.numeric(hormon),
         rfstime = as.numeric(rfstime),
         status = as.factor(status)) %>%
  select( age, meno, size, grade, nodes, pgr,
          er, hormon, rfstime, status)
levels(breastc_cleaned$status)[levels(breastc_cleaned$status)=="0"] <- "alive"
levels(breastc_cleaned$status)[levels(breastc_cleaned$status)=="1"] <- "expired"
```

Step 3: Creating Training and Testing Sets

```
# Testing and Training dataset
set.seed(123)
dttrainIndex <- createDataPartition(y = breastc_cleaned$status,
                                    p = 0.7, list  = FALSE)
dttrain <- breastc_cleaned[trainIndex, ]
dttest <- breastc_cleaned[-trainIndex, ]
View(dttrain)
```

Step 4: Fitting the Model

```
# Model Fitting
dtmodel <- rpart(status~., data = dttrain,
                 method = "class",
                 control = rpart.control(minsplit = 5,
                                         cp = 0.001,
                                         maxdepth = 4))
```

Step 5: Making Predictions

```
# Making Predictions
dtpredict <- predict(dtmodel, dttest, type = "class")
```

Step 6: Model Diagnostics

```
# Model Diagnostics
dtcm <- confusionMatrix(dtpredict, dttest$status)
dtcm
dtcm$byClass
dtcm$overall
```

```
Confusion Matrix and Statistics

          Reference
Prediction alive expired
   alive      94      28
   expired    22      61

               Accuracy : 0.7561
                 95% CI : (0.6914, 0.8132)
    No Information Rate : 0.5659
    P-Value [Acc > NIR] : 0.00000001126

                  Kappa : 0.4997

 Mcnemar's Test P-Value : 0.4795

            Sensitivity : 0.8103
            Specificity : 0.6854
         Pos Pred Value : 0.7705
         Neg Pred Value : 0.7349
             Prevalence : 0.5659
         Detection Rate : 0.4585
   Detection Prevalence : 0.5951
      Balanced Accuracy : 0.7479

       'Positive' Class : alive

> dtcm$byClass
        Sensitivity          Specificity       Pos Pred Value       Neg Pred Value
          0.8103448            0.6853933            0.7704918            0.7349398
          Precision               Recall                   F1           Prevalence
          0.7704918            0.8103448            0.7899160            0.5658537
      Detection Rate Detection Prevalence    Balanced Accuracy
          0.4585366            0.5951220            0.7478690
> dtcm$overall
       Accuracy           Kappa    AccuracyLower    AccuracyUpper     AccuracyNull   AccuracyPValue
0.75609756097561 0.49965830323148 0.69140521607175 0.81322457849008 0.56585365853659 0.00000001125554
   McnemarPValue
0.47950012218695
```

Comparison by Model

Table 4.1

Model Diagnostics by Model

	Accuracy	*Precision*	*Recall*	*F1 Score*
Decision Tree	75.61%	68.54%%	81.03%	78.99%
Random Forest	75.61%	64.04%	84.48%	79.67%
SVM	73.17%	68.54%	76.72%	76.39%
K-NN	63.90%	55.06%%	70.69%	68.91%

Conclusions can be drawn from the table above that the K-NN performed the worst out of all the data mining techniques. It has the lowest accuracy, precision, recall and F1 score. In terms of accuracy, both decision trees and random forest have the highest rate with 75.61% correct prediction over the total instances. In terms of precision, both SVM and decision tree have the highest rate with 68.54% correct prediction over the total predicted positive instances. In terms of Recall, random forest gave the highest prediction rate with 84.58% to all the instance in the actual class. Lastly, in terms of F1 score, random forest gave the highest harmonic average of precision and recall with 79.67%. Thus, for this dataset, the use of Random Forest is recommended as the most optimal model for all.

References

B, H. N. (2019, December 10). Confusion Matrix, Accuracy, Precision, Recall, F1 Score. Retrieved June 25, 2024, from Medium website: https://medium.com/analytics-vidhya/confusion-matrix-accuracy-precision-recall-f1-score-ade299cf63cd

Chaudhary, M. (2022, June 28). Random Forest Algorithm - How It Works & Why It's So Effective. Retrieved June 26, 2024, from Turing.com website: https://www.turing.com/kb/random-forest-algorithm#applications-cases-of-random-forest-algorithm

GeeksforGeeks. (2024, February 22). Random Forest Algorithm in Machine Learning. Retrieved June 24, 2024, from GeeksforGeeks website: https://www.geeksforgeeks.org/random-forest-algorithm-in-machine-learning/

Meltzer, R. (2021, July 15). What is Random Forest? [Beginner's Guide + Examples]. Careerfoundry.com. https://careerfoundry.com/en/blog/data-analytics/what-is-random-forest/

Bhagyashrilakhadive. (2023, November 23). Introduction to support vector machine(SVM) - Bhagyashrilakhadive - Medium. *Medium*. Retrieved June 26, 2024, from https://medium.com/@bhagyashrilakhadive/introduction-to-support-vector-machine-svm-4946083b8052#:~:text=Linear%20Separability%3A%20The%20primary%20assumption,maximizes%20the%20margin%20between%20classes

GeeksforGeeks. (2023, June 10). *Support Vector Machine (SVM) Algorithm*. GeeksforGeeks. Retrieved June 26, 2024, from https://www.geeksforgeeks.org/support-vector-machine-algorithm/

Saini, A. (2024, May 22). *Guide on Support Vector Machine (SVM) Algorithm*. Analytics Vidhya. Retrieved June 26, 2024, from https://www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/

Real-Life Applications of SVM (Support Vector Machines). (n.d.). Data Flair. Retrieved June 26, 2024, from https://data-flair.training/blogs/applications-of-svm/

Rouse, M. (2017, March 14). What is K-Nearest Neighbor (K-NN)? Techopedia. Retrieved June 28, 2024, from: Stack Overflow

Joby, A. G2. (2023). K – Nearest Neighbor (KNN). From: K Nearest Neighbor or KNN Algorithm and It's Essence in ML (g2.com)

Gopal, N., & Lee, S. (2024, February 1). *Classifying data using the SVM algorithm using R on watsonx.ai*. IBM Developer. Retrieved July 1, 2024, from https://developer.ibm.com/tutorials/awb-classifying-data-svm-algorithm-r/

Lucy Martin (2023). When to Use Decision Trees vs. Random Forests in Machine Learning. Dzone. From. https://dzone.com/articles/when-to-use-decision-trees-vs-random-forest-in-mac#:~:text=While%20Random%20Forests%20are%20better,ensemble%20structure%20of%20Random%20Forests.

P. Jyothsna & R. Dhanalakshimi(2023). Analysis of Handwritten Equation Recognition System for Comparing Decision Tree and support Vector Machine Algorithm. doi: 10.1109/iconstem56934.2023.10142869

Florian, Jüngermann., Jan, Křetínský., Maximilian, Weininger. (2022). Algebraically Explainable Controllers: Decision Trees and Support Vector Machines Join Forces. arXiv.org, doi: 10.48550/arXiv.2208.12804

Meredith, L., Wallace., Lucas, Mentch., B., Wheeler., Amanda, L., Tapia., Siyu, Zhou., Susan, Redline., Daniel, J., Buysse. (2023). Use and misuse of random forest variable importance metrics in medicine: demonstrations through incident stroke prediction. BMC Medical Research Methodology, doi: 10.1186/s12874-023-01965-x

Hachnam, A., (2023). The KNN Algorithm – Explanation, Opportunities, Limitations. Retrieved from https://neptune.ai/blog/knn-algorithm-explanation-opportunities-limitations

Chaterjee, M., (2024). A Quick Introduction to KNN Algorithm. Retrieved from: https://www.mygreatlearning.com/blog/knn-algorithm-introduction/