ML HW 2
Nakul Pacheriwala (np2455)

4 Major ML models were tested on the mushroom dataset.

I'll explain in short what has been done and then proceed with analysis of the result.

1. Downloaded the dataset and required libraries and loaded them.
2. Removed the columns with all values same as they don't add any information to the model
3. Split data into Train and test with a ratio of 70:30
4. One hot encoding was done as some classifiers only take numerical values or matrix as input rather than data frame with factors.
5. Generated Logistic regression models
   a. A Model was generated while using all the features in the dataset
   b. Multiple models were generated with different combinations of features (taking 5-6 at a time) only 1 example is displayed in report as all were giving similar results.
   c. Since regular GLM was unstable even after selecting few important variables lasso was used
   d. Lambda with minimum deviation was selected as the data is linearly separable
   e. The model was tested using various tools like confusion matrix, roc curve, AIC.
6. Generated SVM classifier with type as C-Classification as we had a discrete classification.
   a. C was chosen over nu as nu restricts hyperparameter values between 0-1 which was not required for our problem
   b. Linear kernel was used as it was determined that the data linearly separable, so more complex polynomial kernels were not required for the best accuracy, and linear is more cost effective while training and also running the model in future on real data.
   c. The model was tested using various tools like confusion matrix, roc curve, AIC.
7. Generated Decision tree classifier
   a. The model was tested using various tools like confusion matrix, roc curve, AIC.
8. KNN Model was generated using matrix generated by one hot encoding
   a. Different values of k were evaluated
   b. Only 1 model was kept as all smaller values gave same accuracy.
9. Observation table and comparison table were generated.

Analysis of result –

- It is observed that 3 models have 100 percent accuracy while the 4th is very close to 100
- This has been observed as the data is linearly separable and therefore even the basic models are able to predict perfectly.
- For Logistic regression, the regular model becomes unstable giving p values as 1 or 0 since data can be fully separated by a linear discriminant.
- This gives large coefficient values and a very large fisher score which is not ideal. So, Lasso regularization is used to reduce the values of coefficients.
- After user lasso we get a stable model which automatically removes the non-significant features by making their coefficients 0.
- Lasso model also converges unlike the regular glm.
- Specificity and sensitivity values are 1 as well as area under the ROC curve which makes sense as accuracy is 100 percent.
- SVM also gave very similar results, and it took less time to train compared to lasso while it also didn't require extra preprocessing like one hot encoding as SVM worked directly with factors.
- Decision tree didn't perform as well as the other models even though its accuracy is 99.3%
    - It took more time to train than others
    - While training accuracy is 100 the test accuracy is not which means it does not generalize very well.
    - Since it contains very few features is the fastest model when predicting results. However, it does not generalize very well as a small change in 1 value (which is a part of tree) would make a significant difference. For example, just changing the odour would make a mushroom poisonous which means very high value is given to just 1 feature.
    - While it appears to work very well in the given training and testing dataset for real world data with outliers and more variation it is not expected to generalize.
    - All incorrect values in this are false positives (edible were classified as poisonous) thus the model is still good enough with its drawbacks as it prevents consumption of poison with 100 percent accuracy which is more significant to the user when this model is used in real life scenarios.
- KNN gives similar results like SVM and Lasso in terms of accuracy and ability to generalize.
- Higher values of k (>50) are the point when it starts misclassifying a few values in the test dataset.
- For KNN also specificity and sensitivity values are 1 as well as area under the ROC curve which makes sense as accuracy is 100 percent like SVM and Lasso Regression.

Conclusion –

Since 3 models have exact same results it's difficult to differentiate between them based on correctness of the prediction.

Thus, metrics like training cost and time required to predict using a saved trained model would be more important factors.

Lasso takes most time to train in this specific scenario, however prediction time is quick due to simplicity of the model.

SVM trains rather quickly, also it requires least pre-processing of data, while also giving a quick and robust model.

KNN is a cheap method when it comes to running costs, however selection of K would be an added task in more complex data as it was required to test with different values of k and chose the best one based on validation accuracy (de by using the elbow method. (This wasn't useful with our dataset as it gives 100 percent accuracy for majority k values and thus it was excluded from the report also.)

However, KNN required 1 hot encoding and thus pre-processing time increases.

Methods like PCA or correlation between features was not performed as dataset has only categorical variables and these methods were not designed for them. There are some alternatives or techniques to make them work for nominal values but that does not make a lot of sense. For example, if we have 4 colours it does not mean distance between colour 1 and 4 is 3, which is what these methods would assume.

The Code and result of that is attached below for reference.

In [ ]:
```
library(tidyverse)
install.packages('caret')
install.packages("glmnet", repos = "https://cran.us.r-project.org")
library(caret)
require(gh)
library(stringr)
tmp = tempfile()
qurl = 'https://raw.githubusercontent.com/Nakul24-1/ML-Cars/main/mushrooms.cs
v'
gh(paste0('GET ', qurl), .destfile = tmp, .overwrite = TRUE)
```

In [2]:
```
library(rpart)
```

In [101]:
```
mush = read.csv(tmp,stringsAsFactors = T)
head(mush)
mush$veil.type
```

A data.frame: 6 × 23

	class	cap.shape	cap.surface	cap.color	bruises	odor	gill.attachment	gill.spacing	gill.size
	<fct>	<fct>	<fct>	<fct>	<fct>	<fct>	<fct>	<fct>	<fct>
1	p	x	s	n	t	p	f	c	n
2	e	x	s	y	t	a	f	c	b
3	e	b	s	w	t	l	f	c	b
4	p	x	y	w	t	p	f	c	n
5	e	x	s	g	f	n	f	w	b
6	e	x	y	y	t	a	f	c	b

p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p ·
p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p ·
p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p ·
p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p ·
p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p ·
p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p ·
p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p ·
p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p ·
p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p ·
p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p ·
… · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p ·
p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p ·
p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p ·
p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p ·
p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p ·
p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p ·
p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p ·
p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p ·
p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p ·
p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p · p ·
p

▶ **Levels**:

Veil type has only 1 lvl so it is not useful for us as it does not add any valuable information. Thus we will remove it

In [102]:
```
mush = mush %>% select(-veil.type)
mush_x = mush %>% select(-class)
mush_y = mush %>% select(class)
```

In [103]:
```
set.seed(121)
size<- floor(0.7*nrow(mush))
train_ind <- sample(seq_len(nrow(mush)), size = size)
train<-mush[train_ind,]
test<-mush[-train_ind,]
train_y <- as.data.frame(mush_y[train_ind,])
test_y<-as.data.frame(mush_y[-train_ind,])
names(train_y) = 'class'
names(test_y) = 'class'
```

Logistic Regression

In [104]:
```
tree.mush <-rpart(class~.,data=train)
tree.mush$variable.importance
```

odor: 2686.79001921696 **spore.print.color:** 1981.90941226942 **gill.color:** 1618.72929898891 **ring.type:** 1428.5468185695 **stalk.surface.below.ring:** 1428.5468185695 **stalk.surface.above.ring:** 1419.53074467951

I also attempted using top variables as per rpart so that comparisions can be made.

In [105]:
```
g = glm(class~  odor + gill.color + stalk.surface.below.ring + stalk.surface.a
bove.ring + ring.type  , data = train ,family = 'binomial')
g2 = glm(class~ . ,data = train ,family = 'binomial')
```

```
Warning message:
“glm.fit: fitted probabilities numerically 0 or 1 occurred”
Warning message:
“glm.fit: algorithm did not converge”
```

In [106]:

```
summary(g2)
```

```
Call:
glm(formula = class ~ ., family = "binomial", data = train)

Deviance Residuals:
       Min          1Q      Median          3Q         Max
-2.409e-06  -2.409e-06  -2.409e-06   2.409e-06   2.409e-06

Coefficients: (10 not defined because of singularities)
                            Estimate Std. Error z value Pr(>|z|)
(Intercept)               -2.657e+01  3.830e+05       0        1
cap.shapec                -6.361e-07  2.960e+05       0        1
cap.shapef                 2.569e-09  2.831e+04       0        1
cap.shapek                 5.972e-11  3.051e+04       0        1
cap.shapes                 3.041e-10  9.090e+04       0        1
cap.shapex                 2.886e-10  2.710e+04       0        1
cap.surfaceg              -4.417e-06  3.401e+05       0        1
cap.surfaces              -2.387e-09  1.606e+04       0        1
cap.surfacey              -3.175e-09  1.355e+04       0        1
cap.colorc                -8.862e-10  1.006e+05       0        1
cap.colore                -4.070e-09  4.228e+04       0        1
cap.colorg                -7.327e-09  4.041e+04       0        1
cap.colorn                -9.055e-10  4.126e+04       0        1
cap.colorp                -7.407e-11  5.455e+04       0        1
cap.colorr                 9.284e-09  1.530e+05       0        1
cap.coloru                 9.604e-09  1.407e+05       0        1
cap.colorw                 9.613e-09  4.072e+04       0        1
cap.colory                 3.198e-08  4.345e+04       0        1
bruisest                   2.657e+01  1.558e+05       0        1
odorc                      2.657e+01  1.814e+05       0        1
odorf                     -2.657e+01  4.662e+05       0        1
odorl                     -1.024e-09  3.000e+04       0        1
odorm                      1.328e+02  7.449e+05       0        1
odorn                     -7.970e+01  4.522e+05       0        1
odorp                     -5.313e+01  5.695e+05       0        1
odors                     -2.657e+01  4.669e+05       0        1
odory                     -2.657e+01  4.669e+05       0        1
gill.attachmentf          -4.419e-06  1.760e+05       0        1
gill.spacingw             -3.709e-10  6.245e+04       0        1
gill.sizen                -7.970e+01  4.284e+05       0        1
gill.colore               -2.657e+01  2.669e+05       0        1
gill.colorg               -2.657e+01  2.642e+05       0        1
gill.colorh               -2.657e+01  2.640e+05       0        1
gill.colork               -2.657e+01  2.648e+05       0        1
gill.colorn               -2.657e+01  2.644e+05       0        1
gill.coloro               -2.657e+01  2.754e+05       0        1
gill.colorp               -2.657e+01  2.637e+05       0        1
gill.colorr               -2.657e+01  2.866e+05       0        1
gill.coloru               -2.657e+01  2.652e+05       0        1
gill.colorw               -2.657e+01  2.633e+05       0        1
gill.colory               -2.657e+01  2.731e+05       0        1
stalk.shapet              -5.313e+01  2.801e+05       0        1
stalk.rootb               -2.657e+01  1.814e+05       0        1
stalk.rootc               -1.594e+02  8.647e+05       0        1
stalk.roote                2.657e+01  1.693e+05       0        1
stalk.rootr               -1.860e+02  7.410e+05       0        1
stalk.surface.above.ringk  2.850e-09  3.175e+04       0        1
stalk.surface.above.rings -5.797e-10  2.558e+04       0        1
stalk.surface.above.ringy  1.079e-06  3.777e+05       0        1
stalk.surface.below.ringk -1.016e-10  3.175e+04       0        1
stalk.surface.below.rings -1.542e-10  2.559e+04       0        1
stalk.surface.below.ringy  2.657e+01  2.263e+05       0        1
stalk.color.above.ringc           NA         NA      NA       NA
stalk.color.above.ringe    1.362e-06  7.001e+04       0        1
stalk.color.above.ringg    1.362e-06  3.725e+04       0        1
stalk.color.above.ringn    1.163e-07  2.916e+04       0        1
stalk.color.above.ringo   -5.313e+01  3.595e+05       0        1
stalk.color.above.ringp    1.362e-06  2.887e+04       0        1
stalk.color.above.ringw    1.362e-06  3.296e+04       0        1
stalk.color.above.ringy    1.594e+02  6.852e+05       0        1
stalk.color.below.ringc           NA         NA      NA       NA
stalk.color.below.ringe   -1.028e-07  6.999e+04       0        1
stalk.color.below.ringg   -1.025e-07  3.704e+04       0        1
stalk.color.below.ringn    1.376e-06  2.935e+04       0        1
stalk.color.below.ringo           NA         NA      NA       NA
```

```
stalk.color.below.ringp   -1.029e-07  2.882e+04       0        1
stalk.color.below.ringw   -1.028e-07  3.295e+04       0        1
stalk.color.below.ringy    1.373e-06  1.547e+05       0        1
veil.coloro                3.887e-11  6.286e+04       0        1
veil.colorw                       NA         NA      NA       NA
veil.colory                       NA         NA      NA       NA
ring.numbero               1.328e+02  6.912e+05       0        1
ring.numbert                      NA         NA      NA       NA
ring.typef                 5.313e+01  3.318e+05       0        1
ring.typel                        NA         NA      NA       NA
ring.typen                        NA         NA      NA       NA
ring.typep                 2.657e+01  1.438e+05       0        1
spore.print.colorh                NA         NA      NA       NA
spore.print.colork        -1.099e-10  8.843e+04       0        1
spore.print.colorn        -7.623e-11  8.731e+04       0        1
spore.print.coloro        -1.222e-10  9.071e+04       0        1
spore.print.colorr         1.328e+02  4.548e+05       0        1
spore.print.coloru        -5.214e-10  1.220e+05       0        1
spore.print.colorw         7.970e+01  4.377e+05       0        1
spore.print.colory        -7.654e-11  8.793e+04       0        1
populationc                9.677e-09  7.477e+04       0        1
populationn                7.209e-10  4.310e+04       0        1
populations               -1.443e-10  3.091e+04       0        1
populationv                9.737e-09  4.152e+04       0        1
populationy               -1.221e-09  4.302e+04       0        1
habitatg                  -1.477e-07  2.556e+04       0        1
habitatl                  -9.032e-08  2.352e+04       0        1
habitatm                  -1.470e-07  4.382e+04       0        1
habitatp                  -1.803e-07  1.866e+04       0        1
habitatu                  -1.494e-07  4.388e+04       0        1
habitatw                          NA         NA      NA       NA

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 7.8777e+03  on 5685  degrees of freedom
Residual deviance: 3.2988e-08  on 5600  degrees of freedom
AIC: 172

Number of Fisher Scoring iterations: 25
```

In [107]:
```
summary(g)
```

```
Call:
glm(formula = class ~ odor + gill.color + stalk.surface.below.ring + 
    stalk.surface.above.ring + ring.type, family = "binomial", 
    data = train)

Deviance Residuals: 
     Min        1Q    Median        3Q       Max  
-0.87833  -0.00001   0.00000   0.00000   2.48412  

Coefficients: (1 not defined because of singularities)
                            Estimate Std. Error z value Pr(>|z|)
(Intercept)               -1.140e+02  1.904e+04  -0.006    0.995
odorc                      1.091e+02  1.380e+04   0.008    0.994
odorf                      1.407e+02  1.341e+04   0.010    0.992
odorl                     -2.674e-01  7.861e+03   0.000    1.000
odorm                      6.475e+01  3.233e+04   0.002    0.998
odorn                      6.108e+01  7.895e+03   0.008    0.994
odorp                      1.086e+02  1.242e+04   0.009    0.993
odors                      1.407e+02  1.534e+04   0.009    0.993
odory                      1.408e+02  1.537e+04   0.009    0.993
gill.colore               -5.777e+00  2.063e+04   0.000    1.000
gill.colorg                1.792e+01  1.286e+04   0.001    0.999
gill.colorh               -3.766e+00  1.318e+04   0.000    1.000
gill.colork               -3.922e+00  1.455e+04   0.000    1.000
gill.colorn               -4.938e+00  1.350e+04   0.000    1.000
gill.coloro               -5.776e+00  2.509e+04   0.000    1.000
gill.colorp               -4.506e+00  1.253e+04   0.000    1.000
gill.colorr                4.336e+01  3.619e+04   0.001    0.999
gill.coloru               -5.589e+00  1.446e+04   0.000    1.000
gill.colorw                1.575e+01  1.286e+04   0.001    0.999
gill.colory                1.546e+01  1.286e+04   0.001    0.999
stalk.surface.below.ringk -2.340e+00  6.080e+03   0.000    1.000
stalk.surface.below.rings  1.706e+01  4.971e+03   0.003    0.997
stalk.surface.below.ringy  5.988e+01  7.472e+03   0.008    0.994
stalk.surface.above.ringk -1.625e+00  5.690e+03   0.000    1.000
stalk.surface.above.rings  1.708e+01  5.010e+03   0.003    0.997
stalk.surface.above.ringy -2.346e+01  7.498e+03  -0.003    0.998
ring.typef                -2.850e+00  1.915e+04   0.000    1.000
ring.typel                 5.455e+00  1.194e+04   0.000    1.000
ring.typen                        NA         NA      NA       NA
ring.typep                -2.523e-04  5.330e-01   0.000    1.000

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 7877.74  on 5685  degrees of freedom
Residual deviance:  256.94  on 5657  degrees of freedom
AIC: 314.94

Number of Fisher Scoring iterations: 23
```

It can be seen that no of fisher scoring iterations are still more than 20. Ideally it should in the range near 3-5.

In [108]:
```
test5 <- test
test5$model_prob <- predict(g2, newdata = test5, type = "response")
library(dplyr)
test5 <- test5 %>% mutate(model_pred = 1*(model_prob > .51) + 0,
                          target_binary = 1*(class == 'p') + 0)

head(test5)
```

```
Warning message in predict.lm(object, newdata, se.fit, scale = 1, type = if
(type == :
“prediction from a rank-deficient fit may be misleading”
```

A data.frame: 6 × 25

	class	cap.shape	cap.surface	cap.color	bruises	odor	gill.attachment	gill.spacing	gill.size
	<fct>	<fct>	<fct>	<fct>	<fct>	<fct>	<fct>	<fct>	<fct>
7	e	b	s	w	t	a	f	c	b
14	p	x	y	w	t	p	f	c	n
24	e	b	y	w	t	a	f	c	b
33	e	x	y	y	t	l	f	c	b
41	e	b	y	y	t	a	f	c	b
45	e	x	s	y	t	a	f	c	b

In [109]:
```
test5 <- test5 %>% mutate(accurate = 1*(model_pred == target_binary))
sum(test5$accurate)/nrow(test5)
```

1

The data is linearly seperable, in this scenario as we can see the glm becomes very unstable giving huge values of coefficients and p values as 1.

Its known fact that logistic regression becomes unstable in such scenairos where accuracy is 100 percent.

In [110]:
```
coef(g2)
```

(Intercept): -26.5660704098226 **cap.shapec:** -6.36105420169476e-07 **cap.shapef:** 2.56919818559209e-09 **cap.shapek:** 5.97159530607212e-11 **cap.shapes:** 3.04084666392916e-10 **cap.shapex:** 2.8855945191767e-10 **cap.surfaceg:** -4.41655822323808e-06 **cap.surfaces:** -2.38663608560394e-09 **cap.surfacey:** -3.17499570066852e-09 **cap.colorc:** -8.86169357213621e-10 **cap.colore:** -4.06965983102533e-09 **cap.colorg:** -7.32723350272211e-09 **cap.colorn:** -9.05493264214312e-10 **cap.colorp:** -7.40740129398713e-11 **cap.colorr:** 9.2840258995409e-09 **cap.coloru:** 9.60411101642816e-09 **cap.colorw:** 9.61331544619755e-09 **cap.colory:** 3.19812979924448e-08 **bruisest:** 26.5660693305519 **odorc:** 26.5660718597924 **odorf:** -26.5660668114653 **odorl:** -1.02443082741425e-09 **odorm:** 132.830339606492 **odorn:** -79.6982042924686 **odorp:** -53.132136282038 **odors:** -26.566066802902 **odory:** -26.5660668034097 **gill.attachmentf:** -4.41910035834145e-06 **gill.spacingw:** -3.70919712327886e-10 **gill.sizen:** -79.69820432063 **gill.colore:** -26.5660668905843 **gill.colorg:** -26.5660668762063 **gill.colorh:** -26.5660668769947 **gill.colork:** -26.5660668860758 **gill.colorn:** -26.5660668901292 **gill.coloro:** -26.5660668902667 **gill.colorp:** -26.5660668955002 **gill.colorr:** -26.5660667258762 **gill.coloru:** -26.5660668919633 **gill.colorw:** -26.56606689056 **gill.colory:** -26.5660668904843 **stalk.shapet:** -53.132134841037 **stalk.rootb:** -26.5660692970034 **stalk.rootc:** -159.396408307347 **stalk.roote:** 26.566069656277 **stalk.rootr:** -185.962476206452 **stalk.surface.above.ringk:** 2.85021408885526e-09 **stalk.surface.above.rings:** -5.79691739654269e-10 **stalk.surface.above.ringy:** 1.07876954020241e-06 **stalk.surface.below.ringk:** -1.01629090424258e-10 **stalk.surface.below.rings:** -1.54214517366403e-10 **stalk.surface.below.ringy:** 26.5660679275484 **stalk.color.above.ringc:** <NA> **stalk.color.above.ringe:** 1.36231640517323e-06 **stalk.color.above.ringg:** 1.36240431643255e-06 **stalk.color.above.ringn:** 1.16326615215684e-07 **stalk.color.above.ringo:** -53.1321378858593 **stalk.color.above.ringp:** 1.36238999487254e-06 **stalk.color.above.ringw:** 1.36245357002663e-06 **stalk.color.above.ringy:** 159.396404160901 **stalk.color.below.ringc:** <NA> **stalk.color.below.ringe:** -1.02785754614283e-07 **stalk.color.below.ringg:** -1.02532274062461e-07 **stalk.color.below.ringn:** 1.37590207314379e-06 **stalk.color.below.ringo:** <NA> **stalk.color.below.ringp:** -1.02907122151561e-07 **stalk.color.below.ringw:** -1.02750383051259e-07 **stalk.color.below.ringy:** 1.37299438312488e-06 **veil.coloro:** 3.88675959626273e-11 **veil.colorw:** <NA> **veil.colory:** <NA> **ring.numbero:** 132.830341557965 **ring.numbert:** <NA> **ring.typef:** 53.132138977852 **ring.typel:** <NA> **ring.typen:** <NA> **ring.typep:** 26.5660694808903 **spore.print.colorh:** <NA> **spore.print.colork:** -1.09866162054492e-10 **spore.print.colorn:** -7.62270388250565e-11 **spore.print.coloro:** -1.22184205311379e-10 **spore.print.colorr:** 132.830344182786 **spore.print.coloru:** -5.21390675122386e-10 **spore.print.colorw:** 79.6982068924286 **spore.print.colory:** -7.65407597508697e-11 **populationc:** 9.67665261044367e-09 **populationn:** 7.20915527799204e-10 **populations:** -1.44300942813435e-10 **populationv:** 9.73733276455428e-09 **populationy:** -1.22087387860268e-09 **habitatg:** -1.47678896347949e-07 **habitatl:** -9.03163744844645e-08 **habitatm:** -1.4702328173093e-07 **habitatp:** -1.80251382988426e-07 **habitatu:** -1.49442514437625e-07 **habitatw:** <NA>

We will try and use regularization to help logistic regression converge and we will use lasso so that we can remove the unnecessary variables without any need for manual selection.

As the variable importance we got in rpart is makes more sense for decision trees compared to probablistic models.

In [111]:
```
library(glmnet)
```

In [112]:
```
X = model.matrix(class ~ ., train)[, -1]
X2 = model.matrix(class ~ ., test)[, -1]

Y = train_y %>%
  mutate(Edible = ifelse(class=='e', 1, 0),
         Poison = ifelse(class=='p', 1, 0))

Y = model.matrix(class~. ,Y)[,-1]
Y2 = test_y %>%
  mutate(Edible = ifelse(class=='e', 1, 0),
         Poison = ifelse(class=='p', 1, 0))
Y2 = model.matrix(class~. ,Y2)[,-1]

fit_lasso = cv.glmnet(X, Y, alpha = 1,family = 'binomial')
```

In [113]:
```
true_test_y = 1*(test$class == 'p')
true_test_y
```

```
0 · 1 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 1 · 0 · 0 · 0 · 0 · 0 · 0 ·
0 · 0 · 0 · 1 · 0 · 0 · 0 · 0 · 1 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 1 ·
0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 1 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 1 · 0 · 0 · 0 ·
0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 1 · 0 · 0 · 0 · 1 · 0 · 0 · 0 · 0 · 0 ·
0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 1 · 0 · 0 · 0 · 0 · 0 · 0 · 0 ·
0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 1 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 1 · 0 · 0 · 0 ·
0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 ·
0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 1 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 ·
1 · 0 · 0 · 0 · 1 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 1 · 0 · 1 · 0 · 0 · 0 · 0 · 1 ·
0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 0 ·
⋯ · 0 · 0 · 0 · 1 · 1 · 1 · 1 · 1 · 0 · 1 · 1 · 1 · 1 · 1 · 1 · 1 · 1 · 1 · 1 ·
1 · 0 · 1 · 1 · 0 · 1 · 1 · 0 · 0 · 0 · 0 · 0 · 0 · 1 · 1 · 1 · 1 · 0 · 0 · 1 ·
0 · 1 · 1 · 0 · 1 · 1 · 1 · 1 · 1 · 1 · 0 · 1 · 1 · 0 · 1 · 1 · 0 · 0 · 1 · 0 ·
0 · 1 · 1 · 0 · 1 · 1 · 1 · 1 · 0 · 1 · 0 · 1 · 1 · 1 · 0 · 1 · 1 · 1 · 0 · 0 ·
1 · 0 · 0 · 1 · 1 · 0 · 1 · 0 · 0 · 0 · 1 · 1 · 1 · 1 · 1 · 0 · 1 · 1 · 0 · 0 ·
0 · 0 · 1 · 1 · 0 · 0 · 0 · 0 · 1 · 1 · 0 · 0 · 1 · 0 · 1 · 0 · 1 · 0 · 1 · 1 ·
0 · 0 · 1 · 0 · 1 · 0 · 0 · 1 · 0 · 0 · 1 · 1 · 1 · 0 · 0 · 0 · 1 · 0 · 0 · 1 ·
0 · 1 · 0 · 1 · 0 · 1 · 0 · 0 · 1 · 1 · 1 · 1 · 1 · 1 · 0 · 1 · 0 · 1 · 0 · 0 ·
0 · 0 · 0 · 0 · 0 · 1 · 1 · 1 · 0 · 1 · 0 · 1 · 1 · 1 · 0 · 0 · 1 · 0 · 0 · 1 ·
0 · 1 · 1 · 0 · 0 · 1 · 0 · 0 · 0 · 1 · 1 · 0 · 1 · 0 · 0 · 0 · 0 · 0 · 0 · 1 ·
0
```

Here 10 cross-validated lasso regression is used and I'll choose the min lambda value to minimize the prediction error.

In [114]:

```
summary(fit_lasso)
coef(fit_lasso, s = "lambda.min")
```

```
           Length Class  Mode
lambda     86     -none- numeric
cvm        86     -none- numeric
cvsd       86     -none- numeric
cvup       86     -none- numeric
cvlo       86     -none- numeric
nzero      86     -none- numeric
call        5     -none- call
name        1     -none- character
glmnet.fit 13     lognet list
lambda.min  1     -none- numeric
lambda.1se  1     -none- numeric
index       2     -none- numeric
```

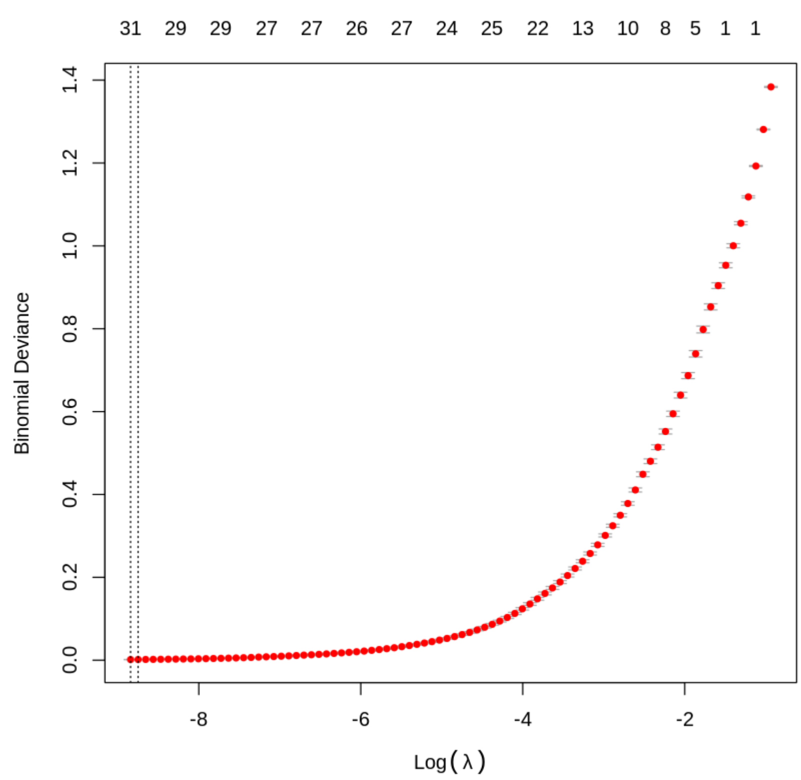
```
96 x 1 sparse Matrix of class "dgCMatrix"
                                    s1
(Intercept)                -4.40974547
cap.shapec                   .
cap.shapef                   .
cap.shapek                   .
cap.shapes                 -0.20427286
cap.shapex                   .
cap.surfaceg                3.52683681
cap.surfaces                 .
cap.surfacey                 .
cap.colorc                 -1.78302172
cap.colore                   .
cap.colorg                   .
cap.colorn                 -0.08674065
cap.colorp                   .
cap.colorr                   .
cap.coloru                   .
cap.colorw                   .
cap.colory                   .
bruisest                     .
odorc                      10.65551105
odorf                      11.09872776
odorl                      -1.08663328
odorm                        .
odorn                      -5.61776703
odorp                       7.50407705
odors                       0.62184682
odory                       0.63009212
gill.attachmentf             .
gill.spacingw              -4.12190631
gill.sizen                  4.26974698
gill.colore                  .
gill.colorg                  .
gill.colorh                  .
gill.colork                  .
gill.colorn                  .
gill.coloro                  .
gill.colorp                  .
gill.colorr                  .
gill.coloru                  .
gill.colorw                  .
gill.colory                  .
stalk.shapet                 .
stalk.rootb                  .
stalk.rootc                -2.31763171
stalk.roote                  .
stalk.rootr                -2.75540058
stalk.surface.above.ringk   2.50986730
stalk.surface.above.rings    .
stalk.surface.above.ringy    .
stalk.surface.below.ringk    .
stalk.surface.below.rings    .
stalk.surface.below.ringy   0.75427483
stalk.color.above.ringc      .
stalk.color.above.ringe      .
stalk.color.above.ringg      .
stalk.color.above.ringn      .
stalk.color.above.ringo    -1.05710481
stalk.color.above.ringp      .
stalk.color.above.ringw      .
stalk.color.above.ringy     2.98204451
stalk.color.below.ringc      .
stalk.color.below.ringe      .
stalk.color.below.ringg      .
stalk.color.below.ringn    -0.29538317
stalk.color.below.ringo    -0.02454767
stalk.color.below.ringp      .
stalk.color.below.ringw      .
stalk.color.below.ringy     2.36716453
veil.coloro                  .
veil.colorw                  .
veil.colory                 0.00509127
ring.numbero                 .
```

```
ring.numbert              -2.24529760
ring.typef                -0.58849681
ring.typel                 .
ring.typen                 .
ring.typep                 .
spore.print.colorh         .
spore.print.colork         .
spore.print.colorn        -0.31908052
spore.print.coloro         .
spore.print.colorr        18.71596855
spore.print.coloru        -1.56330630
spore.print.colorw         6.51642952
spore.print.colory         .
populationc                4.90458406
populationn                .
populations                .
populationv                .
populationy                .
habitatg                   .
habitatl                   .
habitatm                   .
habitatp                   .
habitatu                   .
habitatw                  -6.10539360
```

Using Lasso Regression reduced the number of varibles and also provided smaller coefficient values.

Smaller coefficient values here less complex model, thus less overfitting. It also means that the variance is less.

In [115]:
```
plot(fit_lasso)
```



In [116]:
```
y_prob = predict(fit_lasso, X2, s = "lambda.min",type = "response")
y_pred = 1*(y_prob > .50)
Y_df = as.data.frame(Y)
head(y_pred)

dim(y_pred)
head(true_test_y)
```

A matrix: 6 × 1 of type dbl

	lambda.min
7	0
14	1
24	0
33	0
41	0
45	0

2438 · 1

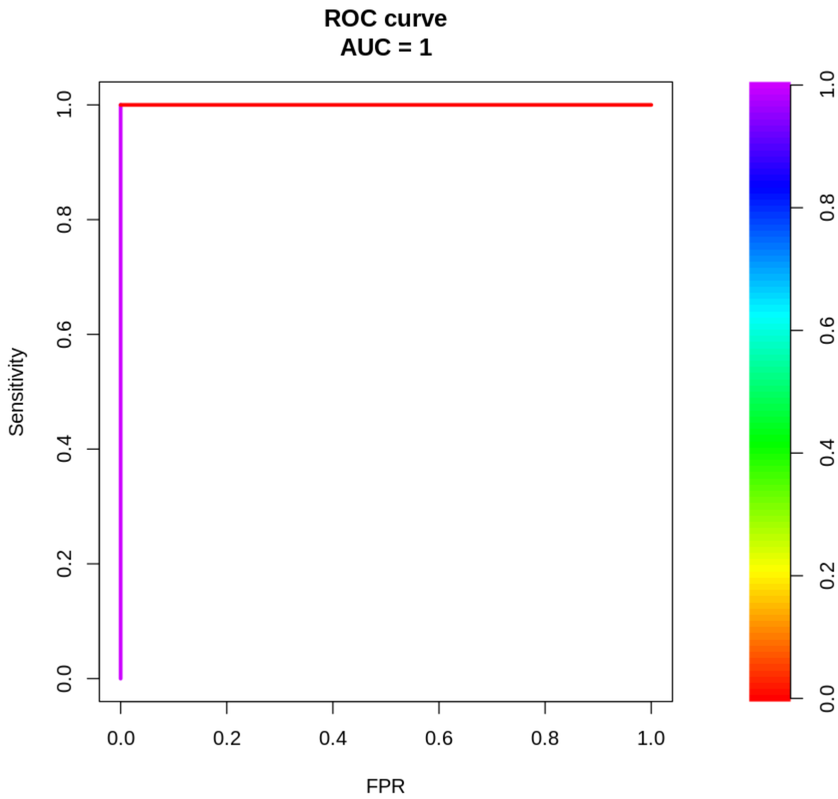
0 · 1 · 0 · 0 · 0 · 0

In [117]:
```
cm <- confusionMatrix(factor(y_pred), reference = factor(true_test_y))
cm
```

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1283    0
         1    0 1155

               Accuracy : 1
                 95% CI : (0.9985, 1)
    No Information Rate : 0.5263
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.5263
         Detection Rate : 0.5263
   Detection Prevalence : 0.5263
      Balanced Accuracy : 1.0000

       'Positive' Class : 0
```

In [118]:
```
install.packages('PRROC')
library(PRROC)

PRROC_obj <- roc.curve(scores.class0 = y_pred, weights.class0=true_test_y,
                       curve=TRUE)
plot(PRROC_obj)
```

```
Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)
```



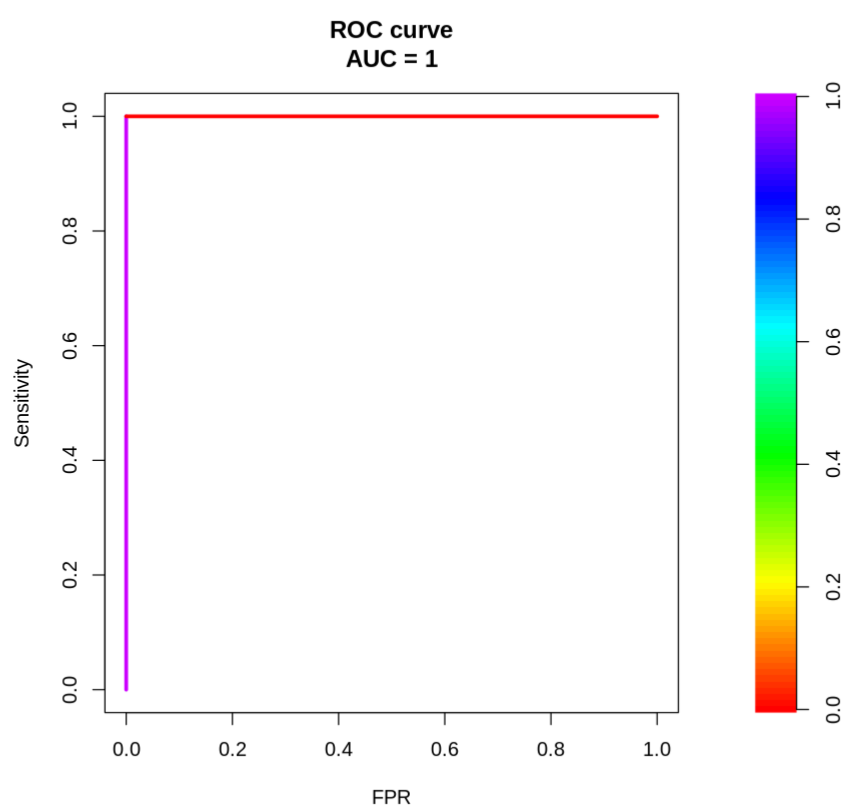
In [119]:
```
names(y_pred) = 'y_pred_lasso_bin'
```

SVM

In [120]:
```
library(e1071)
        classifier_svm = svm(formula = class ~ .,
                             data = train,
                             type = 'C-classification',
                             kernel = 'linear')
        y_pred_svm = predict(classifier_svm, newdata = test[-1])
        confusionMatrix(test$class, y_pred_svm)
```

```
Confusion Matrix and Statistics

          Reference
Prediction    e    p
         e 1283    0
         p    0 1155

               Accuracy : 1
                 95% CI : (0.9985, 1)
    No Information Rate : 0.5263
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.5263
         Detection Rate : 0.5263
   Detection Prevalence : 0.5263
      Balanced Accuracy : 1.0000

       'Positive' Class : e
```

In [121]:
```
true_test_y = 1*(test$class == 'p')
y_pred_svm_bin = 1*(y_pred_svm == 'p')
PRROC_obj <- roc.curve(scores.class0 = y_pred_svm_bin, weights.class0 = true_t
est_y,
                       curve=TRUE)
plot(PRROC_obj)
```



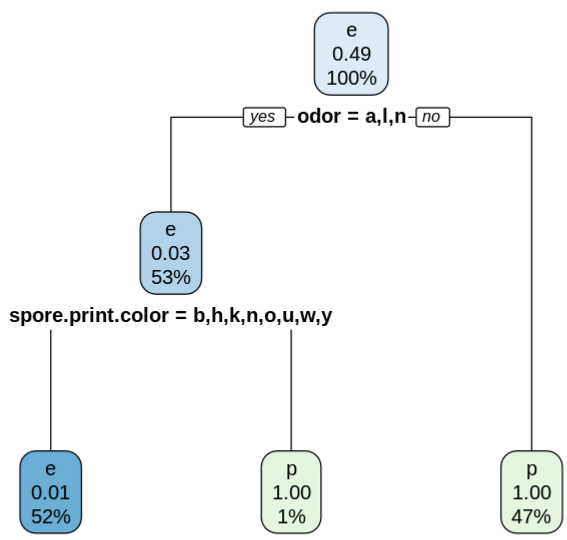
In [122]:

Decision tree

We will use rpart to formulate the tree

In [123]:
```
install.packages("rpart.plot", repos = "https://cran.us.r-project.org")
library(rpart.plot)
tree.mush <-rpart(class~.,data=train)
rpart.plot(tree.mush,extra= 106)
```

```
Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)
```



In [124]:
```
y_pred_dt = predict(tree.mush, newdata = test[-1],type = 'class')
y_pred_dt
```

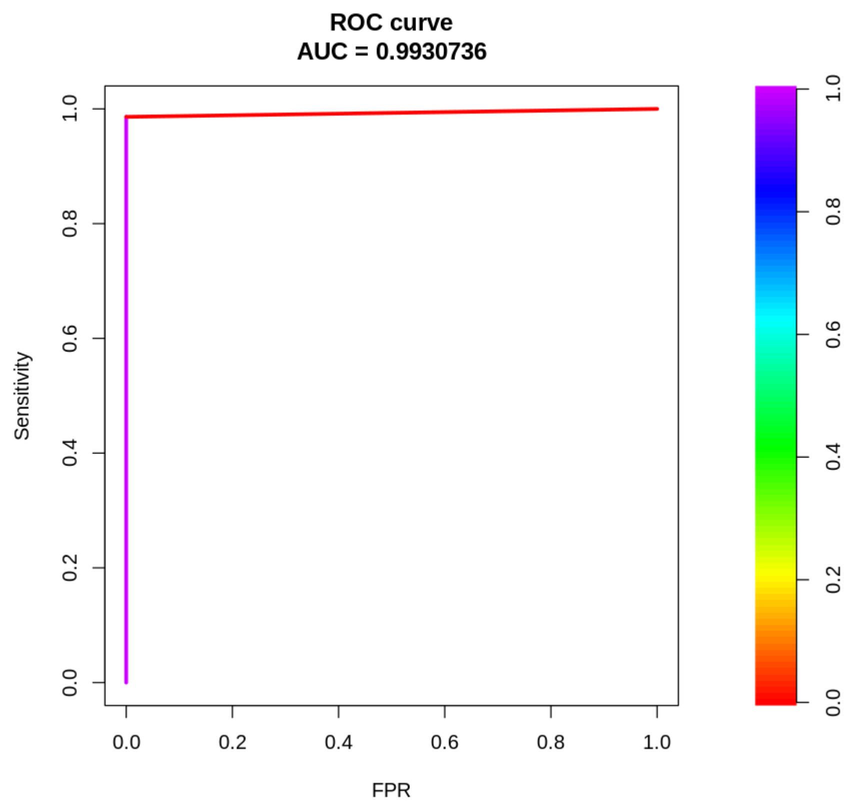
7: e **14:** p **24:** e **33:** e **41:** e **45:** e **46:** e **48:** e **53:** e **57:** e **61:** e **72:** e **80:** e **82:** p **88:** e **90:** e **94:** e **100:** e **105:** e **107:** e **109:** e **112:** e **114:** e **121:** p **122:** e **125:** e **135:** e **137:** e **139:** p **142:** e **147:** e **157:** e **162:** e **164:** e **168:** e **169:** e **174:** e **179:** e **180:** e **186:** p **187:** e **188:** e **190:** e **191:** e **196:** e **202:** e **204:** e **205:** e **206:** p **207:** e **212:** e **216:** e **217:** e **218:** e **219:** e **221:** e **229:** p **233:** e **236:** e **237:** e **238:** e **241:** e **246:** e **250:** e **251:** e **254:** e **255:** e **258:** e **259:** e **261:** e **262:** p **266:** e **268:** e **269:** e **272:** p **274:** e **275:** e **276:** e **278:** e **287:** e **289:** e **295:** e **299:** e **303:** e **304:** e **306:** e **308:** e **311:** e **317:** e **324:** e **325:** e **330:** e **331:** p **332:** e **333:** e **336:** e **339:** e **341:** e **349:** e **368:** e **369:** e **370:** e **371:** e **374:** e **378:** e **379:** e **380:** e **382:** e **386:** p **387:** e **388:** e **390:** e **391:** e **392:** e **394:** e **399:** e **403:** p **405:** e **407:** e **419:** e **420:** e **424:** e **428:** e **431:** e **439:** e **441:** e **444:** e **445:** e **447:** e **450:** e **452:** e **463:** e **464:** e **465:** e **467:** e **471:** e **480:** e **481:** e **491:** e **495:** e **499:** e **502:** e **507:** e **508:** e **511:** e **513:** e **517:** e **518:** e **519:** e **522:** e **524:** p **525:** e **526:** e **528:** e **538:** e **544:** e **545:** e **551:** e **553:** e **556:** e **557:** p **561:** e **564:** e **565:** e **569:** p **572:** e **574:** e **577:** e **578:** e **581:** e **582:** e **592:** e **594:** p **597:** e **600:** p **603:** e **607:** e **611:** e **612:** e **614:** p **615:** e **621:** e **624:** e **626:** e **628:** e **631:** e **635:** e **636:** e **638:** e **639:** e **648:** e **662:** e **677:** e **678:** e **680:** e **683:** e **685:** e **686:** e **690:** e **691:** e **692:** ⋯ **693:** e **695:** e **698:** e **699:** p **701:** p **702:** p **703:** p **704:** p **707:** e **715:** p **723:** p **729:** p **732:** p **733:** p **734:** p **736:** p **745:** p **750:** p **751:** p **752:** p **755:** e **756:** p **759:** p **762:** e **766:** p **769:** p **774:** e **775:** e **779:** e **783:** e **788:** e **791:** e **796:** p **801:** p **803:** p **804:** p **807:** e **808:** e **809:** p **816:** e **818:** p **828:** p **832:** e **833:** p **835:** p **840:** p **842:** p **843:** p **857:** p **863:** e **870:** p **876:** p **882:** e **885:** p **889:** p **890:** e **892:** e **895:** p **896:** e **897:** e **902:** p **903:** p **907:** e **913:** p **924:** p **929:** p **930:** p **932:** e **933:** p **939:** e **940:** p **943:** p **945:** p **946:** e **947:** p **950:** p **955:** p **959:** e **960:** e **962:** p **965:** e **969:** e **970:** e **971:** p **976:** e **978:** p **984:** e **986:** e **992:** e **998:** p **1001:** p **1011:** p **1017:** p **1022:** p **1023:** e **1026:** p **1030:** p **1033:** e **1034:** e **1035:** e **1037:** e **1038:** p **1046:** p **1048:** e **1049:** e **1052:** e **1053:** e **1056:** p **1065:** p **1066:** e **1067:** e **1073:** p **1076:** e **1079:** p **1082:** e **1085:** p **1087:** e **1096:** p **1099:** p **1101:** e **1102:** e **1112:** p **1113:** e **1117:** p **1120:** e **1122:** e **1125:** p **1127:** e **1128:** e **1129:** p **1137:** p **1140:** p **1142:** e **1148:** e **1153:** e **1154:** p **1157:** e **1160:** e **1161:** p **1164:** e **1168:** p **1174:** e **1175:** p **1179:** e **1183:** p **1189:** e **1190:** e **1191:** p **1193:** p **1194:** p **1196:** p **1200:** p **1202:** p **1204:** e **1210:** p **1212:** e **1214:** p **1215:** e **1217:** e **1226:** e **1228:** e **1230:** e **1233:** e **1239:** e **1240:** p **1242:** p **1243:** p **1245:** e **1246:** p **1252:** e **1256:** p **1257:** p **1258:** p **1259:** e **1261:** e **1262:** p **1263:** e **1264:** e **1265:** p **1266:** e **1268:** p **1275:** p **1279:** e **1281:** e **1283:** p **1284:** e **1285:** e **1295:** e **1296:** p **1299:** p **1302:** e **1305:** p **1308:** e **1309:** e **1316:** e **1317:** e **1321:** e **1330:** e **1333:** p **1336:** e

▶ **Levels**:

In [125]:
```
confusionMatrix(y_pred_dt,test$class)
```

```
Confusion Matrix and Statistics

          Reference
Prediction    e    p
         e 1283   16
         p    0 1139

               Accuracy : 0.9934
                 95% CI : (0.9894, 0.9962)
    No Information Rate : 0.5263
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9868

 Mcnemar's Test P-Value : 0.0001768

            Sensitivity : 1.0000
            Specificity : 0.9861
         Pos Pred Value : 0.9877
         Neg Pred Value : 1.0000
             Prevalence : 0.5263
         Detection Rate : 0.5263
   Detection Prevalence : 0.5328
      Balanced Accuracy : 0.9931

       'Positive' Class : e
```

In [126]:
```
y_pred_tree_bin = 1*(y_pred_dt == 'p')
PRROC_obj <- roc.curve(scores.class0 = y_pred_tree_bin, weights.class0=true_te
st_y,
                       curve=TRUE)
plot(PRROC_obj)
```



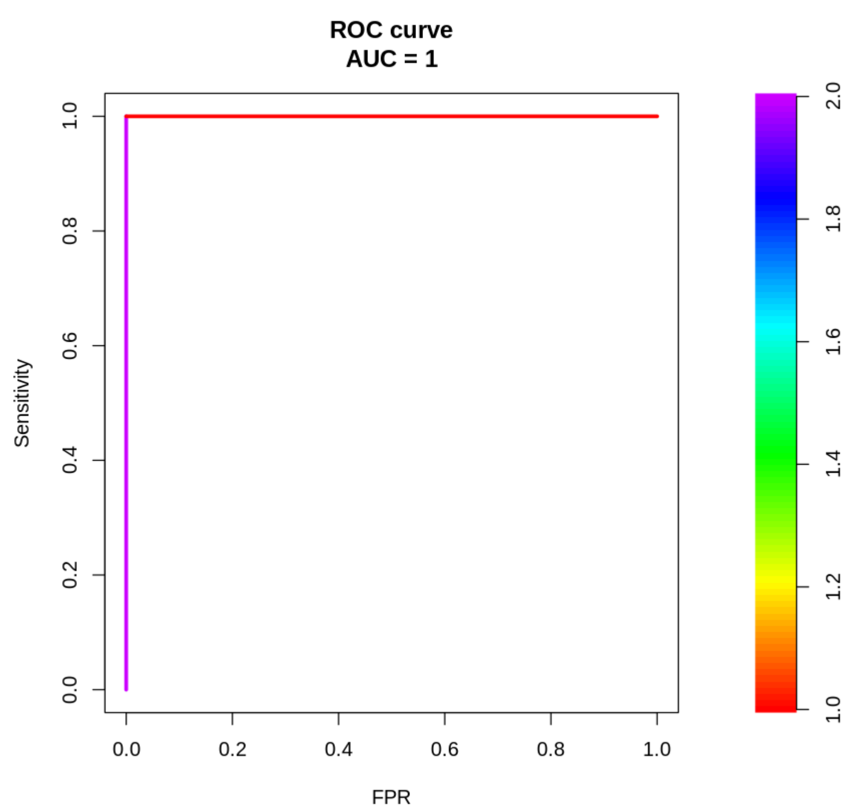
KNN

In [127]:
```
library(class)
set.seed(121)
size<- floor(0.7*nrow(mush))
train_ind <- sample(seq_len(nrow(mush)), size = size)
train<-mush[train_ind,]
test<-mush[-train_ind,]
y2 <- train %>% select(class)
Xtr = as.data.frame( model.matrix(class ~ ., train)[, -1])
Xte = as.data.frame( model.matrix(class ~ ., test)[, -1])

Ytr = y2 %>%
  mutate(Poison = ifelse(class=='p', 1, 0))


pr <- knn(Xtr,Xte,cl=(as.factor(train$class)),k=5)
```

In [128]:
```r
tb <- confusionMatrix(pr,test$class)
tb
```

```
Confusion Matrix and Statistics

          Reference
Prediction    e    p
         e 1283    0
         p    0 1155

               Accuracy : 1
                 95% CI : (0.9985, 1)
    No Information Rate : 0.5263
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.5263
         Detection Rate : 0.5263
   Detection Prevalence : 0.5263
      Balanced Accuracy : 1.0000

       'Positive' Class : e
```

In [129]:
```r
y_pred_knn_bin = 1*(pr == 'p')
PRROC_obj <- roc.curve(scores.class0 = pr, weights.class0=true_test_y,
                       curve=TRUE)
plot(PRROC_obj)
```



Observation table

In [130]:

```
new_df = data.frame(true_test_y,y_pred_knn_bin,y_pred_tree_bin,y_pred_svm_bin,
y_pred)
new_df
```

A data.frame: 2438 × 5

	true_test_y	y_pred_knn_bin	y_pred_tree_bin	y_pred_svm_bin	lambda.min
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
7	0	0	0	0	0
14	1	1	1	1	1
24	0	0	0	0	0
33	0	0	0	0	0
41	0	0	0	0	0
45	0	0	0	0	0
46	0	0	0	0	0
48	0	0	0	0	0
53	0	0	0	0	0
57	0	0	0	0	0
61	0	0	0	0	0
72	0	0	0	0	0
80	0	0	0	0	0
82	1	1	1	1	1
88	0	0	0	0	0
90	0	0	0	0	0
94	0	0	0	0	0
100	0	0	0	0	0
105	0	0	0	0	0
107	0	0	0	0	0
109	0	0	0	0	0
112	0	0	0	0	0
114	0	0	0	0	0
121	1	1	1	1	1
122	0	0	0	0	0
125	0	0	0	0	0
135	0	0	0	0	0
137	0	0	0	0	0
139	1	1	1	1	1
142	0	0	0	0	0
⋮	⋮	⋮	⋮	⋮	⋮
8021	1	1	1	1	1
8023	1	1	1	1	1
8025	1	1	1	1	1
8030	0	0	0	0	0
8031	0	0	0	0	0
8032	1	1	1	1	1
8039	0	0	0	0	0
8043	0	0	0	0	0
8046	1	1	1	1	1
8047	0	0	0	0	0
8050	1	1	1	1	1
8054	1	1	1	1	1
8057	0	0	0	0	0
8058	0	0	0	0	0

	true_test_y	y_pred_knn_bin	y_pred_tree_bin	y_pred_svm_bin	lambda.min
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
8071	1	1	1	1	1
8074	0	0	0	0	0
8075	0	0	0	0	0
8078	0	0	0	0	0
8083	1	1	1	1	1
8084	1	1	1	1	1
8095	0	0	0	0	0
8098	1	1	1	1	1
8101	0	0	0	0	0
8104	0	0	0	0	0
8107	0	0	0	0	0
8108	0	0	0	0	0
8112	0	0	0	0	0
8113	0	0	0	0	0
8117	1	1	1	1	1
8122	0	0	0	0	0

	Logistic Regression (Lasso)	K Nearest Neighbor	S Vector Machine	Decision Tree
Class	Poisionous-1/Edible-0	Poisionous-1/Edible-0	Poisionous-1/Edible-0	Poisionous-1/Edible-0
AUC	1.000	1.000	1.000	0.99307
Accuracy	100%	100%	100%	99.34%
Specificity	1.000	1.000	1.000	0.9861
Sensitivity	1.000	1.000	1.000	1.000

In [131]: