Data Mining- Practice 5

Rajesh Kalakoti[§], Sven Nomm[*]
[*] Taltech, Estonia, 12616
[§] Email: rajesh.kalakoti@outlook.com

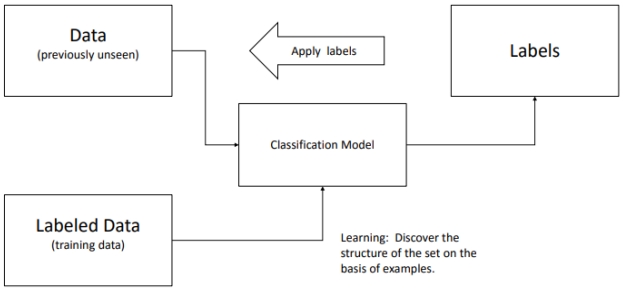


Fig. 1: Classfication

***Abstract*—As part of our ongoing exploration into this dynamic field, todays's lecture delves into the intricacies of classification techniques. Focusing on the versatile R programming language, this session will unravel the implementation of fundamental algorithms such as Naive Bayes, Decision Trees, Fisher Score, Gini Index, Entropy.**

I. Introduction

Classification is a fundamental task in machine learning and data mining where the goal is to assign a label or category to an input based on its features. In other words, classification algorithms learn from labeled training data, allowing them to make predictions or decisions about new, unseen data. It is widely used in various fields such as email filtering, speech recognition, image recognition, and sentiment analysis. you can some articles from medium [1]–[3].

II. Feature selection

In a dataset with **n** instances (data points) and **m** features (variables), represented as a matrix **X** of size **n** × **m**, where X_{ij} represents the value of feature **j** for instance **i**, and **y** is the corresponding vector of size **n** representing the target variable or class labels.

The objective of **feature selection** is to find a subset $S \subseteq \{1, 2, \ldots, m\}$ of features that maximizes (or minimizes) an objective function $J(S)$ representing the performance of the model. Feature selection aims to optimize the model's performance by identifying a subset of relevant features, thereby enhancing the model's accuracy, interpretability, and computational efficiency.

- Filter Methods: A subset of features is evaluated with the use of a class-sensitive discriminative criterion.
- Wrapper Methods: Wrapper models evaluate subsets of features using a specific machine learning algorithm.
- Embedded Methods: Embedded models integrate feature selection into the model training process.

A. Filter Methods

1) Gini Index: It measures Measures the discriminative power of a particular feature. it is used for categorical variables, but it can be generalized to numeric attributes by the process of discretization. Let $v_1, ..., v_r$ are the possible values of the particular categorical. Let p_j denotes the fraction of the data points containing attribute value v_i belonging to the class $j \in 1, ..., k$ to the data points containing attribute value v_i then Gini index defined as follows:

$$G(v_i) = 1 - \sum_{j=1}^{k} p_j^2 \tag{1}$$

A value of $1 - \frac{1}{k}$ indicates that the different classes are distributed evenly for a particular attribute value. Lower value of Gini Index imply Greater discrimination.

```r
#' gini index
#'
#' @param probabilities
#' Gini index, a measure of impurity
#' or inequality,
#' for a set of probabilities.
#'\deqn{G(v_i) = 1 - \sum_{j=1}^{k} p_j^2}.
#'
#' @return
#' @export
#'
#' @examples
gini_score <- function(probabilities) {
  gini_index <- 1 - sum(probabilities^2)
  return(gini_index)
}

# Example probabilities
probabilities <- c(0.2, 0.3, 0.5)

gini_value = gini_score(probabilities)
```

```
# Print the result
print(gini_value)
```

```
## [1] 0.62
```

2) Entropy: The class-based entropy measure is related to notions of information gain resulting from fixing a specific attribute value. The class- base entropy is defied as follows:

$$E(v_i) = -\sum_{j=1}^{k} p_i log_2(p_j) \tag{2}$$

takes its values in $[0, log_2(k)]$, whereas greater values indicate greater mixing.

By analogy with Gini index one may define overall Entropy as

$$E = \sum_{i=1}^{r} \frac{n_i E(v_i)}{n}$$

low entropy shall always be preferred over high entropy

```
entropy <- function(probs) {
  # Make sure the probabilities sum up to 1
  if (abs(sum(probs) - 1) > 1e-10) {
    stop("Probabilities must sum up to 1.")
  }

  probs <- probs[probs > 0]
  entropy_value <- -sum(probs*log2(probs))

  return(entropy_value)
}

probabilities1 <- c(0.5, 0.5)
probabilities2 <- c(0.2, 0.8)

# Calculate entropy
entropy_value1 <- entropy(probabilities1)
entropy_value2 <- entropy(probabilities2)

# Print the results
print(paste("Entropy:", entropy_value1))
```

```
## [1] "Entropy: 1"
```

3) Fisher Score.: The Fisher score is naturally designed for numeric attributes to measure the ratio of the average interclass separation to the average intraclass separation. The larger the Fisher score, the greater the discriminatory power of the attribute. Let μ_j and σ_j denote the mean and the standard deviation of the of the data points belonging to the class j, for a particular feature. And let p_j be the fraction of the points belonging to the class j. Finally let μ define the mean of the entire data set. The Fisher index is defined as follows:

$$F_s = \frac{\sum_{j=1}^{K} p_j(\mu_j^i - \mu^i)^2}{\sum_{j=1}^{K} p_j(\sigma_j^i)^2} \tag{3}$$

The attributes with the largest value of the Fisher score may be selected for use with the classification algorithm.

```
fisher_score <- function(data, labels) {
  cat(rep('==', 2), '\n')

  data_length <- nrow(data)
  list_of_classes <- unique(labels)
  number_of_classes <- length(list_of_classes)
  cat(paste("Data contains:", number_of_classes,

  fishers_score_frame <- data.frame(matrix(NA, nr
  colnames(fishers_score_frame) <- colnames(data)

  for (column in colnames(data)) {
    column_mean <- mean(data[[column]])
    numerator <- 0
    denominator <- 0

    for (label in list_of_classes) {
      indexes <- (labels == label)
      class_in_data <- data[indexes, column]
      class_mean <- mean(class_in_data)
      class_std <- sd(class_in_data)
      class_proportion <- sum(indexes) / data_ler
      numerator <- numerator + class_proportion *
      denominator <- denominator + class_proporti
    }

    if (denominator != 0) {
      fishers_score_frame[1, column] <- numerator
    } else {
      fishers_score_frame[1, column] <- 0
    }
  }

  cat("Fisher's score(s) has/have been computed.\
  fdf <- fishers_score_frame[1, !is.na(fishers_sc
  fisher_score_df <- as.data.frame(t(fdf))
  cat(rep('==', 2), '\n')
  return(fisher_score_df)
}
```

let's compute the fisher score over the dataset.

```
print(xtable(
  iris[sample(nrow(iris), 6), ],
  caption='Example of the iris dataset',
  label='tbl:iris.xtable',
  align=c(rep('r', 5), 'l')))
```

TABLE I: Example of the iris dataset

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.00	3.40	1.50	0.20	setosa
6.80	3.20	5.90	2.30	virginica
5.40	3.40	1.70	0.20	setosa
5.00	3.40	1.60	0.40	setosa
6.60	2.90	4.60	1.30	versicolor
5.50	2.30	4.00	1.30	versicolor

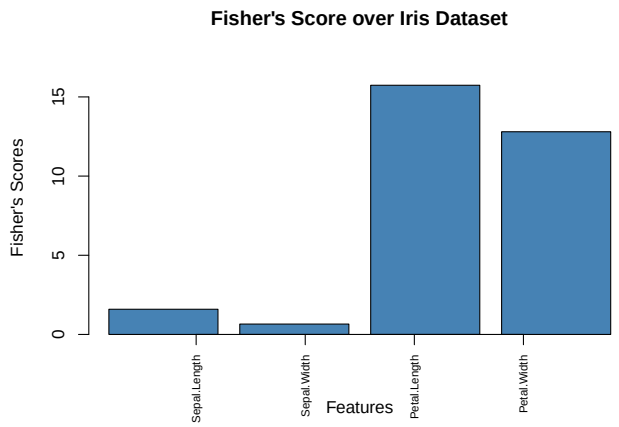
```
data(iris)
scores <- fisher_score(iris[, 1:4],
                       iris$Species)
```

```
## == ==
## Data contains: 3 classes.
## Fisher's score(s) has/have been computed.
## == ==
```

```
feature_names <- rownames(scores)
scores <- as.numeric(scores$'1')


# show fisher score
#in bar graph.
barplot(
scores, names.arg = feature_names,
main = "Fisher's Score over Iris Dataset",
xlab = "Features", ylab = "Fisher's Scores",
col = "steelblue", border = "black",
ylim = c(0, max(scores)+0.1*max(scores)),
axisnames = FALSE,
xaxt = 'n')

axis(1, at = 1:length(feature_names),
labels = feature_names,las = 2,
cex.axis = 0.7)
```



you can see, from the above bar Graph, Top features are Petal length, and petal width.

III. Classification.

A Decision Tree algorithm is one of the most popular machine learning algorithms. It uses a tree like structure and their possible combinations to solve a particular problem. It belongs to the class of supervised learning algorithms where it can be used for both classification and regression purposes.

A decision tree is a structure that includes a root node, branches, and leaf nodes. Each internal node denotes a test on an attribute, each branch denotes the outcome of a test, and each leaf node holds a class label. The topmost node in the tree is the root node.

A. Decision Tree:

```
library(here)
```

```
## here() starts at /home/rajeshkalakoti/Document
```

```
source(here("RMarkDown/week5",
            "decision_tree.R"))
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package
##
##     filter, lag

## The following objects are masked from 'package
##
##     intersect, setdiff, setequal, union
```

```
data(iris)
iris_data <- iris[iris$Species != 'setosa',]
input_data <- iris_data

train_index <- sample(row.names(input_data),
                      nrow(input_data) * 0.66)
test_index <- row.names(input_data)[!row.names(
  input_data) %in% train_index]

input_train <- input_data[train_index,]
input_test <- input_data[test_index,]
```

1) Splitting Data:

```
tree_rules <- fit.decision.tree(
  input_train,min_observations = 3)
tree_rules
input_train$regions <- apply(input_train,1,
```

```
                                identify_region,
                                tree_rules)
err_rate_vals <- error_rate(input_train,
                            "Species",
                            "regions")
print("Training Error Rate")
print(err_rate_vals[[1]])
class_prob <- err_rate_vals[[2]]
```

2) Fitting Decision Tree:

```
test_with_preds <- predict_test(input_test,
                                tree_rules,
                                class_prob)

preds_table <- test_with_preds[c("Species",
                                 "predict_class")]
```

3) Prediction over the Test data:

4) Classification Evaluation.: In the context of binary classification (where there are two classes, typically denoted as positive and negative), true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN) are used to evaluate the performance of a classification model. These values help in understanding how well the model is performing in terms of correctly and incorrectly predicting the classes.

Here's how you calculate these values based on model predictions and actual outcomes:

True Positives (TP): These are the cases where the model correctly predicts the positive class.

False Positives (FP): These are the cases where the model incorrectly predicts the positive class when it should have been negative.

True Negatives (TN): These are the cases where the model correctly predicts the negative class.

False Negatives (FN): These are the cases where the model incorrectly predicts the negative class when it should have been positive.

Let's assume you have a set of predictions and the corresponding actual outcomes:

Predicted: [1, 0, 1, 1, 0, 1] Actual: [1, 1, 1, 0, 0, 1]

To calculate TP, FP, TN, and FN based on these predictions:

- **True Positives (TP):** Count the number of cases where both predicted and actual values are 1. In this case, there are 3 instances (indices 1, 3, and 5).
- **False Positives (FP):** Count the number of cases where the predicted value is 1, but the actual value is 0. In this case, there is 1 instance (index 0).
- **True Negatives (TN):** Count the number of cases where both predicted and actual values are 0. In this case, there are 2 instances (indices 4 and 5).
- **False Negatives (FN):** Count the number of cases where the predicted value is 0, but the actual value is 1. In this case, there is 1 instance (index 2).

So, based on these calculations:

- **TP = 3**
- **FP = 1**
- **TN = 2**
- **FN = 1**

The metrics mentioned below (Precision, Recall, True Negative Rate (Specificity), and Accuracy) are all calculated based on the values of True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN). These metrics provide different perspectives on the performance of a classification model and are essential in evaluating the model's effectiveness using these fundamental elements of classification results.

- **Precision:** Precision is calculated as:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- **Recall:** Recall is calculated as:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- **True Negative Rate (Specificity):** True Negative Rate, also known as Specificity, is calculated as:

$$\text{TNR} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

- **Accuracy:** Accuracy is calculated as:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

- **Predicted Positive Condition Rate:** Predicted Positive Condition Rate is calculated as:

$$\text{Predicted Positive Condition Rate} = \frac{\text{TP} + \text{FP}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

```
library(here)
source(here("RMarkDown/week5",
            "confusion_matrix_.R"))
results = ConfusionMatrix(preds_table$Species,
                  preds_table$predict_class)

# Calculate Accuracy
accuracy <- results$Accuracy
print(paste("Accuracy:", accuracy))
```

```
## [1] "Accuracy: 0.824"
```

```r
# Calculate Precision
precision <- results$Precision
print(paste("Precision:", precision))
```

```
## [1] "Precision: 0.833"
```

```r
# Calculate F1-Score
f1_score <- results$F1_Score
print(paste("F1-Score:", f1_score))
```

```
## [1] "F1-Score: 0.833"
```

```r
# Calculate Sensitivity
#(True Positive Rate or Recall)
sensitivity <- results$Sensitivity
print(paste("Sensitivity(Recall):",
            sensitivity))
```

```
## [1] "Sensitivity(Recall): 0.833"
```

```r
# Calculate AUC (Area Under the Curve)
auc <- results$AUC
print(paste("AUC (Area Under the Curve):"
            , auc))
```

```
## [1] "AUC (Area Under the Curve): 0.823"
```

B. k-nearest neighbour (k-NN) classification

Let N be a labeled set of points belonging to c different classes such that

$$\sum_{i=1}^{c} N_i = N \tag{4}$$

During the classification of a given point x, the algorithm identifies the k nearest points to x and assigns x the majority label among its k nearest neighbors. K-NN relies on the concept of proximity, and for this, it employs a distance function to calculate the distances between points.

```r
library(here)
source(here("RMarkDown/week5","knn_classification.R"))
```

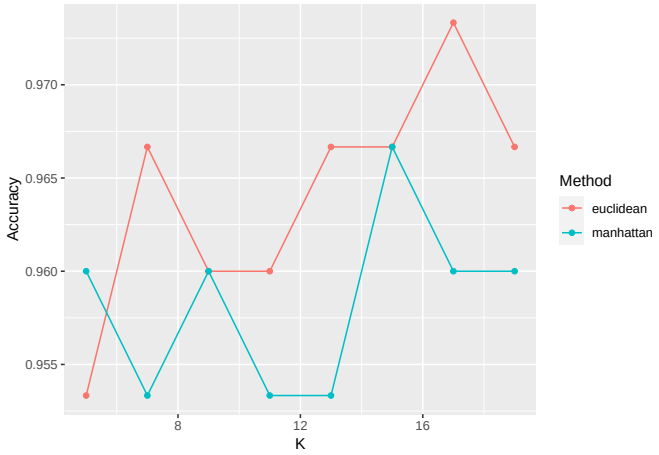
```
## -- Attaching core tidyverse packages ----
## v forcats   1.0.0     v readr     2.1.4
## v ggplot2   3.4.3     v stringr   1.5.0
## v lubridate 1.9.2     v tibble    3.2.1
## v purrr     1.0.1     v tidyr     1.3.0
## -- Conflicts ------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://con
##
## Attaching package: 'reshape'
##
```

```
##
## The following object is masked from 'package:l
##
##     stamp
##
##
## The following objects are masked from 'package
##
##     expand, smiths
##
##
## The following object is masked from 'package:d
##
##     rename
```

```r
data(iris)

labels <- iris$Species
data <- iris[, -5]
suppressMessages(eval_knn(
  data, labels,
  k_neighbors=seq(5, 19, by=2),
  metrics = c("euclidean", "manhattan")))
```

```
## [1] "Going for method euclidean"
## [1] "Going for method manhattan"
```




```
##       Method  K  Accuracy
## 1  euclidean  5 0.9733333
## 2  euclidean  7 0.9666667
## 3  euclidean  9 0.9600000
## 4  euclidean 11 0.9600000
## 5  euclidean 13 0.9666667
## 6  euclidean 15 0.9666667
## 7  euclidean 17 0.9733333
## 8  euclidean 19 0.9666667
## 9  manhattan  5 0.9600000
## 10 manhattan  7 0.9533333
## 11 manhattan  9 0.9600000
## 12 manhattan 11 0.9533333
```

```
## 13 manhattan 13 0.9533333
## 14 manhattan 15 0.9666667
## 15 manhattan 17 0.9600000
## 16 manhattan 19 0.9600000
```

C. Naive Bayes

It is based on Bayes' theorem with an assumption of independence between features. The "naive" assumption here implies that the presence of a particular feature in a class is independent of the presence of other features. This assumption simplifies the calculation process.

In Naive Bayes classification, the probability of a class C_k given the features x is calculated using Bayes' theorem as follows:

$$P(C_k|x) = \frac{P(x|C_k) \times P(C_k)}{P(x)}$$

Where: - $P(C_k|x)$ is the posterior probability of class C_k given features x. - $P(x|C_k)$ is the likelihood, representing the probability of observing the features x given class C_k. - $P(C_k)$ is the prior probability of class C_k. - $P(x)$ is the probability of observing the features x.

This equation is fundamental to the Naive Bayes classification algorithm.

$$y^* = \text{argmax}_{y \in \{0,1\}} \, p(y|x, \theta)$$

here i have used the package "e1071", you can install it.

```
library(e1071)
# Set a seed for reproducibility
set.seed(123)


split_index <- sample(1:nrow(iris),
                      0.7 * nrow(iris))
train_data <- iris[split_index, ]
test_data <- iris[-split_index, ]

naive_bayes_model <- naiveBayes(
Species ~ Sepal.Length + Sepal.Width,
data = train_data)
# Make predictions on the test data
predictions <- predict(naive_bayes_model,
newdata = test_data)
actual_labels = test_data$Species
```

1) Classification Evaluation: Naive Bayes Classification Results.

TABLE II: confusion matrix results

Accuracy	Precision	Sensitivity	F1_Score	Specificity	AUC_average
0.98	1.00	0.93	0.97	1.00	0.90
0.80	0.67	0.80	0.73	0.80	0.90
0.82	0.77	0.67	0.71	0.90	0.90

```
library(here)
source(here("RMarkDown/week5",
            "confusion_matrix_.R"))
results = ConfusionMatrix(
as.character(predictions),
actual_labels)
```

```
print(xtable(
   results,
   caption='confusion matrix results',
   label='tbl:results.xtable',
   align=c(rep('r', 6), 'l')))
```

References

[1] "Classification in machine learning — by amit upadhyay — analytics vidhya — medium," https://medium.com/analytics-vidhya/classification-in-machine-learning-ed30753d9461, (Accessed on 10/05/2023).

[2] "Intro to types of classification algorithms in machine learning — by mandy sidana — sifium — medium," https://medium.com/sifium/machine-learning-types-of-classification-9497bd4f2e14, (Accessed on 10/05/2023).

[3] "Lecture_05_dm2023_classification_i.pdf," https://courses.cs.taltech.ee/w/images/2/21/Lecture_05_DM2023_Classification_I.pdf, (Accessed on 10/05/2023).