STATISTICAL MODELLING FOR CLASSIFICATION OF MUSHROOM USING MACHINE LEARNING TECHNIQUES

Mohammed Kadhim
Department of statistics, Osmania University, Hyderabad, India, 500007.
mohammedalnasry2@gmail.com

Abstract

This study explores the efficacy of various machine learning algorithms for classifying mushrooms as either edible or poisonous, leveraging a comprehensive dataset from the UCI Machine Learning Repository. Employing techniques such as k-Nearest Neighbors, Support Vector Machines, Decision Trees, Random Forests, and Generalized Linear Models, this research aims to identify the most reliable method for mushroom classification. Through meticulous data preparation, feature selection, model training, evaluation, and validation, the study highlights the superior accuracy of the Random Forest algorithm. Further, it emphasizes the identification of key features that significantly influence mushroom classification, thereby enhancing our understanding of the crucial attributes distinguishing edible from poisonous mushrooms.

Keywords: *Mushroom, Classification, Random Forest, Machine Learning, Repository*

1. Introduction

The classification of mushrooms into edible or poisonous categories is a pivotal issue with far-reaching implications for public health, food safety, and biodiversity conservation. The annual incidence of mushroom poisoning underscores the critical need for accurate identification methods to prevent the consumption of toxic species. While traditional classification methods rely on expert knowledge, which can be limited by scalability and accessibility, the advent of *machine learning* (ML) technologies offers a promising alternative. These technologies employ computational models to accurately classify mushrooms based on a range of features, from morphological to genetic data.

Significant contributions to the development of ML algorithms for classification include the introduction of the *k-Nearest Neighbors* (k-NN) algorithm by Cover, T., & Hart, P. (1967), a foundational method for classification based on proximity to the nearest data points. Following this, the concept of optimal margin classifiers, crucial for the development of Support Vector Machines (SVM), was elaborated by Boser, B.E., Guyon, I.M., & Vapnik, V.N. (1992). Cortes, C., & Vapnik, V. (1995) further advanced SVM, laying the groundwork for its application in complex classification problems. Ho, T.K.

(1998) introduced the random subspace method for constructing decision forests, contributing to the development of robust ensemble methods. Breiman, L. (2001) proposed the *Random Forests* (RF) algorithm, an ensemble learning method that enhances prediction accuracy by combining multiple decision trees. Rish, I. (2001) conducted an empirical study of the Naive Bayes classifier, emphasizing the need for comparative studies in performance across algorithms. Kotsiantis, S.B. (2007) reviewed classification techniques within supervised machine learning, offering insights that inform algorithm selection. This was followed by the comprehensive guide on statistical learning by Hastie, T., Tibshirani, R., & Friedman, J. (2009), covering a wide array of ML algorithms.

In more recent years, Friedman, J., Hastie, T., & Tibshirani, R. (2010) explored regularization paths for *Generalized Linear Models* (GLM) via coordinate descent, refining predictive models., and James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013) introduced foundational concepts in statistical learning, focusing on methods like regression, classification, and clustering.

Through the application and evaluation of a suite of ML algorithms on a dataset from the UCI Machine Learning Repository, this paper aims to compare the performance of k-NN, SVM, DT, RF, and GLM in mushroom classification. Our goal is to identify the most effective strategies for this crucial task, contributing valuable insights to the fields of botany, public health, and machine learning.

The problem concerns whether the mushroom’s nature is fit for human consumption. There are 22 diverse characteristics defined for each mushroom in the dataset. The interest here is to find different combinations of aspects of their features and ecospheres that mostly influence their kind. To assess the class/type of a mushroom entrenched on their varied environmental surroundings and physical appearance. The overarching objectives of our study are threefold and revolve around the intricate relationship between a mushroom's characteristics and its classification as either edible or poisonous. Firstly, we aim to pinpoint specific attributes that significantly influence a mushroom's type, thereby shedding light on the critical factors that differentiate edible from poisonous varieties. This involves a meticulous analysis of various physiognomies such as color, shape, habitat, and more to understand which features are most predictive of a mushroom's classification.

Building upon this foundational knowledge, our second goal is to develop an innovative algorithm capable of classifying mushrooms based on their distinct physiognomies. This algorithm is expected to leverage the identified attributes, employing statistical or machine learning techniques to accurately categorize mushrooms into their respective groups.

Lastly, our ambition extends to the realm of prediction. Armed with a set of characteristics, we intend to forecast the classification group of new mushroom specimens, effectively determining their edibility or toxicity. This predictive capability is crucial for not only advancing scientific understanding but also for practical applications in fields such as culinary arts, foraging, and toxicology, offering a robust tool for ensuring food safety and public health. Through these efforts, our research seeks to bridge the gap between empirical observation and predictive accuracy, contributing significantly to the field of mycology and beyond.

2. Machine Learning Models Used

The methodology of this work centers on employing a range of ML algorithms to classify mushrooms as either edible or poisonous, leveraging a dataset compiled from the UCI Machine Learning Repository.

1. **k-Nearest Neighbors (k-NN)**
   k-NN was chosen for its effectiveness in handling categorical data. The model was tuned by experimenting with different values of k and distance metrics (Cover, T., & Hart, P., 1967).

2. **Support Vector Machines (SVM)**
   SVM was utilized for its robustness in high-dimensional spaces, applying kernel tricks to achieve better separation. Parameters such as the choice of kernel and regularization constant were optimized (Cortes, C., & Vapnik, V., 1995).

3. **Decision Trees (DT)**
   DTs were employed for their interpretability, with optimization focusing on the depth of the tree and the minimum number of samples required to split a node (Quinlan, J.R., 1986).

4. **Random Forests (RF)**
   RFs, an ensemble of decision trees, were chosen for their superior performance, especially in handling dataset variability. Hyperparameters like the number of trees and the maximum features were tuned (Breiman, L., 2001).

5. **Generalized Linear Models (GLM)**
   GLMs, specifically logistic regression, were used to estimate the probability of a mushroom being edible, with regularization techniques applied to improve model generalizability (Nelder, J.A., & Wedderburn, R.W.M., 1972).

6. **Model Training and Evaluation**
   Models were trained using the prepared and feature-selected dataset, with their performance evaluated using accuracy, precision, recall, and the F1 score. Confusion matrices and k-fold cross-validation were also employed for a comprehensive assessment (Kohavi, R., 1995; Powers, D.M.W., 2011).

3. Results and Discussion

In this section, we are going to discuss about the results of different ML algorithms in order to obtain the solution for the problem mentioned in section 2.

1. Approach to the Problem

The dataset consists of 8124 observations, each representing a single mushroom. The first column is the target variable containing the class labels, identifying whether the mushroom is poisonous or edible. The remaining columns are 22 discrete features that describe the mushroom in some observable way.

In order to carry out the analysis, we have extracted a random sample of 500 records from the 8124 and the information of the same is mentioned in section 2.

2. ML Process

2.1 Data Preparation

The dataset has been acquired from a secondary source. It was downloaded from UCI repository. (Source: https://archive.ics.uci.edu/ml/datasets/mushroom). The original dataset consists of 8124 observations taken over 22 features and the target variable being the type of mushroom. All the variables considered are categorical type.

The foremost and an essential phase of any ML Algorithm is cleaning and preparing data for modelling. The process of data cleaning starts with reading the dataset into R. Next, we look at basic description of the data: datatypes, observation, attributes, missing values, and summary. That is, at this point we seek for any inconsistencies in the data and attempt to get a grip on what the data actually is. Simply, we try to understand the data.

```
> # Reading the data file into R
> # File name: Mushroom
> M1=read.csv("Mushroom.csv")
> # Looking at basic framework of the dataset - observations, attributes, datatypes
> dim(M1)
[1] 500  23
> class(M1)
[1] "data.frame"
> names(M1)
 [1] "y"   "x1"  "x2"  "x3"  "x4"  "x5"  "x6"  "x7"  "x8"  "x9"  "x10" "x11" "x12" "x13"
 "x14"
[16] "x15" "x16" "x17" "x18" "x19" "x20" "x21" "x22"
```

Output 1: Reading Mushroom data file into R and looking at dimensions, class, columns

.

```
> summary(M1)
 y       x1      x2      x3      x4      x5      x6      x7      x8      x9      x10     x11     x12     x13     x14     x15
 0:360   1:116   1:106   3 :  1   1:104   1:173   1:  7   1:404   1:365   3 : 64   1:392   1: 50   1: 27   1: 30   4:  7   4:  2
 1:140   3:108   3:188   4 : 49   2:396   4:160   2:493   2: 96   2:135   4 : 11   2:108   2:220   3:473   3:395   7:  3   7:  5
         5: 14   4:206   5 :116           6:116                           5 :101           3:155           4: 75   8:490   8:493
         6:262           9 :168           7: 51                           6 :140           4: 75
                         10:166                                           8 : 78
                                                                          10:  2
                                                                          11:104

 x16     x17     x18     x19     x20     x21     x22
 1:500   2:  6   2:495   1: 58   3:223   1: 31   1: 50
         3:494   3:  5   5:442   4:256   3:100   2:238
                         7: 21   4:209   4:103
                                 5: 87   5: 39
                                 6: 73   6: 70
```

Output 2: Counts of each level on every variable in the data set. (Categorical variables)

```
> str(M1)
'data.frame':   500 obs. of  23 variables:
 $ y  : int  1 0 0 0 1 0 0 0 0 1 ...
 $ x1 : int  6 1 6 1 6 1 6 6 1 6 ...
 $ x2 : int  3 3 3 3 4 3 4 4 3 4 ...
 $ x3 : int  5 9 4 9 9 10 10 10 10 9 ...
 $ x4 : int  2 2 1 2 2 2 2 2 2 2 ...
 $ x5 : int  7 4 6 1 7 1 4 1 1 7 ...
 $ x6 : int  2 2 1 1 1 1 1 1 1 2 ...
 $ x7 : int  1 1 2 1 1 1 1 1 1 1 ...
 $ x8 : int  2 1 1 1 2 1 1 1 1 2 ...
 $ x9 : int  5 6 5 3 8 3 3 6 11 5 ...
 $ x10: int  1 1 2 1 1 1 1 1 1 1 ...
 $ x11: int  3 2 3 2 3 2 2 2 2 3 ...
 $ x12: int  3 3 3 3 3 3 3 3 3 3 ...
 $ x13: int  3 3 3 3 3 3 3 3 3 3 ...
 $ x14: int  8 8 8 8 8 8 8 8 8 8 ...
 $ x15: int  8 8 8 8 8 8 8 8 8 8 ...
 $ x16: int  1 1 1 1 1 1 1 1 1 1 ...
 $ x17: int  3 3 3 3 3 3 3 3 2 2 ...
 $ x18: int  2 2 2 2 2 2 2 2 2 2 ...
 $ x19: int  5 5 1 5 5 5 5 5 5 5 ...
 $ x20: int  3 4 4 3 3 3 4 3 4 4 ...
 $ x21: int  4 3 1 3 5 4 3 4 4 5 ...
 $ x22: int  6 4 2 4 2 4 2 4 2 6 ...
```

Output 3: Structure of Mushroom data – all categorical variables displayed as integers

Here, we observe that all the variables are in integer type, and they do not agree with the problem. R has, by default, understood the different levels of variables as numbers. As such, before we can carry out any analyses, these variables must be converted to factors.

```
> str(M1)
'data.frame':   500 obs. of  23 variables:
 $ y  : Factor w/ 2 levels "0","1": 2 1 1 1 2 1 1 1 1 2 ...
 $ x1 : Factor w/ 4 levels "1","3","5","6": 4 1 4 1 4 1 4 4 1 4 ...
 $ x2 : Factor w/ 3 levels "1","3","4": 2 2 2 2 3 2 3 3 2 3 ...
 $ x3 : Factor w/ 5 levels "3","4","5","9",..: 3 4 2 4 4 5 5 5 5 4 ...
 $ x4 : Factor w/ 2 levels "1","2": 2 2 1 2 2 2 2 2 2 2 ...
 $ x5 : Factor w/ 4 levels "1","4","6","7": 4 2 3 1 4 1 2 1 1 4 ...
 $ x6 : Factor w/ 2 levels "1","2": 2 2 1 1 1 1 1 1 1 2 ...
 $ x7 : Factor w/ 2 levels "1","2": 1 1 2 1 1 1 1 1 1 1 ...
 $ x8 : Factor w/ 2 levels "1","2": 2 1 1 1 2 1 1 1 1 2 ...
 $ x9 : Factor w/ 7 levels "3","4","5","6",..: 3 4 3 1 5 1 1 4 7 3 ...
 $ x10: Factor w/ 2 levels "1","2": 1 1 2 1 1 1 1 1 1 1 ...
 $ x11: Factor w/ 4 levels "1","2","3","4": 3 2 3 2 3 2 2 2 2 3 ...
 $ x12: Factor w/ 2 levels "1","3": 2 2 2 2 2 2 2 2 2 2 ...
 $ x13: Factor w/ 3 levels "1","3","4": 2 2 2 2 2 2 2 2 2 2 ...
 $ x14: Factor w/ 3 levels "4","7","8": 3 3 3 3 3 3 3 3 3 3 ...
 $ x15: Factor w/ 3 levels "4","7","8": 3 3 3 3 3 3 3 3 3 3 ...
 $ x16: Factor w/ 1 level "1": 1 1 1 1 1 1 1 1 1 1 ...
 $ x17: Factor w/ 2 levels "2","3": 2 2 2 2 2 2 2 2 1 1 ...
 $ x18: Factor w/ 2 levels "2","3": 1 1 1 1 1 1 1 1 1 1 ...
 $ x19: Factor w/ 2 levels "1","5": 2 2 1 2 2 2 2 2 2 2 ...
 $ x20: Factor w/ 3 levels "3","4","7": 1 2 2 1 1 1 2 1 2 2 ...
 $ x21: Factor w/ 5 levels "1","3","4","5",..: 3 2 1 2 4 3 2 3 3 4 ...
 $ x22: Factor w/ 5 levels "1","2","4","5",..: 5 3 2 3 2 3 2 3 2 5 ...
```

Output 4: Structure of Mushroom data – converting into factor variables

As we can see, variables are now changed to factors and suit the data captured.

Moreover, from above structure of data, we can notice that there is only one level for variable x16. This variable can be removed since it is not of much use in defining the type of mushroom. Also, it may cause troubles at later stages while running the models like: it can trigger error:

`contrasts<-`(`*tmp*`, value = contr.funs[1 + isOF[nn]]) : contrasts can be applied only to factors with 2 or more levels

Removing the x16 variable

```
> # Removing x16 variable as it has only one level and irrelevant for data analysis
> M2=M1[,c(1:16,18:23)]
> names(M2)
 [1] "y"   "x1"  "x2"  "x3"  "x4"  "x5"  "x6"  "x7"  "x8"  "x9"  "x10" "x11" "x12" "x13" "x14" "x15" "x17" "x18" "x19" "x20" "x21"
[22] "x22"
> summary(M2)
 y       x1      x2       x3      x4      x5      x6      x7      x8       x9      x10     x11     x12     x13     x14     x15
 0:360   1:116   1:106   3 :  1   1:104   1:173   1:  7   1:404   1:365   3 : 64   1:392   1: 50   1: 27   1: 30   4:  7   4:  2
 1:140   3:108   3:188   4 : 49   2:396   4:160   2:493   2: 96   2:135   4 : 11   2:108   2:220   3:473   3:395   7:  3   7:  5
         5: 14   4:206   5 :116           6:116                           5 :101           3:155           4: 75   8:490   8:493
         6:262           9 :168           7: 51                           6 :140           4: 75
                         10:166                                           8 : 78
                                                                          10:  2
                                                                          11:104

 x17     x18     x19     x20     x21     x22
 2:  6   2:495   1: 58   3:223   1: 31   1: 50
 3:494   3:  5   5:442   4:256   3:100   2:238
                         7: 21   4:209   4:103
                                 5: 87   5: 39
                                 6: 73   6: 70
```

Output 5: Deleting x16 variable and again looking at the counts

Lastly, we search for missing values.

```
> #Checking for missing value
> print(all(!is.na(M2)))
[1] TRUE
```

Output 6: Finding Missing Values

There are no missing values and need no attention. The irregularities in the data are now taken care of.

2.2. Understanding Relationships Between Data

At the second phase, we try to discover certain relationships: among the variables, between variables and the target. Furthermore, we make an effort to represent these relations using plots.

Contingency Tables: Contingence tables are valuable for revealing how edible/poisonous mushrooms are segmented across their features.

For all variables x1 through x22, the following holds:

prop.table(m,1) – value of each cell divided by the sum of the rows cells

prop.table(m,2) – value of each cell divided by the sum of the column cells where 'm' represents an object in R, here it is a table.

```
> prop.table(table(M2$x1,M2$y),2)

           0         1
1 0.25000000 0.18571429
3 0.22500000 0.19285714
5 0.02500000 0.03571429
6 0.50000000 0.58571429
> prop.table(table(M2$x1,M2$y),1)

           0         1
1 0.7758621 0.2241379
3 0.7500000 0.2500000
5 0.6428571 0.3571429
6 0.6870229 0.3129771
```

Output 7: Proportionate table – x1 v/s y

```
> prop.table(table(M2$x2,M2$y),2)

           0         1
1 0.2416667 0.1357143
3 0.3750000 0.3785714
4 0.3833333 0.4857143
> prop.table(table(M2$x2,M2$y),1)

           0         1
1 0.8207547 0.1792453
3 0.7180851 0.2819149
4 0.6699029 0.3300971
```

Output 8: Proportionate table – x2 v/s y

```
> prop.table(table(M2$x3,M2$y),2)

            0           1
3  0.002777778 0.000000000
4  0.108333333 0.071428571
5  0.188888889 0.342857143
9  0.338888889 0.328571429
10 0.361111111 0.257142857
> prop.table(table(M2$x3,M2$y),1)

           0         1
3  1.0000000 0.0000000
4  0.7959184 0.2040816
5  0.5862069 0.4137931
9  0.7261905 0.2738095
10 0.7831325 0.2168675
```

Output 9: Proportionate table – x3 v/s y

```
> prop.table(table(M2$x4,M2$y),2)

           0         1
1 0.2388889 0.1285714
2 0.7611111 0.8714286
> prop.table(table(M2$x4,M2$y),1)

           0         1
1 0.8269231 0.1730769
2 0.6919192 0.3080808
```

Output 10: Proportionate table – x4 v/s y

```
> prop.table(table(M2$x5,M2$y),2)

           0         1
1 0.3888889 0.2357143
4 0.3444444 0.2571429
6 0.2666667 0.1428571
7 0.0000000 0.3642857
> prop.table(table(M2$x5,M2$y),1)

           0         1
1 0.8092486 0.1907514
4 0.7750000 0.2250000
6 0.8275862 0.1724138
7 0.0000000 1.0000000
```

Output 11: Proportionate table – x5 v/s y

```
> prop.table(table(M2$x6,M2$y),2)

            0           1
1 0.016666667 0.007142857
2 0.983333333 0.992857143
> prop.table(table(M2$x6,M2$y),1)

           0         1
1 0.8571429 0.1428571
2 0.7180527 0.2819473
```

Output 12: Proportionate table – x6 v/s y

```
> prop.table(table(M2$x7,M2$y),2)

            0         1
  1 0.7722222 0.9000000
  2 0.2277778 0.1000000
> prop.table(table(M2$x7,M2$y),1)

            0         1
  1 0.6881188 0.3118812
  2 0.8541667 0.1458333
```

Output 13: Proportionate table – x7v/s y

```
> prop.table(table(M2$x8,M2$y),2)

            0         1
  1 0.8111111 0.5214286
  2 0.1888889 0.4785714
> prop.table(table(M2$x8,M2$y),1)

            0         1
  1 0.8000000 0.2000000
  2 0.5037037 0.4962963
```

Output 14: Proportionate table – x8 v/s y

```
> prop.table(table(M2$x9,M2$y),2)

              0           1
  3  0.136111111 0.107142857
  4  0.027777778 0.007142857
  5  0.188888889 0.235714286
  6  0.294444444 0.242857143
  8  0.152777778 0.164285714
  10 0.005555556 0.000000000
  11 0.194444444 0.242857143
> prop.table(table(M2$x9,M2$y),1)

              0           1
  3  0.76562500 0.23437500
  4  0.90909091 0.09090909
  5  0.67326733 0.32673267
  6  0.75714286 0.24285714
  8  0.70512821 0.29487179
  10 1.00000000 0.00000000
  11 0.67307692 0.32692308
```

Output 15: Proportionate table – x9 v/s y

```
> prop.table(table(M2$x10,M2$y),2)

            0         1
  1 0.7444444 0.8857143
  2 0.2555556 0.1142857
> prop.table(table(M2$x10,M2$y),1)

            0         1
  1 0.6836735 0.3163265
  2 0.8518519 0.1481481
```

Output 16: Proportionate table – x10 v/s y

```
prop.table(table(M2$x11,M2$y),2)

            0         1
1 0.11666667 0.05714286
2 0.48888889 0.31428571
3 0.23888889 0.49285714
4 0.15555556 0.13571429
prop.table(table(M2$x11,M2$y),1)

          0         1
1 0.8400000 0.1600000
2 0.8000000 0.2000000
3 0.5548387 0.4451613
4 0.7466667 0.2533333
```

Output 17: Proportionate table – x11 v/s y

```
prop.table(table(M2$x12,M2$y),2)

            0         1
1 0.06944444 0.01428571
3 0.93055556 0.98571429
prop.table(table(M2$x12,M2$y),1)

            0         1
1 0.92592593 0.07407407
3 0.70824524 0.29175476
```

```
prop.table(table(M2$x13,M2$y),2)

            0          1
1 0.06944444 0.03571429
3 0.77500000 0.82857143
4 0.15555556 0.13571429
prop.table(table(M2$x13,M2$y),1)

          0         1
1 0.8333333 0.1666667
3 0.7063291 0.2936709
4 0.7466667 0.2533333
```

Output 19: Proportionate table – x13 v/s y

```
prop.table(table(M2$x15,M2$y),2)

            0          1
4 0.002777778 0.007142857
7 0.013888889 0.000000000
8 0.983333333 0.992857143
prop.table(table(M2$x15,M2$y),1)

          0         1
4 0.5000000 0.5000000
7 1.0000000 0.0000000
8 0.7180527 0.2819473
```

Output 18: Proportionate table – x12 v/s y

Output 20: Proportionate table – x14 v/s y

```
prop.table(table(M2$x14,M2$y),2)

            0          1
4 0.019444444 0.000000000
7 0.005555556 0.007142857
8 0.975000000 0.992857143
prop.table(table(M2$x14,M2$y),1)

          0         1
4 1.0000000 0.0000000
7 0.6666667 0.3333333
8 0.7163265 0.2836735
```

```
prop.table(table(M2$x15,M2$y),2)

            0          1
4 0.002777778 0.007142857
7 0.013888889 0.000000000
8 0.983333333 0.992857143
prop.table(table(M2$x15,M2$y),1)

          0         1
4 0.5000000 0.5000000
7 1.0000000 0.0000000
8 0.7180527 0.2819473
```

Output 22: Proportionate table – x15 v/s y

```
prop.table(table(M2$x17,M2$y),2)

            0          1
2 0.008333333 0.021428571
3 0.991666667 0.978571429
prop.table(table(M2$x17,M2$y),1)

          0         1
2 0.5000000 0.5000000
3 0.7226721 0.2773279
```

Output 21: Proportionate table – x15 v/s y

Output 23: Proportionate table – x15 v/s y

```
prop.table(table(M2$x15,M2$y),2)

             0           1
4 0.002777778 0.007142857
7 0.013888889 0.000000000
8 0.983333333 0.992857143
prop.table(table(M2$x15,M2$y),1)

          0         1
4 0.5000000 0.5000000
7 1.0000000 0.0000000
8 0.7180527 0.2819473
```

```
prop.table(table(M2$x18,M2$y),2)

             0           1
2 0.991666667 0.985714286
3 0.008333333 0.014285714
prop.table(table(M2$x18,M2$y),1)

          0         1
2 0.7212121 0.2787879
3 0.6000000 0.4000000
```

Output 25: Proportionate table – x18 v/s y

```
prop.table(table(M2$x20,M2$y),2)

             0           1
3 0.45555556 0.42142857
4 0.50277778 0.53571429
7 0.04166667 0.04285714
prop.table(table(M2$x20,M2$y),1)

          0         1
3 0.7354260 0.2645740
4 0.7070312 0.2929688
7 0.7142857 0.2857143
```

Output 27: Proportionate table – x20 v/s y

Output 24: Proportionate table – x17 v/s y

```
prop.table(table(M2$x19,M2$y),2)

             0           1
1 0.13888889 0.05714286
5 0.86111111 0.94285714
prop.table(table(M2$x19,M2$y),1)

          0         1
1 0.8620690 0.1379310
5 0.7013575 0.2986425
```

Output 26: Proportionate table – x19 v/s y

```
prop.table(table(M2$x21,M2$y),2)

             0           1
1 0.06944444 0.04285714
3 0.23055556 0.12142857
4 0.39444444 0.47857143
5 0.14722222 0.24285714
6 0.15833333 0.11428571
prop.table(table(M2$x21,M2$y),1)

          0         1
1 0.8064516 0.1935484
3 0.8300000 0.1700000
4 0.6794258 0.3205742
5 0.6091954 0.3908046
6 0.7808219 0.2191781
```

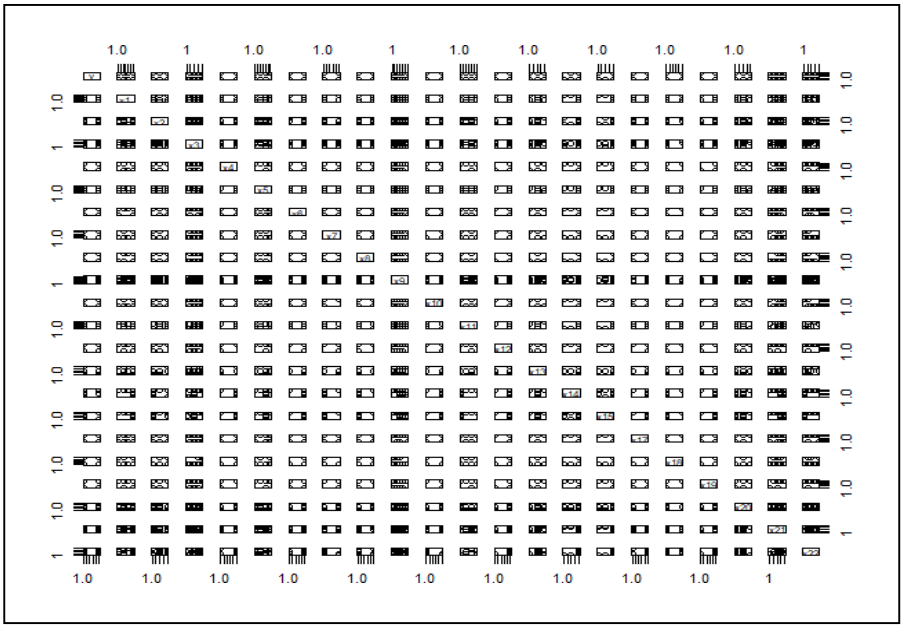
Output 28: Proportionate table – x21 v/s y

```
prop.table(table(M2$x22,M2$y),2)

             0          1
1 0.11666667 0.05714286
2 0.45277778 0.53571429
4 0.23888889 0.12142857
5 0.09166667 0.04285714
6 0.10000000 0.24285714
prop.table(table(M2$x22,M2$y),1)

             0         1
1 0.8400000 0.1600000
2 0.6848739 0.3151261
4 0.8349515 0.1650485
5 0.8461538 0.1538462
6 0.5142857 0.4857143
```

Output 29: Proportionate table – x22 v/s y

All of these Tables specify the proportions of each level of a variable as either edible or poisonous. Some levels of these variables have high proportions of being edible and others have low. While in other variables, the levels show almost equal proportions of poisonous and edible types, in which case, it implies that a mushroom chosen at random based on these features has an equal chance of being edible or poisonous.

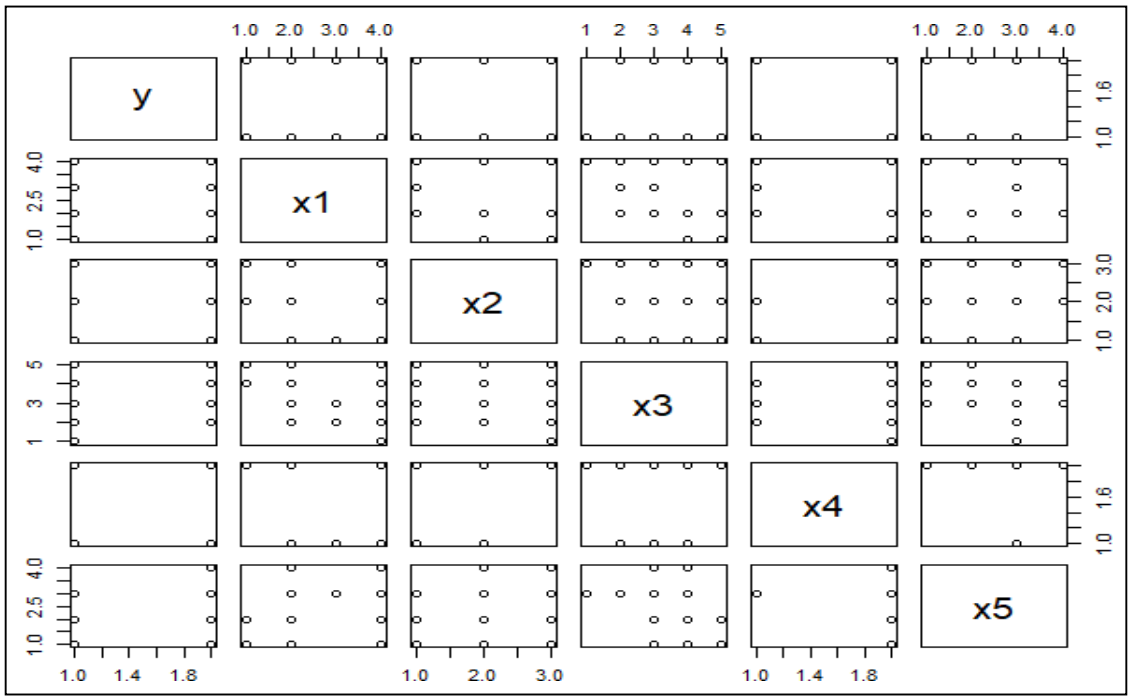
Visual representation of relations among features and target variable: The below plot shows the associations: all variables v/s all other variables



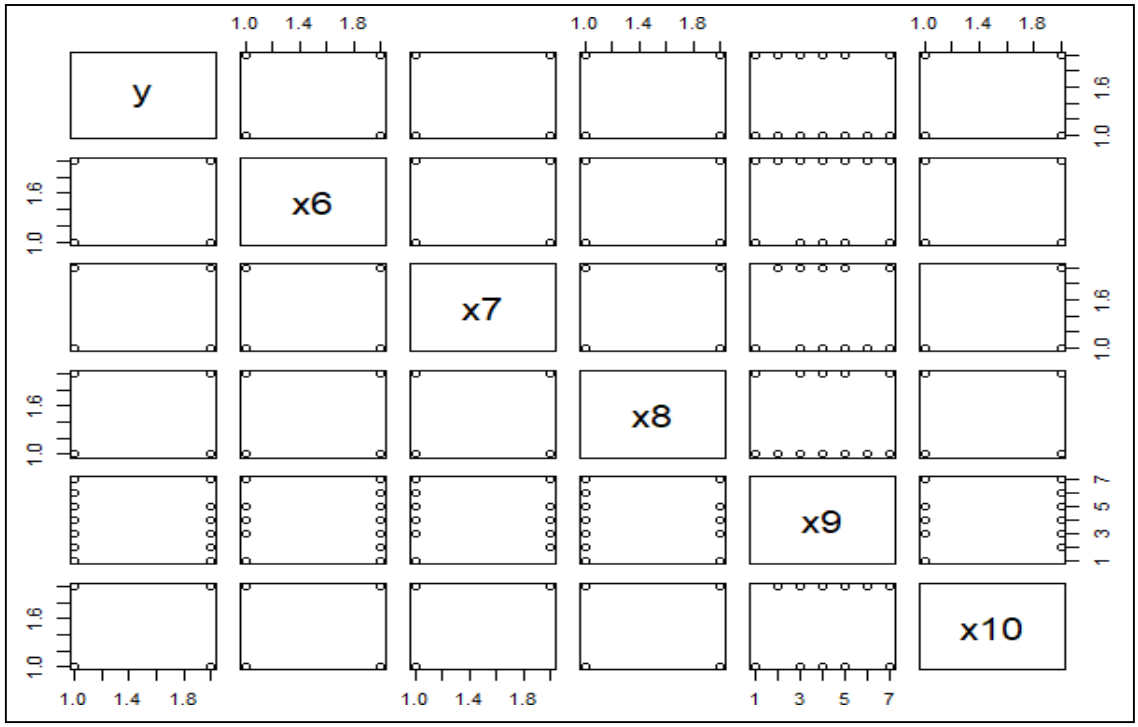
Output 30: Scatter plot

Scatter plot shows relation between all variables with every other variable Because the dataset considered consists of several features, this is unpleasant and is not helping the purpose. We have, therefore split the above diagrams as smaller subsets of plots, where the relationships can easily be identified. Below are the plots taken with 5 – 6 variables at a time along with target variable.

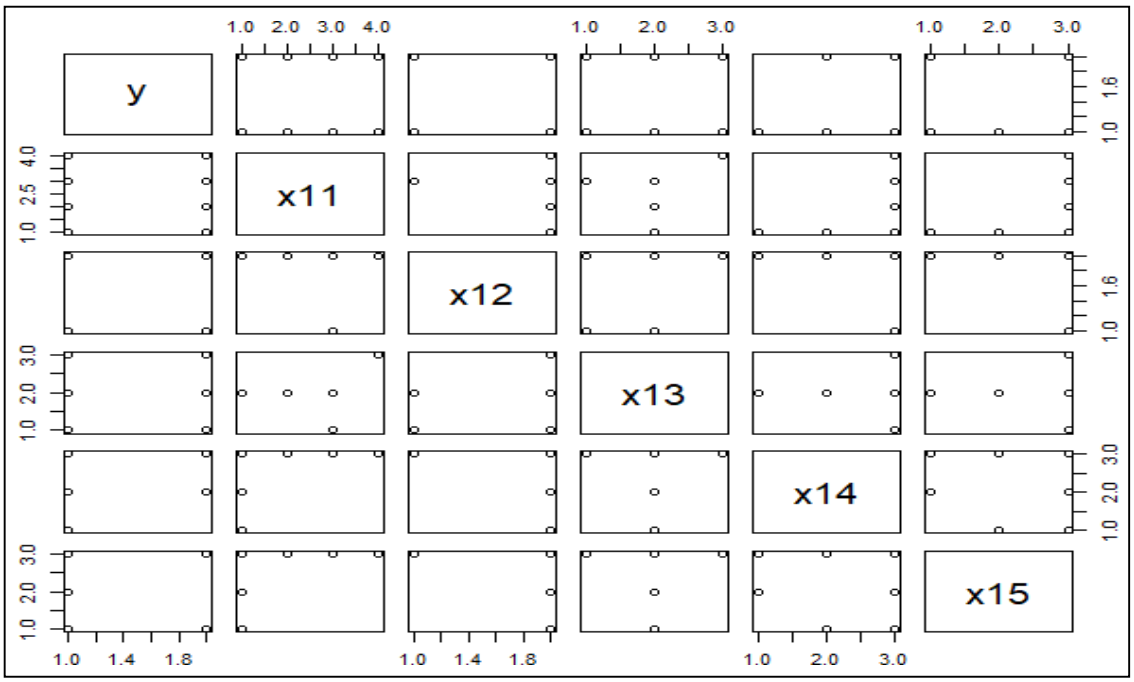
.



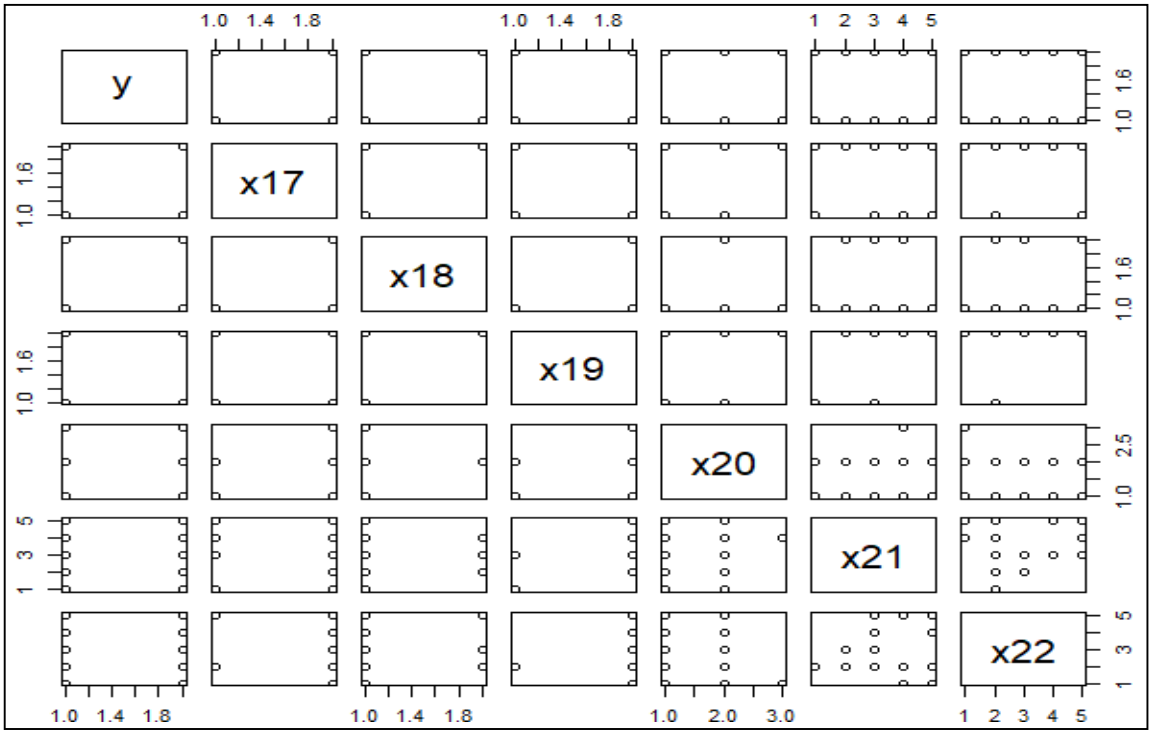
Output 31: Scatter plot, shows pairwise relations among variables x1 to x5 with y



Output 32: Scatter plot, shows pairwise relations among variables x6 to x10 with y



Output 33: Scatter plot, shows pairwise relations among variables x11 to x15 with y



Output 34: Scatter plot, shows pairwise relations among variables x17 to x22 with y

Variable significance: At this phase of modelling, we even find the significance of variables. We are using the chi-squared test of significance.

```
> # Finding if variables are significant by performing chi-squared test
> chisq.test(M2$y,M2$x1)

        Pearson's Chi-squared test

data:  M2$y and M2$x1
X-squared = 4.1043, df = 3, p-value = 0.2504

Warning message:
In chisq.test(M2$y, M2$x1) : Chi-squared approximation may be incorrect
```

Output 35: $\chi 2$ test – y v/s x1

```
> chisq.test(M2$y,M2$x2)

        Pearson's Chi-squared test

data:  M2$y and M2$x2
X-squared = 7.9055, df = 2, p-value = 0.0192
```

Output 36: $\chi 2$ test – y v/s x2

```
> chisq.test(M2$y,M2$x3)

        Pearson's Chi-squared test

data:  M2$y and M2$x3
X-squared = 15.404, df = 4, p-value = 0.003933

Warning message:
In chisq.test(M2$y, M2$x3) : Chi-squared approximation may be incorrect
```

Output 37: $\chi 2$ test – y v/s x3

```
> chisq.test(M2$y,M2$x4)

        Pearson's Chi-squared test with Yates' continuity correction

data:  M2$y and M2$x4
X-squared = 6.792, df = 1, p-value = 0.009157
```

Output 38: $\chi 2$ test – y v/s x4

```
> chisq.test(M2$y,M2$x5)

        Pearson's Chi-squared test

data:  M2$y and M2$x5
X-squared = 147.04, df = 3, p-value < 2.2e-16
```

Output 39: $\chi 2$ test – y v/s x5

```
> chisq.test(M2$y,M2$x6)

        Pearson's Chi-squared test with Yates' continuity correction

data:  M2$y and M2$x6
X-squared = 0.15207, df = 1, p-value = 0.6966

Warning message:
In chisq.test(M2$y, M2$x6) : Chi-squared approximation may be incorrect
```

Output 40: $\chi 2$ test – y v/s x6

```
> chisq.test(M2$y,M2$x7)

        Pearson's Chi-squared test with Yates' continuity correction

data:  M2$y and M2$x7
X-squared = 9.8009, df = 1, p-value = 0.001744
```

Output 41: $\chi 2$ test – y v/s x7

```
> chisq.test(M2$y,M2$x8)

        Pearson's Chi-squared test with Yates' continuity correction

data:  M2$y and M2$x8
X-squared = 41.459, df = 1, p-value = 1.204e-10
```

Output 42: $\chi 2$ test – y v/s x8

```
> chisq.test(M2$y,M2$x9)

        Pearson's Chi-squared test

data:  M2$y and M2$x9
X-squared = 6.6632, df = 6, p-value = 0.3531

Warning message:
In chisq.test(M2$y, M2$x9) : Chi-squared approximation may be incorrect
```

Output 43: $\chi 2$ test – y v/s x9

```
> chisq.test(M2$y,M2$x10)

        Pearson's Chi-squared test with Yates' continuity correction

data:  M2$y and M2$x10
X-squared = 11.06, df = 1, p-value = 0.0008822
```

Output 44: $\chi 2$ test – y v/s x10

```
> chisq.test(M2$y,M2$x11)

        Pearson's Chi-squared test

data:  M2$y and M2$x11
X-squared = 31.793, df = 3, p-value = 5.787e-07
```

Output 45: χ2 test – y v/s x11

```
> chisq.test(M2$y,M2$x12)

        Pearson's Chi-squared test with Yates' continuity correction

data:  M2$y and M2$x12
X-squared = 4.9723, df = 1, p-value = 0.02576
```

Output 46: χ2 test – y v/s x12

```
> chisq.test(M2$y,M2$x13)

        Pearson's Chi-squared test

data:  M2$y and M2$x13
X-squared = 2.5421, df = 2, p-value = 0.2805
```

Output 47: χ2 test – y v/s x13

```
> chisq.test(M2$y,M2$x14)

        Pearson's Chi-squared test

data:  M2$y and M2$x14
X-squared = 2.7973, df = 2, p-value = 0.2469

Warning message:
In chisq.test(M2$y, M2$x14) : Chi-squared approximation may be incorrect
```

Output 48: χ2 test – y v/s x14

```
> chisq.test(M2$y,M2$x15)

        Pearson's Chi-squared test

data:  M2$y and M2$x15
X-squared = 2.4339, df = 2, p-value = 0.2961

Warning message:
In chisq.test(M2$y, M2$x15) : Chi-squared approximation may be incorrect
```

Output 48: χ2 test – y v/s x15

```
> chisq.test(M2$y,M2$x17)

        Pearson's Chi-squared test with Yates' continuity correction

data:  M2$y and M2$x17
X-squared = 0.56264, df = 1, p-value = 0.4532

Warning message:
In chisq.test(M2$y, M2$x17) : Chi-squared approximation may be incorrect
```

Output 49: χ2 test – y v/s x17

```
> chisq.test(M2$y,M2$x18)

        Pearson's Chi-squared test with Yates' continuity correction

data:  M2$y and M2$x18
X-squared = 0.010021, df = 1, p-value = 0.9203

Warning message:
In chisq.test(M2$y, M2$x18) : Chi-squared approximation may be incorrect
```

Output 50: χ2 test – y v/s x18

```
> chisq.test(M2$y,M2$x19)

        Pearson's Chi-squared test with Yates' continuity correction

data:  M2$y and M2$x19
X-squared = 5.7958, df = 1, p-value = 0.01606
```

Output 51: χ2 test – y v/s x19

```
> chisq.test(M2$y,M2$x20)

        Pearson's Chi-squared test

data:  M2$y and M2$x20
X-squared = 0.4802, df = 2, p-value = 0.7866
```

Output 52: χ2 test – y v/s x20

```
> chisq.test(M2$y,M2$x21)

        Pearson's Chi-squared test

data:  M2$y and M2$x21
X-squared = 15.496, df = 4, p-value = 0.003776
```

Output 53: χ2 test – y v/s x21

```
> chisq.test(M2$y,M2$x22)

        Pearson's Chi-squared test

data:  M2$y and M2$x22
X-squared = 29.552, df = 4, p-value = 6.038e-06
```

Output 54: χ2 test – y v/s x22

As p-values (probability values) of x1, x6, x9, x13, x14, x15, x17, x18, x20 are greater than α, these variables may be insignificant in determining the dependent variable and all other variables have lesser p – values, which indicates that they significantly determine the dependent variable.

2.3 Modelling

This phase of ML is to fit an appropriate model to the data that takes few variables but explains most of the substantial differences adequately. In our case, we are trying to locate key features of mushroom that will classify it as edible or poisonous.

Initially, dividing data as train and test data. Then, we performed a 5 – fold cross validation on the train data. Later, we fitted different classification algorithms such as k-NN, SVM, RF, DT, and GLM on this train data.

Splitting the data: Here, we split our dataset into two: Train & Test.

```
> dim(M2)
[1] 500  22
> dim(Train)
[1] 400  22
> dim(Test)
[1] 100  22
```

Output 55: Splitting – Train & Test

Deciding to do a 5 – fold Cross Validation: The train data is divided into 5 samples of equal size. After division of the data 4 samples are trained to fit the model and 1 sample is tested using the fitted model. We have done this using 3 repeats.

```
> # Run algorithms using 5-fold cross validation
> library(caret)
> control = trainControl(method="repeatedcv", number=5, repeats=3)
```

Output 56: Running a 5-Fold CV

Running model pipeline: Fitting different classification algorithms and checking for their accuracies.

K-NN

```
k-Nearest Neighbors

400 samples
 21 predictor
  2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (5 fold, repeated 3 times)
Summary of sample sizes: 321, 320, 320, 319, 320, 321, ...
Resampling results across tuning parameters:

  k  Accuracy   Kappa
  1  0.7399514  0.3254608
  3  0.7740888  0.3611547
  5  0.8091958  0.4270617
  7  0.8183216  0.4409094

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 7.
```

Output 57: Employing k-NN Algorithm

DT

```
CART

400 samples
 21 predictor
  2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (5 fold, repeated 3 times)
Summary of sample sizes: 321, 320, 320, 319, 320, 321, ...
Resampling results across tuning parameters:

  cp    Accuracy   Kappa
  0.01  0.8174885  0.4378799
  0.05  0.8225096  0.4453328
  0.10  0.8225096  0.4453328

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was cp = 0.1.
```

Output 58: Employing DT Algorithm

GLM

```
Generalized Linear Model

400 samples
 21 predictor
  2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (5 fold, repeated 3 times)
Summary of sample sizes: 321, 320, 320, 319, 320, 321, ...
Resampling results:

  Accuracy   Kappa
  0.7723828  0.3490256
```

Output 59: Employing GLM Algorithm

Random Forest

```
Random Forest

400 samples
 21 predictor
  2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (5 fold, repeated 3 times)
Summary of sample sizes: 320, 319, 320, 320, 321, 321, ...
Resampling results across tuning parameters:

  mtry  Accuracy   Kappa
   2    0.8191282  0.4322720
  24    0.7775117  0.3564410
  46    0.7566879  0.3159725

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 2.
```

Output 60: Employing RF Algorithm

SVM

```
Support Vector Machines with Radial Basis Function Kernel

400 samples
 21 predictor
  2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (5 fold, repeated 3 times)
Summary of sample sizes: 321, 320, 320, 319, 320, 321, ...
Resampling results across tuning parameters:

  sigma  Accuracy   Kappa
  0.01   0.8200199  0.4373778
  0.05   0.8066855  0.3855275
  0.10   0.8024978  0.3810492

Tuning parameter 'C' was held constant at a value of 1
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were sigma = 0.01 and C = 1.
```

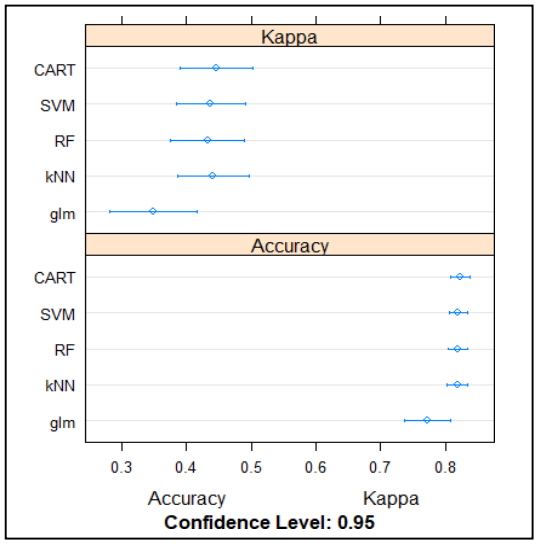
Output 61: Employing SVM Algorithm

Inspecting accuracy: After running these classification algorithms, we compare their accuracies. That is, how well are these models explaining the data. Below table displays this comparison.

```
Call:
summary.resamples(object = M3)

Models: SVM, CART, kNN, glm, RF
Number of resamples: 15

Accuracy
          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
SVM  0.7750000 0.8113133 0.8125000 0.8200199 0.8364715 0.8641975    0
CART 0.7750000 0.8113133 0.8250000 0.8225096 0.8364715 0.8765432    0
kNN  0.7777778 0.8037975 0.8227848 0.8183216 0.8312500 0.8888889    0
glm  0.5750000 0.7609968 0.7875000 0.7723828 0.8049842 0.8641975    0
RF   0.7750000 0.8062500 0.8250000 0.8191282 0.8395062 0.8500000    0

Kappa
          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
SVM  0.2436975 0.4029996 0.4241843 0.4373778 0.5003581 0.6097240    0
CART 0.2436975 0.4029996 0.4531250 0.4453328 0.5003581 0.6505608    0
kNN  0.2845927 0.3886079 0.4519326 0.4409094 0.4863289 0.6901827    0
glm  0.1099476 0.2800411 0.3474088 0.3490256 0.4094101 0.6213345    0
RF   0.2436975 0.3779026 0.4531250 0.4322720 0.5241753 0.5471698    0
```

Output 62: Comparing results of various algorithms

The visual representation of these accuracies is presented in the dot plot below



Output 63: Pictorial representation of results of various algorithms

Inference made based on accuracy: As we can notice, RF has the best accuracy of all. We can do two things at this stage: Parameter Tuning and Variable Importance.

Tuning the parameters in RF: Hyper parameters in RF are tuned to extract the best parameters for final model. Here,

Setting the number of trees as 100 or 200 or 300

Number of variables in each tree would range from 1 to 6

```
400 samples
 21 predictor
  2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (5 fold, repeated 3 times)
Summary of sample sizes: 321, 320, 320, 319, 320, 321, ...
Resampling results across tuning parameters:

  mtry  ntree  Accuracy   Kappa
  1     100    0.7225137  0.0000000
  1     200    0.7225137  0.0000000
  1     300    0.7225137  0.0000000
  2     100    0.8199991  0.4355296
  2     200    0.8208430  0.4389490
  2     300    0.8216763  0.4420297
  3     100    0.8225096  0.4453328
  3     200    0.8225096  0.4453328
  3     300    0.8225096  0.4453328
  4     100    0.8225096  0.4453328
  4     200    0.8225096  0.4453328
  4     300    0.8225096  0.4453328
  5     100    0.8216763  0.4433820
  5     200    0.8225096  0.4453328
  5     300    0.8225096  0.4453328
  6     100    0.8225096  0.4453328
  6     200    0.8225096  0.4453328
  6     300    0.8216763  0.4433820

Accuracy was used to select the optimal model using the largest value.
The final values used for the model were mtry = 3 and ntree = 100.
```

Output 64: Tuning parameters in RF

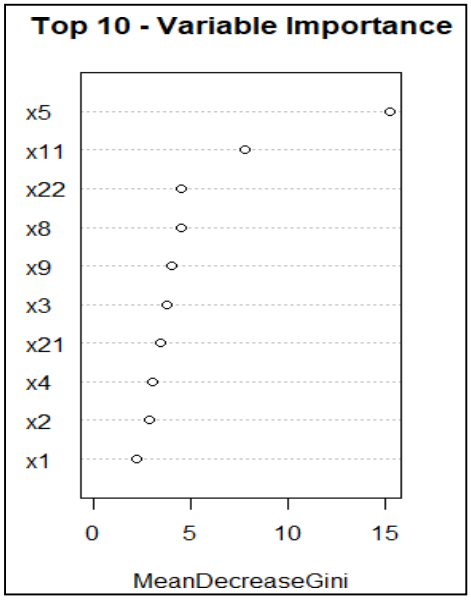
Variable Importance: Here, we identify the key features that establish whether a mushroom is edible or not.

From the following table, we observe that variables x5, x8, x11, x22 mostly influence the type of mushroom.

	MeanDecreaseGini
x1	2.1971011
x2	2.9113411
x3	3.7505024
x4	3.0454743
x5	15.2631907
x6	0.1932431
x7	1.1393105
x8	4.5285122
x9	4.0040750
x10	1.7429771
x11	7.8514497
x12	0.4340706
x13	1.5383482
x14	0.3569888
x15	0.4555700
x17	0.3381016
x18	0.1105427
x19	0.7001473
x20	1.5975524
x21	3.4782738
x22	4.5339521

Output 65: Feature Selection in RF

The plot showing variable importance is as follows



Output 66: Graphical representation of Feature Selection in RF

This feature selection plot gives the top 10 variables that prominently define a mushroom into its type.

Final Model: The final step under this phase of modelling is to fit a model with 4 key features x5, x8, x11, x22. We fit a Logistic Regression due to its simplicity to the end-user.

```
Call:
NULL

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-1.09394  -0.64743  -0.58114   0.00013   2.08643

Coefficients: (1 not defined because of singularities)
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -2.260e+00  2.015e+03  -0.001   0.9991
x54          3.932e-01  3.195e-01   1.231   0.2184
x56          8.735e-01  2.015e+03   0.000   0.9997
x57          2.093e+01  1.459e+03   0.014   0.9886
x112         5.667e-01  8.408e-01   0.674   0.5003
x113        -3.054e-01  2.015e+03   0.000   0.9999
x114         1.667e+00  8.799e-01   1.895   0.0582 .
x222        -1.204e-07  2.015e+03   0.000   1.0000
x224        -1.562e-01  2.015e+03   0.000   0.9999
x225        -1.156e+00  2.015e+03  -0.001   0.9995
x226                NA         NA      NA       NA
x82          2.036e-01  2.015e+03   0.000   0.9999
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 470.54  on 399  degrees of freedom
Residual deviance: 338.01  on 389  degrees of freedom
AIC: 360.01

Number of Fisher Scoring iterations: 17
```

Output 67: Decision to apply Logistic Regression – calculating required parameters

We have fitted the model using the 4 important features.

The following is the accuracy obtained from Logistic Regression model.

```
Generalized Linear Model

400 samples
  4 predictor
  2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (5 fold, repeated 3 times)
Summary of sample sizes: 320, 320, 320, 320, 320, 320, ...
Resampling results:

  Accuracy   Kappa
  0.8258333  0.4597917
```

Output 68: Employing Logistic Regression as the final model

2.4 Validation & Prediction

The last phase is the evaluation of the model. This is a crucial step of the ML process. This is where we can find if the algorithms perform well at any circumstances. That is, determining how well do they perform on unseen data.

Validating the accuracy of the trained data

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 290  68
         1   0  42

               Accuracy : 0.83
                 95% CI : (0.7895, 0.8655)
    No Information Rate : 0.725
    P-Value [Acc > NIR] : 5.665e-07

                  Kappa : 0.4725
 Mcnemar's Test P-Value : 4.476e-16

            Sensitivity : 1.0000
            Specificity : 0.3818
         Pos Pred Value : 0.8101
         Neg Pred Value : 1.0000
             Prevalence : 0.7250
         Detection Rate : 0.7250
   Detection Prevalence : 0.8950
      Balanced Accuracy : 0.6909

       'Positive' Class : 0
```

Output 69: Confusion Matrix – Train data

Validating the accuracy of the test data

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 70 21
         1  0  9

               Accuracy : 0.79
                 95% CI : (0.6971, 0.8651)
    No Information Rate : 0.7
    P-Value [Acc > NIR] : 0.02883

                  Kappa : 0.375
 Mcnemar's Test P-Value : 1.275e-05

            Sensitivity : 1.0000
            Specificity : 0.3000
         Pos Pred Value : 0.7692
         Neg Pred Value : 1.0000
             Prevalence : 0.7000
         Detection Rate : 0.7000
   Detection Prevalence : 0.9100
      Balanced Accuracy : 0.6500

       'Positive' Class : 0
```

Output 70: Confusion Matrix – Test data

From the above confusion matrices, we can see that both train and test data show almost nearing accuracies. We can therefore conclude that Logistic regression with 4 variables: Oduor, Habitat, Gill size, Stalk root classify the mushroom correctly 80% of the time.

4. Conclusions and Summary

Classification systems play a major role in decision-making tasks by categorizing the available information based on some principles.

In this paper, we focused on prediction accuracy. Our intent was to learn a model that has a good generalization performance on the Mushroom dataset. We recognized the characteristics of the mushroom that are best-suited for its classification. In a way, we compared and evaluated the relative performance of some well-known classification models: k-NN, Decision Trees, SVM, Random Forest, GLM, for the problem at hand.

Over all the algorithms, Random Forest gave a pre-eminent accuracy. We have then tried to tune the parameter of Random Forest Technique to obtain superior performance.

The summary of accuracies of various algorithms are as follows. Table 4.1.: Accuracy obtained from different algorithms

k-NN	SVM	Decision Trees	Random Forest	GLM
81%	80%	82%	82%	77%

After this, we have identified 4 key variables that mostly categorize the mushroom as poisonous or edible. These variables are: Odour, Gill Size, Stalk Root, Habitat. Using these, we have fitted logistic regression and obtained the final model. The accuracy of logistic regression was 83% on train data and around 80% on test data. The performance of the model is considered reasonable and we accept it.

A decision to choose Logistic regression model with 4 variables: Oduor, Gill Size, Stalk Root, Habitat has been accepted to group the mushroom as edible or poisonous.

References

Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In Proceedings of the Fifth Annual Workshop on Computational Learning Theory (pp. 144-152).

Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5-32.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. Machine Learning, 20(3), 273-297.

Cover, T., & Hart, P. (1967). Nearest neighbor pattern classification. IEEE Transactions on Information Theory, 13(1), 21-27.

Friedman, J., Hastie, T., & Tibshirani, R. (2010). Regularization Paths for Generalized Linear Models via Coordinate Descent. Journal of Statistical Software, 33(1), 1-22.

Han, J., Pei, J., & Kamber, M. (2011). Data Mining: Concepts and Techniques. Morgan Kaufmann.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning. Springer.

Ho, T. K. (1998). The random subspace method for constructing decision forests. IEEE Transactions on Pattern Analysis and Machine Intelligence, 20(8), 832-844.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An Introduction to Statistical Learning. Springer.

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In IJCAI'95: Proceedings of the 14th International Joint Conference on Artificial Intelligence (Vol. 2, pp. 1137-1145).

Kotsiantis, S. B. (2007). Supervised Machine Learning: A Review of Classification Techniques. Informatica, 31, 249-268.

Nelder, J. A., & Wedderburn, R. W. M. (1972). Generalized Linear Models. Journal of the Royal Statistical Society. Series A (General), 135(3), 370-384.

Powers, D. M. W. (2011). Evaluation: From precision, recall and F-factor to ROC, informedness, markedness & correlation. Journal of Machine Learning Technologies, 2(1), 37-63.

Quinlan, J. R. (1986). Induction of decision trees. Machine Learning, 1(1), 81-106.

Rish, I. (2001). An empirical study of the naive Bayes classifier. In IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence (Vol. 3, No. 22, pp. 41-46).