Practical 2: Model Evaluation

Laura Rodriguez Navas

January 2020

Section 1

Load the data into R. Name the columns to better identify the board, as visited from left to right and from top to down.

```
data <- read.table("tic-tac-toe.data.txt", header=FALSE, sep=",")
names(data) <- c("top-left-square",
                 "top-middle-square",
                 "top-right-square",
                 "middle-left-square",
                 "middle-middle-square",
                 "middle-right-square",
                 "bottom-left-square",
                 "bottom-middle-square",
                 "bottom-right-square",
                 "Class")
str(data)
```

```
## 'data.frame':    958 obs. of  10 variables:
##  $ top-left-square     : Factor w/ 3 levels "b","o","x": 3 3 3 3 3 3 3 3 3 3 ...
##  $ top-middle-square   : Factor w/ 3 levels "b","o","x": 3 3 3 3 3 3 3 3 3 3 ...
##  $ top-right-square    : Factor w/ 3 levels "b","o","x": 3 3 3 3 3 3 3 3 3 3 ...
##  $ middle-left-square  : Factor w/ 3 levels "b","o","x": 3 3 3 3 3 3 3 3 3 3 ...
##  $ middle-middle-square: Factor w/ 3 levels "b","o","x": 2 2 2 2 2 2 2 2 2 1 ...
##  $ middle-right-square : Factor w/ 3 levels "b","o","x": 2 2 2 2 2 2 1 1 1 2 ...
##  $ bottom-left-square  : Factor w/ 3 levels "b","o","x": 3 2 2 2 1 1 2 2 1 2 ...
##  $ bottom-middle-square: Factor w/ 3 levels "b","o","x": 2 3 2 1 2 1 2 1 2 2 ...
##  $ bottom-right-square : Factor w/ 3 levels "b","o","x": 2 2 3 1 1 2 1 2 2 1 ...
##  $ Class               : Factor w/ 2 levels "negative","positive": 2 2 2 2 2 2 2 2 2 2 ...
```

Check for missing values.

```
any(is.na(data))
```

```
## [1] FALSE
```

Section 2

Read the "data splitting" section at the web page of caret. Then split the data into 70% training and 30% test by keeping the original proportion of classes.

```
set.seed(825)
inTraining <- createDataPartition(data$Class, p=.7, list=FALSE)
data_training <- data[ inTraining,]
```

```r
data_testing  <- data[-inTraining,]
str(data_training)
```

```
## 'data.frame':    672 obs. of  10 variables:
##  $ top.left.square     : Factor w/ 3 levels "b","o","x": 3 3 3 3 3 3 3 3 3 3 ...
##  $ top.middle.square   : Factor w/ 3 levels "b","o","x": 3 3 3 3 3 3 3 3 3 3 ...
##  $ top.right.square    : Factor w/ 3 levels "b","o","x": 3 3 3 3 3 3 3 3 3 3 ...
##  $ middle.left.square  : Factor w/ 3 levels "b","o","x": 3 3 3 3 3 3 3 3 3 3 ...
##  $ middle.middle.square: Factor w/ 3 levels "b","o","x": 2 2 2 2 2 2 2 2 1 1 ...
##  $ middle.right.square : Factor w/ 3 levels "b","o","x": 2 2 2 2 2 2 1 1 2 2 ...
##  $ bottom.left.square  : Factor w/ 3 levels "b","o","x": 3 2 2 2 1 1 2 2 2 1 ...
##  $ bottom.middle.square: Factor w/ 3 levels "b","o","x": 2 3 2 1 2 1 2 1 2 2 ...
##  $ bottom.right.square : Factor w/ 3 levels "b","o","x": 2 2 3 1 1 2 1 2 1 2 ...
##  $ Class               : Factor w/ 2 levels "negative","positive": 2 2 2 2 2 2 2 2 2 2 ...
```

```r
str(data_testing)
```

```
## 'data.frame':    286 obs. of  10 variables:
##  $ top.left.square     : Factor w/ 3 levels "b","o","x": 3 3 3 3 3 3 3 3 3 3 ...
##  $ top.middle.square   : Factor w/ 3 levels "b","o","x": 3 3 3 3 3 3 3 3 3 3 ...
##  $ top.right.square    : Factor w/ 3 levels "b","o","x": 3 3 3 3 3 3 3 3 3 3 ...
##  $ middle.left.square  : Factor w/ 3 levels "b","o","x": 3 3 2 2 2 2 2 2 2 2 ...
##  $ middle.middle.square: Factor w/ 3 levels "b","o","x": 2 1 3 3 3 3 2 1 1 1 ...
##  $ middle.right.square : Factor w/ 3 levels "b","o","x": 1 2 2 2 2 1 1 3 3 2 ...
##  $ bottom.left.square  : Factor w/ 3 levels "b","o","x": 1 2 3 2 1 2 2 2 1 3 ...
##  $ bottom.middle.square: Factor w/ 3 levels "b","o","x": 2 1 2 1 2 1 1 2 2 2 ...
##  $ bottom.right.square : Factor w/ 3 levels "b","o","x": 2 2 2 1 1 2 3 1 2 1 ...
##  $ Class               : Factor w/ 2 levels "negative","positive": 2 2 2 2 2 2 2 2 2 2 ...
```

Section 3

Specifiy the type of resampling.

```r
fitControl <- trainControl(method="repeatedcv",
                           number=10,
                           repeats=1,
                           classProbs=TRUE)
```

Apply the models: Naive Bayes, Decision Tree, Neural Networks, Nearest Neighbour and SVM (linear kernel) to the data training dataset using the same seed.

1. Model Naive Bayes

```r
set.seed(825)
nb <- train(Class ~ .,
            data=data_training,
            method="naive_bayes",
            trControl=fitControl)
nb
```

```
## Naive Bayes
##
## 672 samples
##   9 predictor
##   2 classes: 'negative', 'positive'
##
## No pre-processing
```

```
## Resampling: Cross-Validated (10 fold, repeated 1 times)
## Summary of sample sizes: 606, 605, 604, 605, 605, 605, ...
## Resampling results across tuning parameters:
##
##   usekernel  Accuracy   Kappa
##   FALSE      0.6756219  0.2666418
##    TRUE      0.6845565  0.1130575
##
## Tuning parameter 'laplace' was held constant at a value of 0
## Tuning
##  parameter 'adjust' 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 laplace = 0, usekernel = TRUE
##  and adjust = 1.
```

2. Model Decision Tree

```r
set.seed(825)
dt <- train(Class ~ .,
            data=data_training,
            method="rpart2",
            trControl=fitControl)
dt
```

```
## CART
##
## 672 samples
##   9 predictor
##   2 classes: 'negative', 'positive'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 1 times)
## Summary of sample sizes: 606, 605, 604, 605, 605, 605, ...
## Resampling results across tuning parameters:
##
##   maxdepth  Accuracy   Kappa
##    1        0.6889703  0.3190082
##    5        0.7530403  0.3667066
##   10        0.9107511  0.7973708
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was maxdepth = 10.
```

3. Model Neural Network

```r
set.seed(825)
nn <- train(Class ~ .,
            data=data_training,
            method="nnet",
            trControl=fitControl)
```

```r
nn
```

```
## Neural Network
##
## 672 samples
##   9 predictor
```

```
##   2 classes: 'negative', 'positive'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 1 times)
## Summary of sample sizes: 606, 605, 604, 605, 605, 605, ...
## Resampling results across tuning parameters:
##
##   size  decay  Accuracy   Kappa
##   1     0e+00  0.9732221  0.9402993
##   1     1e-04  0.9717296  0.9369543
##   1     1e-01  0.9776778  0.9498311
##   3     0e+00  0.8646593  0.6307553
##   3     1e-04  0.9612592  0.9140460
##   3     1e-01  0.9776997  0.9499157
##   5     0e+00  0.9598771  0.9103137
##   5     1e-04  0.9582954  0.9085208
##   5     1e-01  0.9761846  0.9466124
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were size = 3 and decay = 0.1.
```

4. Model Nearest Neighbour

```r
set.seed(825)
knn <- train(Class ~ .,
             data=data_training,
             method="knn",
             trControl=fitControl)
knn
```

```
## k-Nearest Neighbors
##
## 672 samples
##   9 predictor
##   2 classes: 'negative', 'positive'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 1 times)
## Summary of sample sizes: 606, 605, 604, 605, 605, 605, ...
## Resampling results across tuning parameters:
##
##   k  Accuracy   Kappa
##   5  0.9420751  0.8667396
##   7  0.8066433  0.5374263
##   9  0.7709534  0.4451248
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 5.
```

5. Model SVM (linear kernel)

```r
set.seed(825)
svm <- train(Class ~ .,
             data=data_training,
             method="svmLinear",
             trControl=fitControl)
```

```r
svm
```

```
## Support Vector Machines with Linear Kernel
##
## 672 samples
##   9 predictor
##   2 classes: 'negative', 'positive'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 1 times)
## Summary of sample sizes: 606, 605, 604, 605, 605, 605, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9806629  0.9563793
##
## Tuning parameter 'C' was held constant at a value of 1
```

Collect the results for all the models.

```r
resamps <- resamples(list("Naive Bayes"=nb,
                          "Decision Tree"=dt,
                          "Neural Network"=nn,
                          "Nearest Neighbour"=knn,
                          "SVM (linear kernel)"=svm))
summary(resamps)
```

```
##
## Call:
## summary.resamples(object = resamps)
##
## Models: Naive Bayes, Decision Tree, Neural Network, Nearest Neighbour, SVM (linear kernel)
## Number of resamples: 10
##
## Accuracy
##                          Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## Naive Bayes         0.6567164 0.6642340 0.6865672 0.6845565 0.6943691 0.7205882
## Decision Tree       0.8507463 0.9000668 0.9104478 0.9107511 0.9253731 0.9701493
## Neural Network      0.9552239 0.9702590 0.9850746 0.9776997 0.9852392 1.0000000
## Nearest Neighbour   0.8955224 0.9188982 0.9402985 0.9420751 0.9700362 0.9850746
## SVM (linear kernel) 0.9552239 0.9702590 0.9850746 0.9806629 0.9852392 1.0000000
##                     NA's
## Naive Bayes            0
## Decision Tree          0
## Neural Network         0
## Nearest Neighbour      0
## SVM (linear kernel)    0
##
## Kappa
##                          Min.    1st Qu.    Median      Mean   3rd Qu.
## Naive Bayes         0.0000000 0.05403608 0.1111813 0.1130575 0.1504189
## Decision Tree       0.6469968 0.77864355 0.7984862 0.7973708 0.8318554
## Neural Network      0.8975013 0.93288438 0.9665502 0.9499157 0.9672590
## Nearest Neighbour   0.7556019 0.81184946 0.8647830 0.8667396 0.9330473
## SVM (linear kernel) 0.8975013 0.93288438 0.9665502 0.9563793 0.9672590
```

```
##                          Max. NA's
## Naive Bayes         0.2540416    0
## Decision Tree       0.9337945    0
## Neural Network      1.0000000    0
## Nearest Neighbour   0.9665502    0
## SVM (linear kernel) 1.0000000    0
```

Complete the following table with the final values of accuracy and kappa for the training data:

	Accuracy	Kappa
Naive Bayes	0.6845565	0.1130575
Decision Tree	0.9107511	0.7973708
Neural Network	0.9776997	0.9499157
Nearest Network	0.9420751	0.8667396
SVM (linear tree)	0.9806629	0.9563793

Section 4

Apply the models: Naive Bayes, Decision Tree, Neural Networks, Nearest Neighbour and SVM (linear kernel) to the data testing dataset. Print the confusion matrix of each model.

1. Model Naive Bayes

```r
nbPredict <- predict(nb, newdata=data_testing)
confusionMatrix(nbPredict, data_testing$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction negative positive
##   negative        7        0
##   positive       92      187
##
##                Accuracy : 0.6783
##                  95% CI : (0.6208, 0.7321)
##     No Information Rate : 0.6538
##     P-Value [Acc > NIR] : 0.2102
##
##                   Kappa : 0.0905
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.07071
##             Specificity : 1.00000
##          Pos Pred Value : 1.00000
##          Neg Pred Value : 0.67025
##              Prevalence : 0.34615
##          Detection Rate : 0.02448
##    Detection Prevalence : 0.02448
##       Balanced Accuracy : 0.53535
##
##        'Positive' Class : negative
##
```

2. Model Decision Tree

```r
dtPredict <- predict(dt, newdata=data_testing)
confusionMatrix(dtPredict, data_testing$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction negative positive
##   negative       92       14
##   positive        7      173
##
##                Accuracy : 0.9266
##                  95% CI : (0.8899, 0.954)
##     No Information Rate : 0.6538
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8404
##
##  Mcnemar's Test P-Value : 0.1904
##
##             Sensitivity : 0.9293
##             Specificity : 0.9251
##          Pos Pred Value : 0.8679
##          Neg Pred Value : 0.9611
##              Prevalence : 0.3462
##          Detection Rate : 0.3217
##    Detection Prevalence : 0.3706
##       Balanced Accuracy : 0.9272
##
##        'Positive' Class : negative
##
```

3. Model Neural Network

```r
nnPredict <- predict(nn, newdata=data_testing)
confusionMatrix(nnPredict, data_testing$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction negative positive
##   negative       96        0
##   positive        3      187
##
##                Accuracy : 0.9895
##                  95% CI : (0.9697, 0.9978)
##     No Information Rate : 0.6538
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9767
##
##  Mcnemar's Test P-Value : 0.2482
##
##             Sensitivity : 0.9697
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
```

```
##          Neg Pred Value : 0.9842
##              Prevalence : 0.3462
##          Detection Rate : 0.3357
##    Detection Prevalence : 0.3357
##       Balanced Accuracy : 0.9848
##
##        'Positive' Class : negative
##
```

4. Model Nearest Neighbour

```
knnPredict <- predict(knn, newdata=data_testing)
confusionMatrix(knnPredict, data_testing$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction negative positive
##   negative       92        0
##   positive        7      187
##
##                Accuracy : 0.9755
##                  95% CI : (0.9502, 0.9901)
##     No Information Rate : 0.6538
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.945
##
##  Mcnemar's Test P-Value : 0.02334
##
##             Sensitivity : 0.9293
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9639
##              Prevalence : 0.3462
##          Detection Rate : 0.3217
##    Detection Prevalence : 0.3217
##       Balanced Accuracy : 0.9646
##
##        'Positive' Class : negative
##
```

5. Model SVM (linear kernel)

```
svmPredict <- predict(svm, newdata=data_testing)
confusionMatrix(svmPredict, data_testing$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction negative positive
##   negative       96        0
##   positive        3      187
##
##                Accuracy : 0.9895
##                  95% CI : (0.9697, 0.9978)
```

```
##     No Information Rate : 0.6538
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9767
##
##  Mcnemar's Test P-Value : 0.2482
##
##             Sensitivity : 0.9697
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9842
##              Prevalence : 0.3462
##          Detection Rate : 0.3357
##    Detection Prevalence : 0.3357
##       Balanced Accuracy : 0.9848
##
##        'Positive' Class : negative
##
```

Calculate the AUC value for all the models.

1. Model Naive Bayes

```r
auc(roc(nbPredict, data_testing$Class))
```

```
## [1] 0.5353535
```

2. Model Decison Tree

```r
auc(roc(dtPredict, data_testing$Class))
```

```
## [1] 0.9272133
```

3. Model Neural Network

```r
auc(roc(nnPredict, data_testing$Class))
```

```
## [1] 0.9848485
```

4. Model Nearest Network

```r
auc(roc(knnPredict, data_testing$Class))
```

```
## [1] 0.9646465
```

5. Model SVM (linear kernel)

```r
auc(roc(svmPredict, data_testing$Class))
```

```
## [1] 0.9848485
```

Complete the following table with the final values of accuracy, kappa and AUC for the testing data.

	Accuracy	Kappa	AUC
Naive Bayes	0.6783	0.0905	0.5353535
Decision Tree	0.9266	0.8404	0.9272133
Neural Network	0.9895	0.9767	0.9848485
Nearest Network	0.9755	0.945	0.9646465
SVM (linear tree)	0.9895	0.9767	0.9848485

Section 5

Plot the ROC curves of the models.

a) Calculate again the predictions on the test set but now setting the type parameter of the predict function to "prob".

1. Model Naive Bayes

```
nbPredictProb <- predict(nb, newdata=data_testing, type = "prob")
head(nbPredictProb)
```

```
##      negative  positive
## 1 0.105410725 0.8945893
## 2 0.048803363 0.9511966
## 3 0.002235508 0.9977645
## 4 0.004361079 0.9956389
## 5 0.001757359 0.9982426
## 6 0.012029302 0.9879707
```

2. Model Decision Tree

```
dtPredictProb <- predict(dt, newdata=data_testing, type = "prob")
head(dtPredictProb)
```

```
##    negative positive
## 9      0.00     1.00
## 11     0.16     0.84
## 13     0.00     1.00
## 16     0.00     1.00
## 17     0.00     1.00
## 20     0.16     0.84
```

3. Model Neural Network

```
nnPredictProb <- predict(nn, newdata=data_testing, type = "prob")
head(nnPredictProb)
```

```
##       negative  positive
## 9  0.014915208 0.9850848
## 11 0.022068633 0.9779314
## 13 0.018293877 0.9817061
## 16 0.009860493 0.9901395
## 17 0.005332153 0.9946678
## 20 0.015123710 0.9848763
```

4. Model Nearest Neighbour

```
knnPredictProb <- predict(knn, newdata=data_testing, type = "prob")
head(knnPredictProb)
```

```
##   negative positive
## 1        0        1
## 2        0        1
## 3        0        1
## 4        0        1
## 5        0        1
## 6        0        1
```

5. Model SVM (linear kernel)

```r
svmPredictProb <- predict(svm, newdata=data_testing, type = "prob")
head(svmPredictProb)
```

```
##     negative  positive
## 1 0.03087548 0.9691245
## 2 0.03086058 0.9691394
## 3 0.03082860 0.9691714
## 4 0.03084267 0.9691573
## 5 0.03084003 0.9691600
## 6 0.03085089 0.9691491
```

b) Construct a "prediction" object for each classifier using the vector of estimated probabilities for the positive class as the first parameter, and the vector of actual class labels as the second parameter.

1. Model Naive Bayes

```r
nbPred <- prediction(nbPredictProb$positive, data_testing$Class)
```

2. Model Decision Tree

```r
dtPred <- prediction(dtPredictProb$positive, data_testing$Class)
```

3. Model Neural Network

```r
nnPred <- prediction(nnPredictProb$positive, data_testing$Class)
```

4. Model Nearest Neighbour

```r
knnPred <- prediction(knnPredictProb$positive, data_testing$Class)
```

5. Model SVM (linear kernel)

```r
svmPred <- prediction(svmPredictProb$positive, data_testing$Class)
```

c) Calculate the measures we want to plot on the y-axis (TPR) and on the x-axis (FPR) by using the performance function.

1. Model Naive Bayes

```r
nbPerf <- performance(nbPred, "tpr", "fpr")
```

2. Model Decision Tree

```r
dtPerf <- performance(dtPred, "tpr", "fpr")
```

3. Model Neural Network

```r
nnPerf <- performance(nnPred, "tpr", "fpr")
```

4. Model Nearest Neighbour

```r
knnPerf <- performance(knnPred, "tpr", "fpr")
```

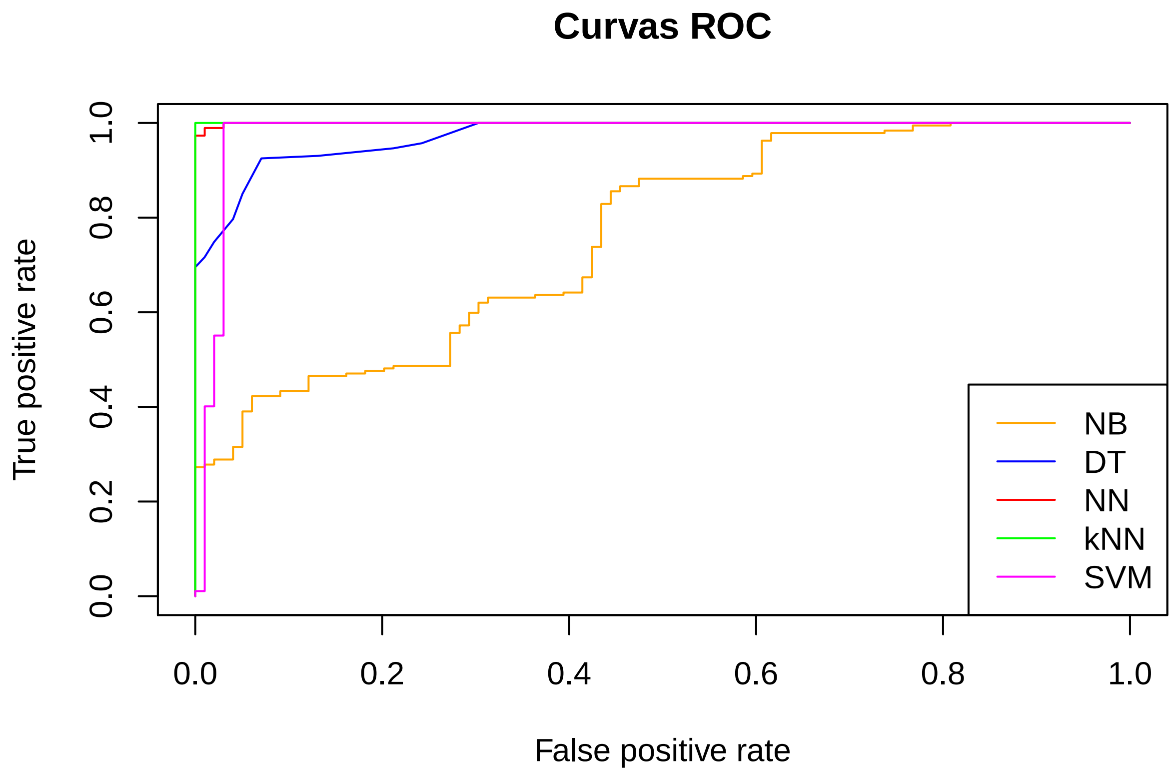
5. Model SVM (linear kernel)

```r
svmPerf <- performance(svmPred, "tpr", "fpr")
```

d) Draw all the curves in the same plot.

```r
plot(nbPerf, col="orange", add=FALSE, main="Curvas ROC")
plot(dtPerf, col="blue", add=TRUE, main="Curvas ROC")
plot(nnPerf, col="red", add=TRUE, main="Curvas ROC")
plot(knnPerf, col="green", add=TRUE, main="Curvas ROC")
```

```r
plot(svmPerf, col="magenta", add=TRUE, main="Curvas ROC")
legend("bottomright",
       legend = c("NB","DT", "NN", "kNN", "SVM"),
       col = c("orange", "blue", "red", "green", "magenta"),
       lty = 1, lwd = 1)
```



Q1. ¿Si el modelo A tiene mayor Accuracy que B, siempre tendrá mayor Kappa que B? Justifica tu respuesta.

Sí.

Q2. ¿Vemos eso en tus resultados?

Sí

Q3. ¿Te cambian los resultados cuando cambias la semilla?

Sí.

Q4. ¿Es recomendable quedarse con los resultados mejores después de cambiar las semillas varias veces? Justifica la respuesta.

Q5. ¿Qué modelos puedes descartar porque van a ser siempre subóptimos (asumiendo una buena evaluación)?

Q6. ¿Por qué puedes descartar esos modelos?