Performance evaluation

Ruxandra Stoean

Further bibliography

- N. Japkowicz, M. Shah, Evaluating Learning Algorithms, Cambridge University Press, 2011
- T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning, Springer Series in Statistics, 2001
- C. Huyen, Designing Machine Learning Systems: An Iterative Process for Production-Ready Applications, O'Reilly, 2022

Directions

- Performance measures
  - Classification
  - Regression
- Error estimation
- Statistical significance

Accuracy / Error rate

- The two complementary measures give an insight over the general performance.

- Accuracy is computed as the ratio between the amount of data labelled correctly and the total number of examples
  - The error rate is the ratio between the amount of samples labelled incorrectly and the total number of samples

- Disadvantages in the cases when:
  - Data with class imbalance
  - Different misclassification costs for each class

Confusion matrix

- For k classes, a matrix of k × k
  - The element at position (i, j) gives the amount of data of class i that the classifier labels as j
- Generically, for two classes (positive and negative labels)
- N = TN + FP, P = FN + TP

	Predicted negative	**Predicted positive**
Real negative	True negative (TN)	False positive (FP)
Real positive	False negative (FN)	True positive (TP)

Performance measures for a single class of interest 1/2

- Class of interest denoted by the positive.
- True-positive rate (TPR) gives the proportion of data of class *i* that are also labelled as *i* by the classifier.
- $TPR = \frac{TP}{TP+FN}$

- False-positive rate (FPR) – proportion of data that do not belong to class *i* but are labelled with that class.
- $FPR = \frac{FP}{FP+TN}$

Performance measures for a single class of interest 2/2

- In the binary case, the two measures can be also defined for the opposite, negative class.
- True-negative rate $TNR = \frac{TN}{TN+FP}$
- False-negative rate $FNR = \frac{FN}{FN+TP}$

- TPR is also called sensitivity (or recall).
- TNR is also named specificity.

Likelihood ratio

- Combines sensitivity and specificity to see the extent to which the classifier is efficient in prediction over the two classes.

- $\mathrm{LR}_{+} = \frac{Sensitivity}{1 - Specificity} \rightarrow max$
- $\mathrm{LR}_{-} = \frac{1 - Sensitivity}{Specificity} \rightarrow min$

- LR_{+} computes the times it is more likely that positive objects have a positive prediction compared to negative objects.
- LR_{-} computes the times it is less likely that positive objects have a negative prediction compared to negative objects.

LR in comparison

- Two algorithms A1 and A2.
- First, $LR_+ \geq 1$
  - Contrarily, revert LR_+ with LR_-.
- If $LR_+(A1)>LR_+(A2)$ and $LR_-(A1)<LR_-(A2)$, then A1 is overall superior.
- If $LR_+(A1)<LR_+(A2)$ and $LR_-(A1)<LR_-(A2)$, then A1 is superior in the confirmation of negative examples.
- If $LR_+(A1)>LR_+(A2)$ and $LR_-(A1)>LR_-(A2)$, then A1 is superior in confirming positive examples.

Positive predictive value (PPV)

- It is also called precision.
- The proportion of data that belong to class *i* of all detected by the algorithms as being class i.

- For the binary case:
- $PPV = \frac{TP}{TP+FP}$
- and the reverse NPV = $\frac{TN}{TN+FN}$
- The value PPV = a states that a positive prediction of the corresponding classifier will be true only in a% of the cases.

F1-score

- F-score combines precision and recall in a single measure, as the weighted harmonic mean of the two.
  - F1-score (balanced F-score): when the two measures have equal importance
- Useful for class imbalanced problems
- $Precision = \frac{TP}{TP+FP}$
- $Recall = \frac{TP}{TP+FN}$
- $F1 = \frac{2}{\frac{1}{Precision}+\frac{1}{Recall}} = \frac{2\ x\ Precision\ x\ Recall}{Precision+Recall}$

ROC analysis

- Receiver operating characteristic (ROC)
- The ROC curve for a classifier is a plot where
  - FPR on the horizontal axis
  - TPR on the vertical axis.
- ROC thus studies the relationship between sensitivity si specificity for a classifier.
- The ROC space – is a square of side 1.

The ROC space 1/2

- The output of a classifier determines a point in the ROC space.
  - (0,0) denotes a classifier that labels all data as negative (TPR = FPR = 0).
  - (1,1) denotes a classifier that labels all data as positive (TPR = FPR = 1).
  - The classifiers whose output are positioned on the main diagonal (TPR = FPR) label data in a random fashion.
  - Those whose output lies above the main diagonal have a better performance than random labelling.
    - Those whose output is situated under the diagonal have a performance weaker than random labelling.

The ROC space 2/2

- For two points p1 and p2 in the ROC space, p1 is a better classifier than p2 if p1 is to the upper left side of p2.
- (1,0) (FPR = 1, TPR = 0) denotes a classifier that has only wrong predictions.
- (0,1) denotes the ideal classifier (labels all positive data correctly and does not make mistakes in labelling the negative examples).
- Classifiers with the output on the secondary diagonal have an equal performance both on the positive and the negative examples.

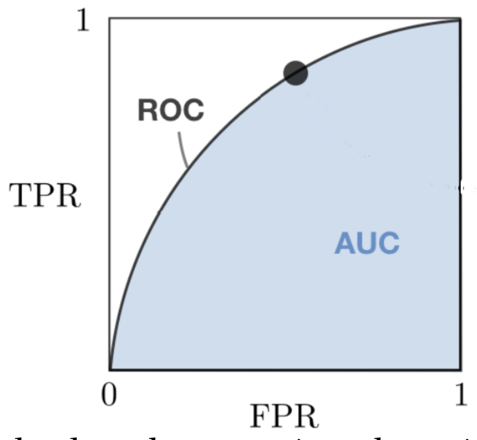
The ROC curve

- An operating point corresponds to a decision threshold under which the classifier discriminates data as being positive or negative.
- Data with a score above this threshold are labelled as positive, and those with a score under are considered negative.
- By varying the threshold between the minimum and maximum score of the data, one such TPR and FPR are obtained that can be plotted:
  - This is the ROC curve.

Area under the ROC curve (AUC)

- AUC – the measure of the performance of a classifier.

- The AUC value belongs to the interval [0,1]
  - 1 is the value for a perfect classifier.
  - The AUC value for a random classifier is 0.5.
  - For a reasonable performance, a classifier must have an AUC over 0.5.



https://stanford.edu/~shervine/teaching/cs-229

Statistical agreement measures

- Measures the weight of coincidence in the agreement between the classifier predictions and the true labels of the data.
- The most used statistics: Cohen's k (kappa)
  - Traditionally measures inter-rater reliability on qualitative ratings
  - Considers the possibility of random agreement
- $k = (p_0 - p_e)/(1 - p_e)$

Calculation

	Pos	Neg
Pos	X	Y
Neg	Z	T

- p_0 – relative agreement observed between the two
  - The number of cases when both agree on positive and on negative labels over the total number of cases
  - $(X + T)/S$, $S = X + Y + Z + T$
- p_e – hypothetical probability of random agreement
  - (The proportion of cases when the classifier labeled as positive) X (the proportion of cases when the real data was positive) + (the proportion of cases when the classifier labeled negative) X (the proportion of cases when the data was negative)
  - $((X + Y)/S) * ((X+ Z)/S) + ((Z + T)/S) * ((Y + T)/S)$

Cohen’s k

- $k = (p_0 - p_e)/(1 - p_e)$

- The k resulting value indicates:
  - $k < 0$ - "No agreement"
  - $0 < k < 0.2$ - "Slight agreement"
  - $0.2 < k < 0.4$ - "Fair agreement“
  - $0.4 < k < 0.6$ - "Moderate agreement“
  - $0.6 < k < 0.8$ - "Substantial agreement“
  - $0.8 < k < 1.0$ - "Almost perfect agreement"

Metrics for regression

- $\mathbf{y}_i$ - actual values, $\hat{\mathbf{y}}_i$ or $\mathbf{f}(\mathbf{x}_i)$- predicted values, $\bar{\mathbf{y}}$ - mean of $\mathbf{y}$
- n - number of records, p – number of attributes
- Mean squared error (MSE)
  - Root mean squared error (RMSE)

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

- Mean absolute error (MAE) – absolute value of the difference
  - Useful for data with outliers

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} | y_i - \hat{y}_i |$$

- Mean Absolute Percentage Error (MAPE)
  - Independent of the attribute scale

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n} \frac{| y_i - \hat{y}_i |}{y_i} \,.\, 100\%$$

Metrics for regression

- $\mathbf{y}_i$ - actual values, $\hat{\mathbf{y}}_i$ or $\mathbf{f}(\mathbf{x}_i)$- predicted values, $\bar{\mathbf{y}}$ - mean of $\mathbf{y}$
- n - number of records, p – number of attributes

$$R^2 = 1 - \frac{RSS}{TSS}$$

- Coefficient of determination R^2
  - RSS – residual sum of squares

$$\text{RSS} = \sum_{i=1}^{n} (y_i - f(x_i))^2$$

  - TSS – total sum of squares

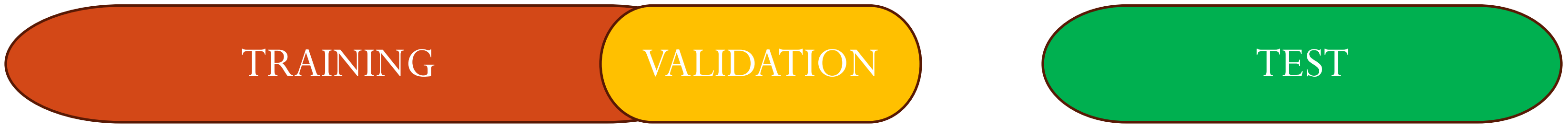
$$\text{TSS} = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

  - Adjusted R^2 – takes into account the number of variables as well

$$R^2_{adj} = 1 - \frac{(1-R^2)(n-1)}{n-p-1}$$

Error estimation 1/2

- It is used for an objective estimation of the performance of a learner.
- Data are split into three parts:
  - A training set from which the algorithm learns the associations between attribute values and outcomes.
  - A validation set to determine the error of prediction of the model (when the data collection is sufficiently large).
  - A test set to measure the generalization error.
  - For the last two sets, the data targets are considered to be unknown.



Error estimation 2/2

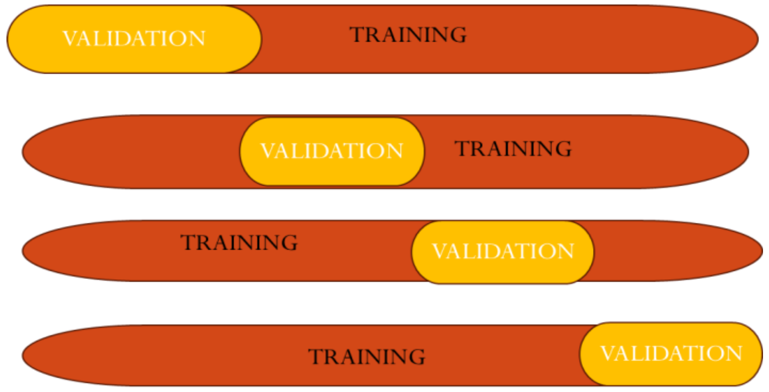
- Cross-validation estimates the prediction accuracy/error that the model will exhibit in practice.
- Training and test sets are selected a number of times following different schemes.
- The generalization ability of the learner is verified by computing the prediction accuracy/error on the test set as mean over several cross-validation runs.
- If the data set allows, the prediction accuracy/error is estimated on the validation set, before proceeding to the test set.

Random subsampling

- The simplest resampling technique.
- n runs of the learner are executed
  - Usually $n = 30$.
- In each run, the data set is split:
  - 2/3 (or 3/4) the training set
  - 1/3 (or 1/4) the test set
  - Data sampling into the two sets is done randomly without replacement.
- The final prediction accuracy/error is the mean of the corresponding values obtained in the 30 runs on the test set.

K-fold cross-validation

- Split the training data set into *k* folds
  - Train on the *k-1* folds
  - Validate on the remaining fold
  - Average prediction accuracy/error on the *k* folds.



Testing the statistical significance 1/2

- Useful (and compulsory) when:
  - The performance of a new model is compared to those of baseline algorithms.
  - The best algorithm for a given problem has to be selected.

- n ($n = 30$) runs of the algorithms through cross-validation are performed.

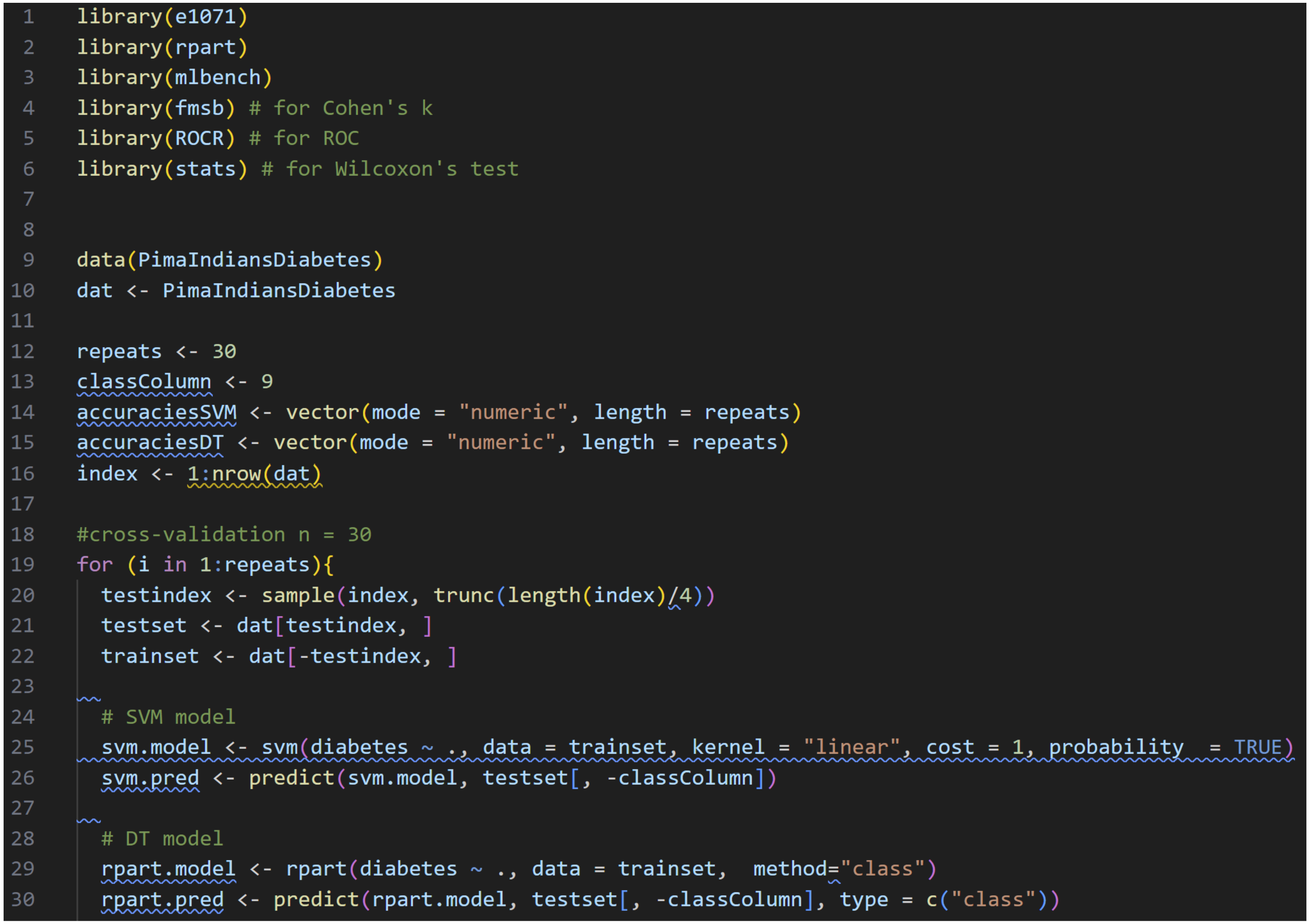
Testing the statistical significance 2/2

- The null hypothesis H_0 is presumed – the tested algorithms have a similar performance.
- The null hypothesis H_0 is rejected if the p-value returned by the test is lower or equal to the significance level of 0.05.
- The safe option to test the statistical significance is the employment of a non-parametric test that does not make assumptions on the distribution of the performance values.

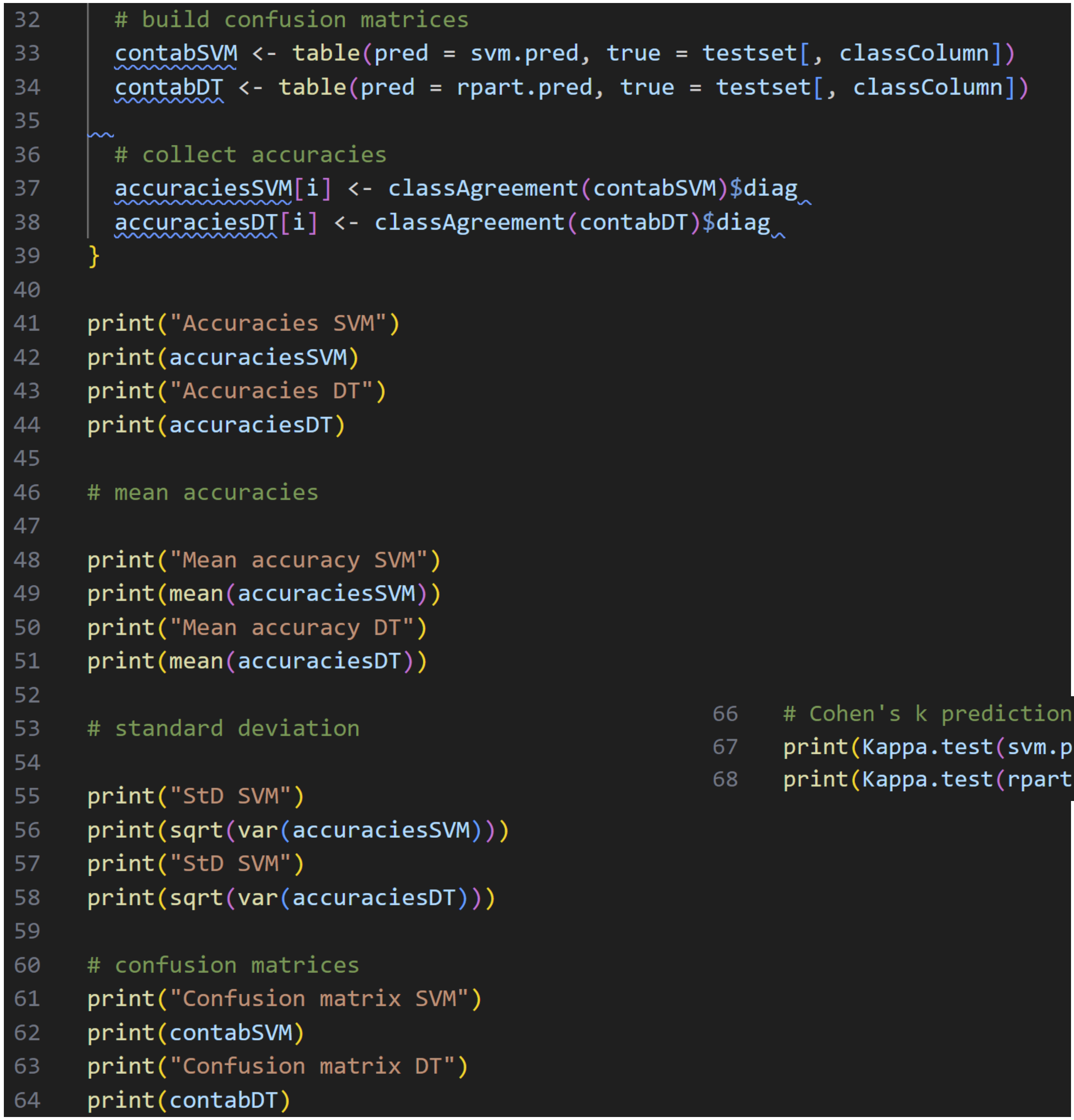
Comparing learners

- Wilcoxon signed-rank test is a non-parametric method to compare 2 learners on 1 data set
  - An obtained p-value ≤ 0.05 implies that H_0 is rejected
- Friedman test is also a non-parametric means to compare M machine learning algorithms over N datasets
  - Each learner is ranked per data set
  - The average rank is computed over all data sets
  - The Friedman statistics is calculated and interpreted
  - If the null hypothesis is rejected, a post-hoc Nemeyi test should be applied to find those that are statistically different from the others

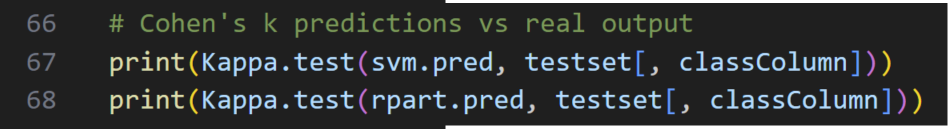
Model evaluation in R



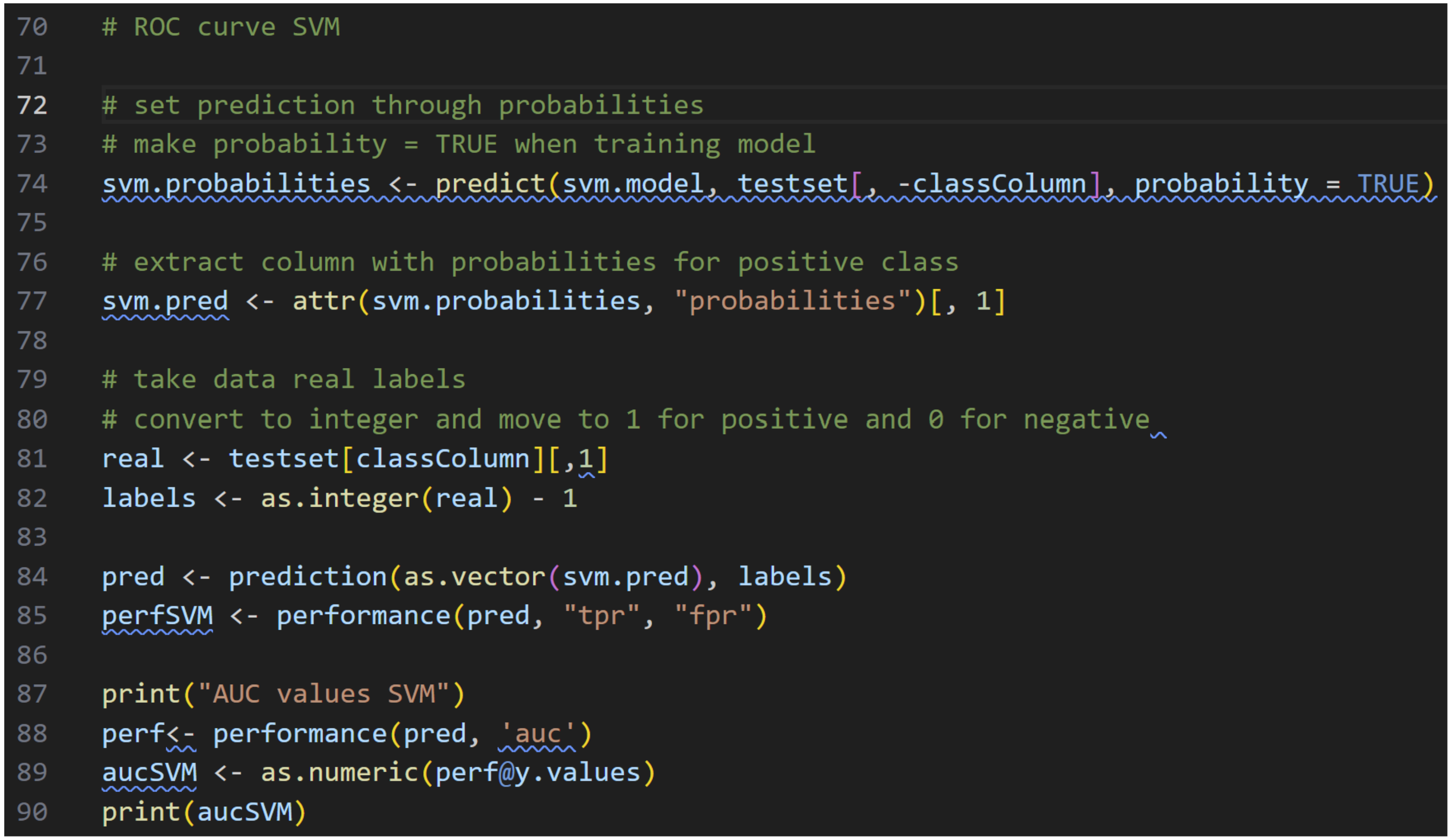
```r
library(e1071)
library(rpart)
library(mlbench)
library(fmsb) # for Cohen's k
library(ROCR) # for ROC
library(stats) # for Wilcoxon's test


data(PimaIndiansDiabetes)
dat <- PimaIndiansDiabetes

repeats <- 30
classColumn <- 9
accuraciesSVM <- vector(mode = "numeric", length = repeats)
accuraciesDT <- vector(mode = "numeric", length = repeats)
index <- 1:nrow(dat)

#cross-validation n = 30
for (i in 1:repeats){
  testindex <- sample(index, trunc(length(index)/4))
  testset <- dat[testindex, ]
  trainset <- dat[-testindex, ]

  # SVM model
  svm.model <- svm(diabetes ~ ., data = trainset, kernel = "linear", cost = 1, probability  = TRUE)
  svm.pred <- predict(svm.model, testset[, -classColumn])

  # DT model
  rpart.model <- rpart(diabetes ~ ., data = trainset,  method="class")
  rpart.pred <- predict(rpart.model, testset[, -classColumn], type = c("class"))
```



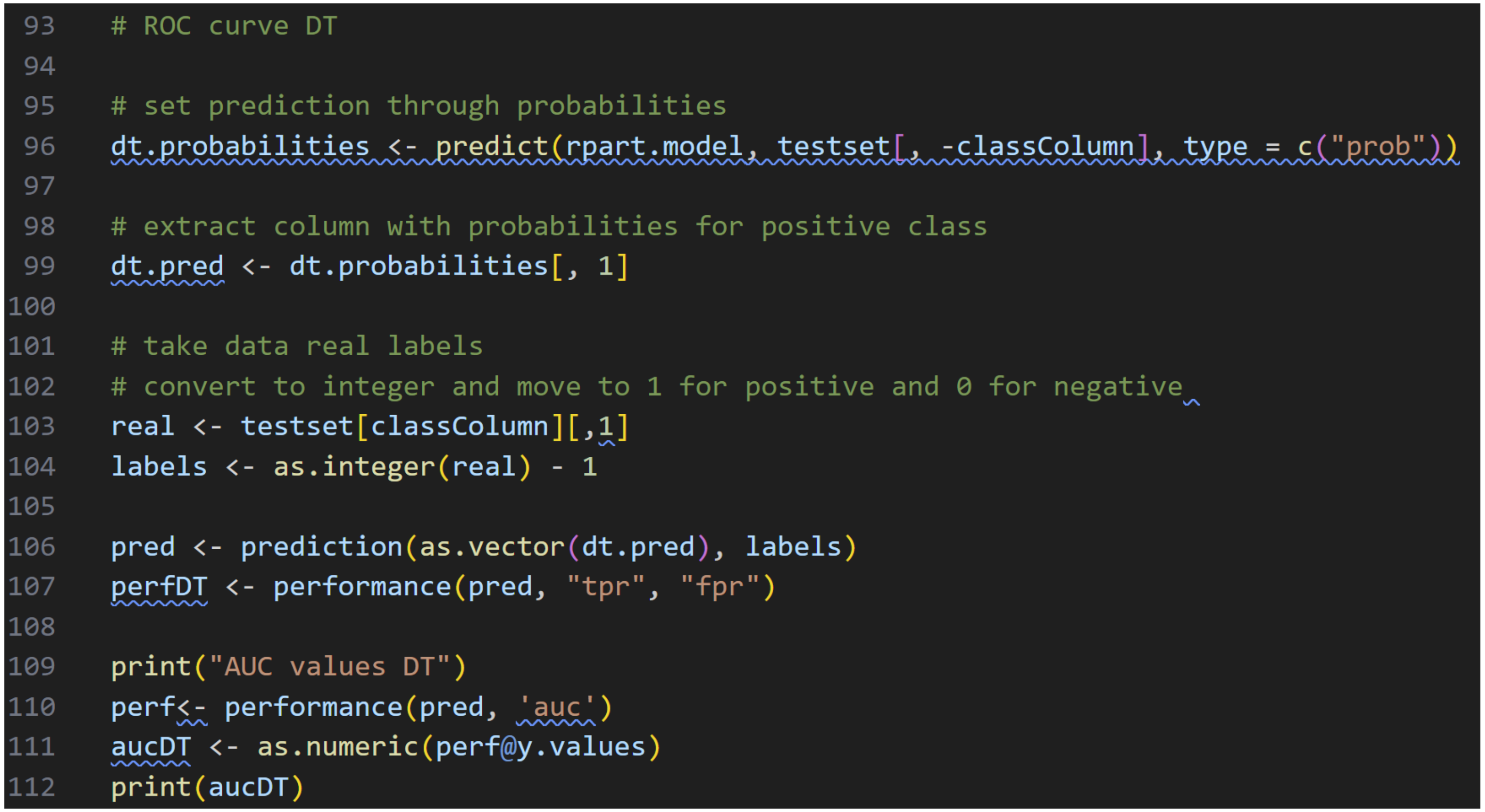
```r
  # build confusion matrices
  contabSVM <- table(pred = svm.pred, true = testset[, classColumn])
  contabDT <- table(pred = rpart.pred, true = testset[, classColumn])

  # collect accuracies
  accuraciesSVM[i] <- classAgreement(contabSVM)$diag
  accuraciesDT[i] <- classAgreement(contabDT)$diag
}

print("Accuracies SVM")
print(accuraciesSVM)
print("Accuracies DT")
print(accuraciesDT)

# mean accuracies

print("Mean accuracy SVM")
print(mean(accuraciesSVM))
print("Mean accuracy DT")
print(mean(accuraciesDT))

# standard deviation

print("StD SVM")
print(sqrt(var(accuraciesSVM)))
print("StD SVM")
print(sqrt(var(accuraciesDT)))

# confusion matrices
print("Confusion matrix SVM")
print(contabSVM)
print("Confusion matrix DT")
print(contabDT)
```



```r
# Cohen's k predictions vs real output
print(Kappa.test(svm.pred, testset[, classColumn]))
print(Kappa.test(rpart.pred, testset[, classColumn]))
```

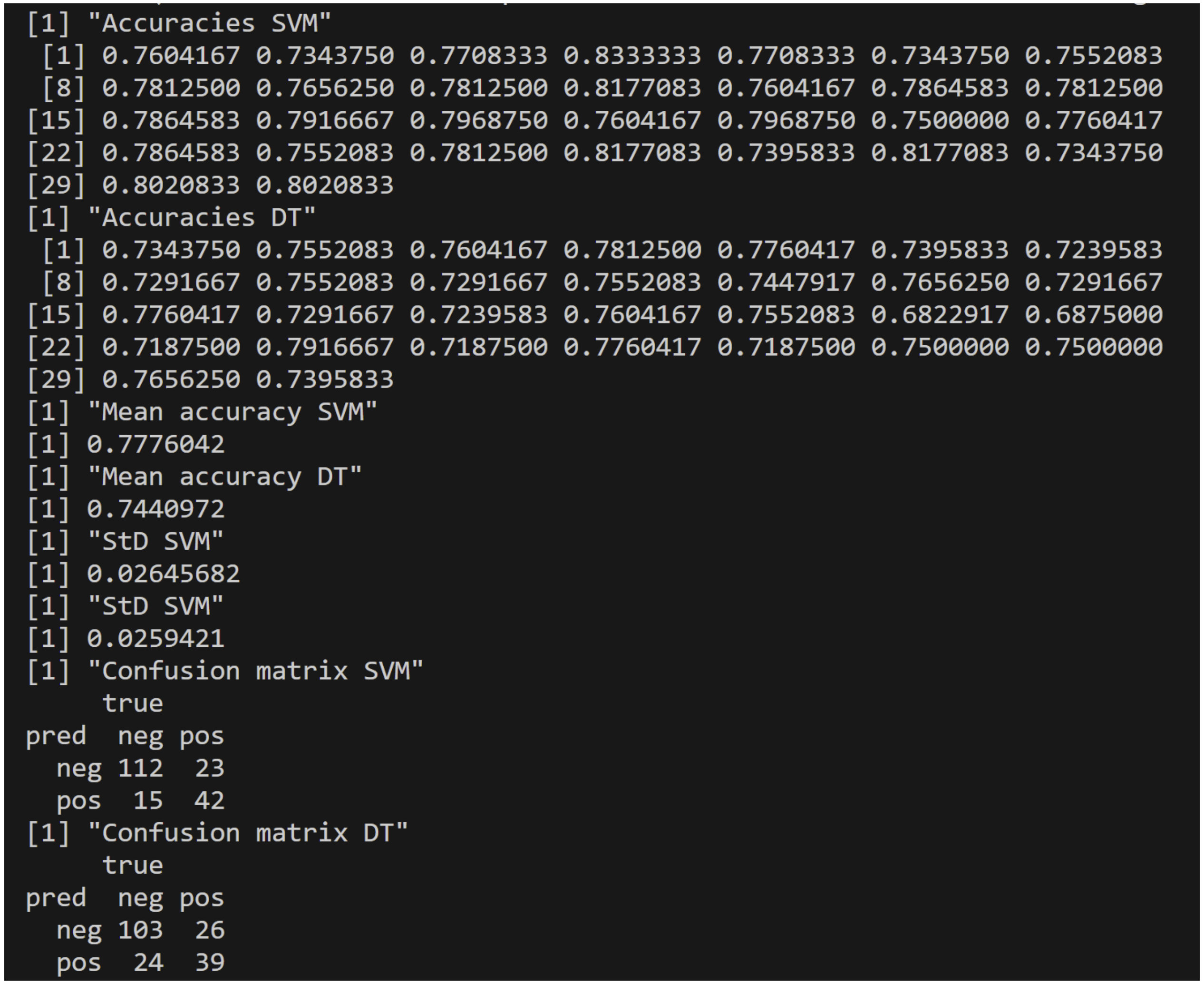


```
# ROC curve SVM

# set prediction through probabilities
# make probability = TRUE when training model
svm.probabilities <- predict(svm.model, testset[, -classColumn], probability = TRUE)

# extract column with probabilities for positive class
svm.pred <- attr(svm.probabilities, "probabilities")[, 1]

# take data real labels
# convert to integer and move to 1 for positive and 0 for negative
real <- testset[classColumn][,1]
labels <- as.integer(real) - 1

pred <- prediction(as.vector(svm.pred), labels)
perfSVM <- performance(pred, "tpr", "fpr")

print("AUC values SVM")
perf<- performance(pred, 'auc')
aucSVM <- as.numeric(perf@y.values)
print(aucSVM)
```



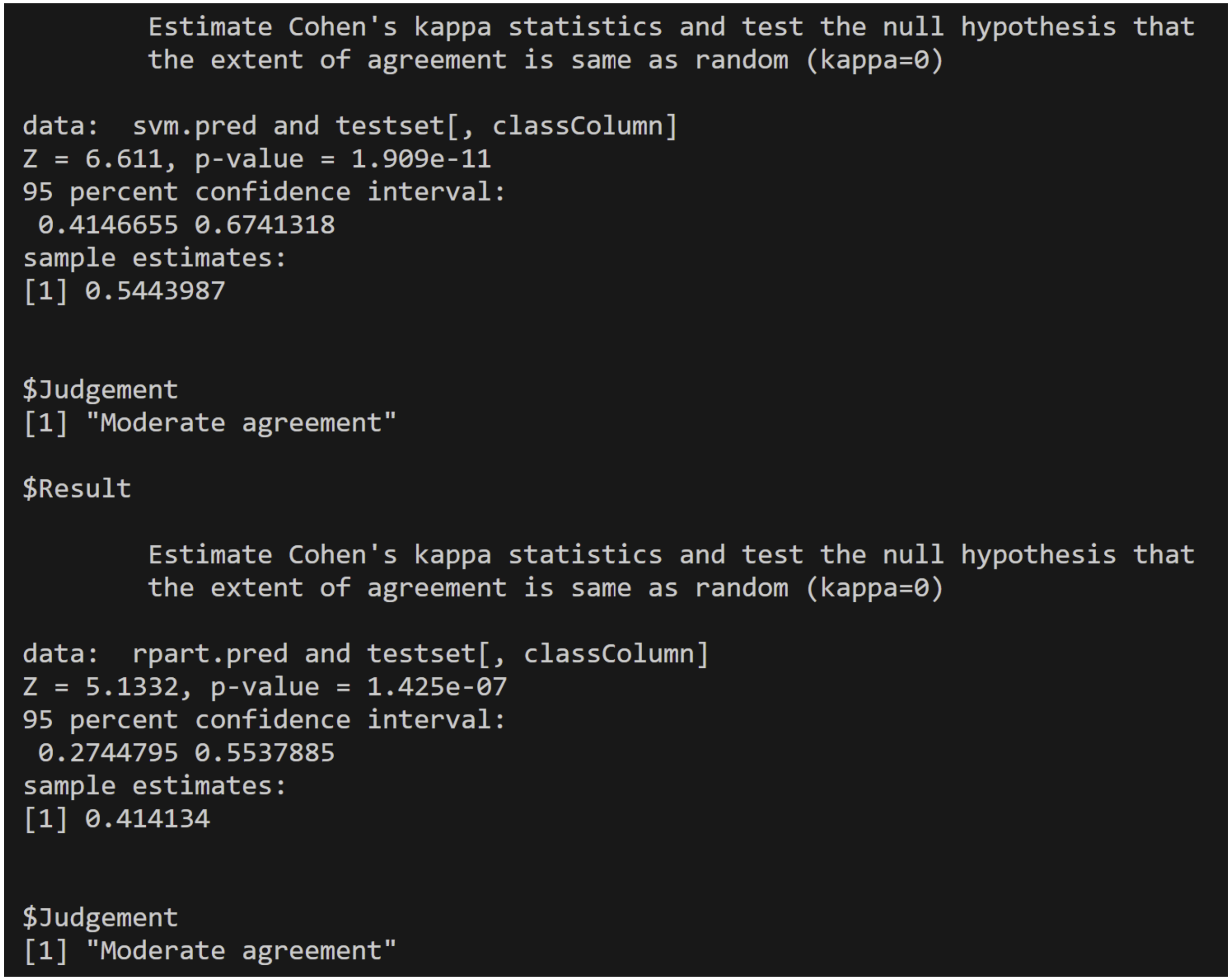
```r
# ROC curve DT

# set prediction through probabilities
dt.probabilities <- predict(rpart.model, testset[, -classColumn], type = c("prob"))

# extract column with probabilities for positive class
dt.pred <- dt.probabilities[, 1]

# take data real labels
# convert to integer and move to 1 for positive and 0 for negative
real <- testset[classColumn][,1]
labels <- as.integer(real) - 1

pred <- prediction(as.vector(dt.pred), labels)
perfDT <- performance(pred, "tpr", "fpr")

print("AUC values DT")
perf<- performance(pred, 'auc')
aucDT <- as.numeric(perf@y.values)
print(aucDT)
```

Results



```
[1] "Accuracies SVM"
 [1] 0.7604167 0.7343750 0.7708333 0.8333333 0.7708333 0.7343750 0.7552083
 [8] 0.7812500 0.7656250 0.7812500 0.8177083 0.7604167 0.7864583 0.7812500
[15] 0.7864583 0.7916667 0.7968750 0.7604167 0.7968750 0.7500000 0.7760417
[22] 0.7864583 0.7552083 0.7812500 0.8177083 0.7395833 0.8177083 0.7343750
[29] 0.8020833 0.8020833
[1] "Accuracies DT"
 [1] 0.7343750 0.7552083 0.7604167 0.7812500 0.7760417 0.7395833 0.7239583
 [8] 0.7291667 0.7552083 0.7291667 0.7552083 0.7447917 0.7656250 0.7291667
[15] 0.7760417 0.7291667 0.7239583 0.7604167 0.7552083 0.6822917 0.6875000
[22] 0.7187500 0.7916667 0.7187500 0.7760417 0.7187500 0.7500000 0.7500000
[29] 0.7656250 0.7395833
[1] "Mean accuracy SVM"
[1] 0.7776042
[1] "Mean accuracy DT"
[1] 0.7440972
[1] "StD SVM"
[1] 0.02645682
[1] "StD SVM"
[1] 0.0259421
[1] "Confusion matrix SVM"
     true
pred  neg pos
  neg 112  23
  pos  15  42
[1] "Confusion matrix DT"
     true
pred  neg pos
  neg 103  26
  pos  24  39
```

Cohen’s K agreement

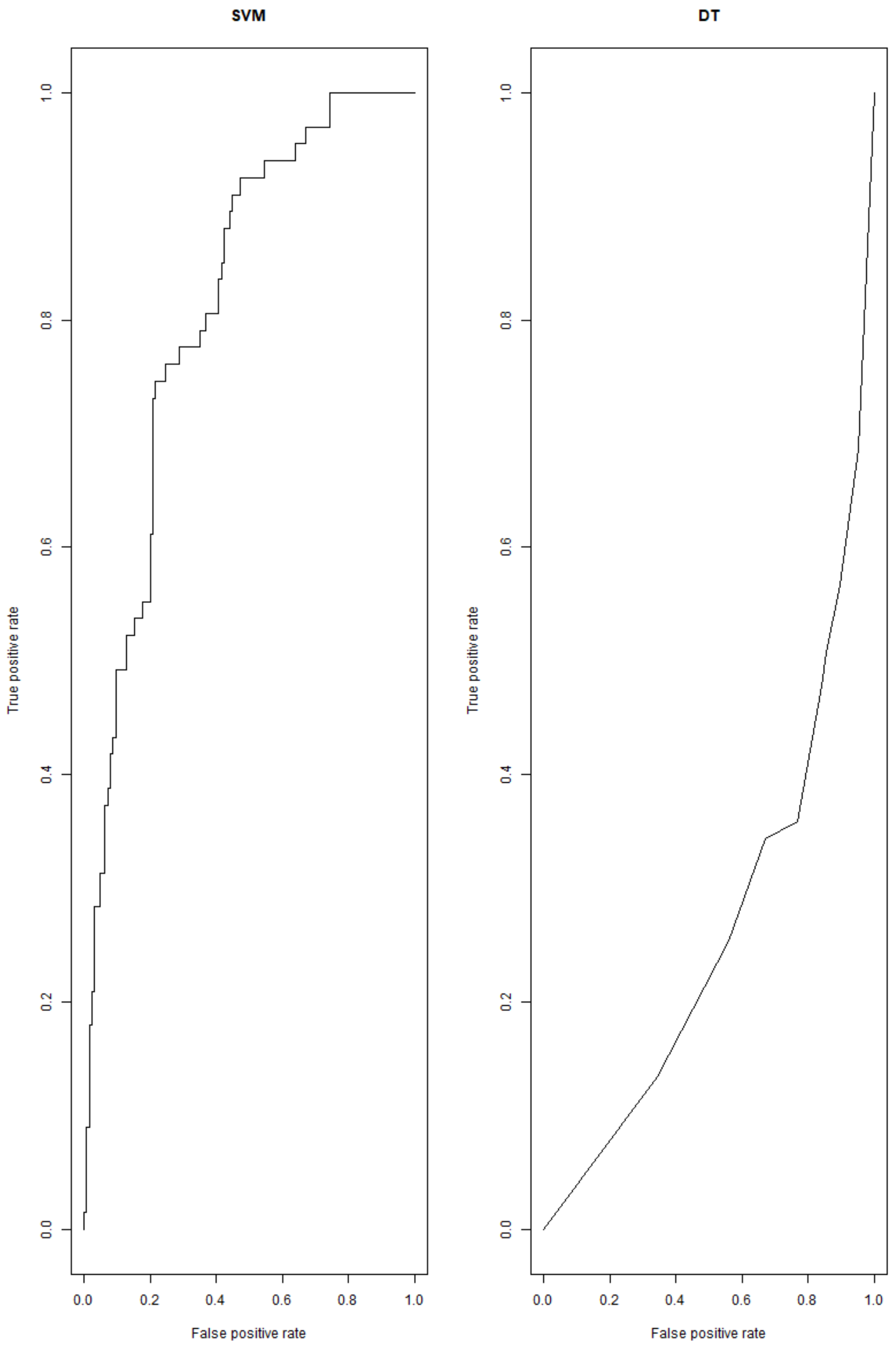


```
        Estimate Cohen's kappa statistics and test the null hypothesis that
        the extent of agreement is same as random (kappa=0)

data:  svm.pred and testset[, classColumn]
Z = 6.611, p-value = 1.909e-11
95 percent confidence interval:
 0.4146655 0.6741318
sample estimates:
[1] 0.5443987


$Judgement
[1] "Moderate agreement"

$Result

        Estimate Cohen's kappa statistics and test the null hypothesis that
        the extent of agreement is same as random (kappa=0)

data:  rpart.pred and testset[, classColumn]
Z = 5.1332, p-value = 1.425e-07
95 percent confidence interval:
 0.2744795 0.5537885
sample estimates:
[1] 0.414134


$Judgement
[1] "Moderate agreement"
```

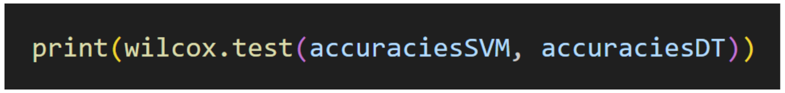
ROC curve

```r
par(mfrow = c(1, 2))
plot(perfSVM)
title('SVM')
plot(perfDT)
title('DT')
```

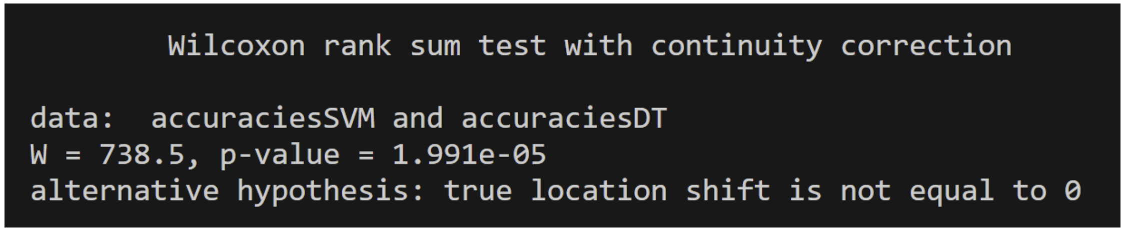
```
[1] "AUC values SVM"
[1] 0.8103881
[1] "AUC values DT"
[1] 0.2671642
```



Wilcoxon test results



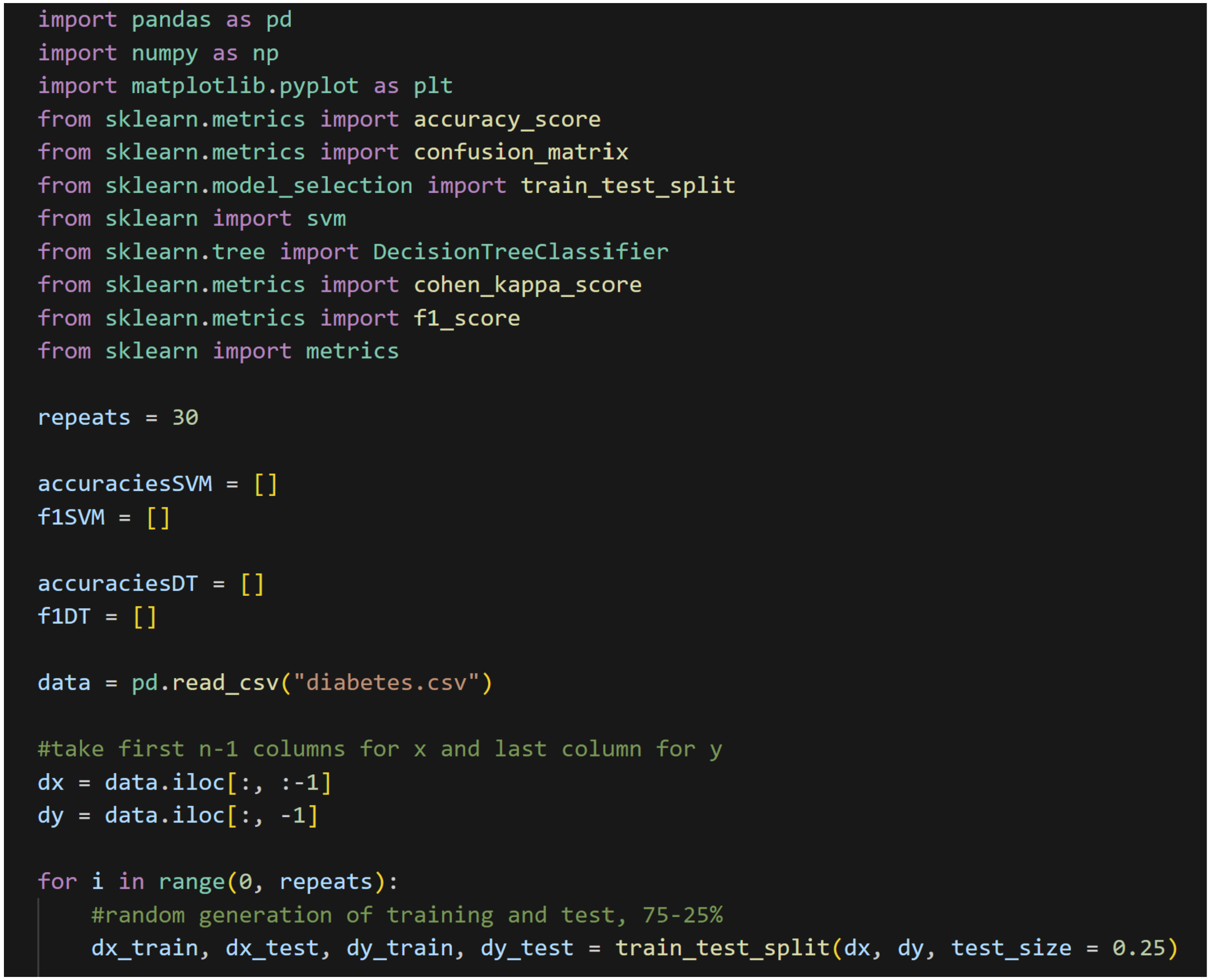
```
print(wilcox.test(accuraciesSVM, accuraciesDT))
```



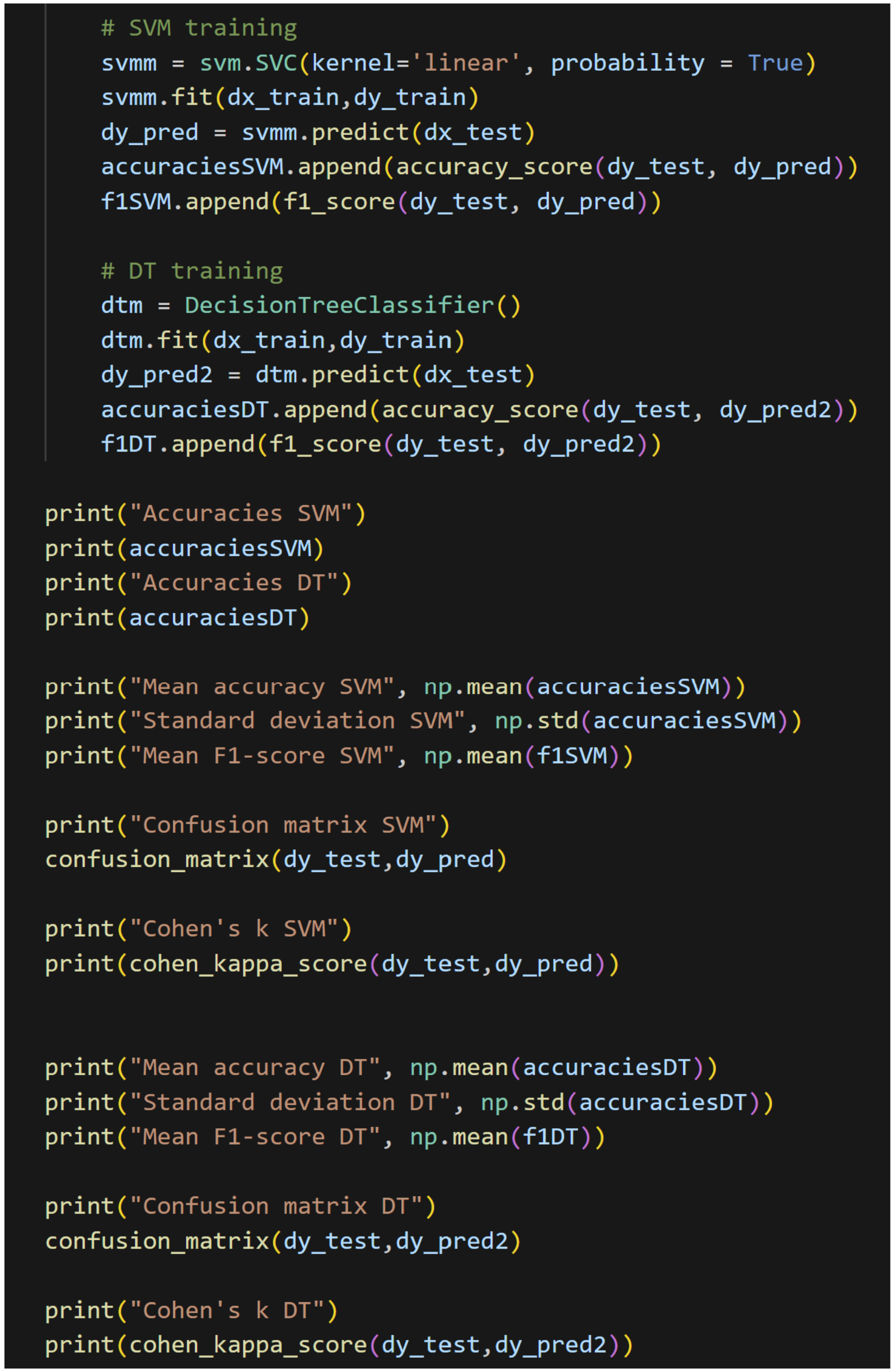
```
        Wilcoxon rank sum test with continuity correction

data:  accuraciesSVM and accuraciesDT
W = 738.5, p-value = 1.991e-05
alternative hypothesis: true location shift is not equal to 0
```

- There is statistical difference between the performances of the two classifiers.

Python

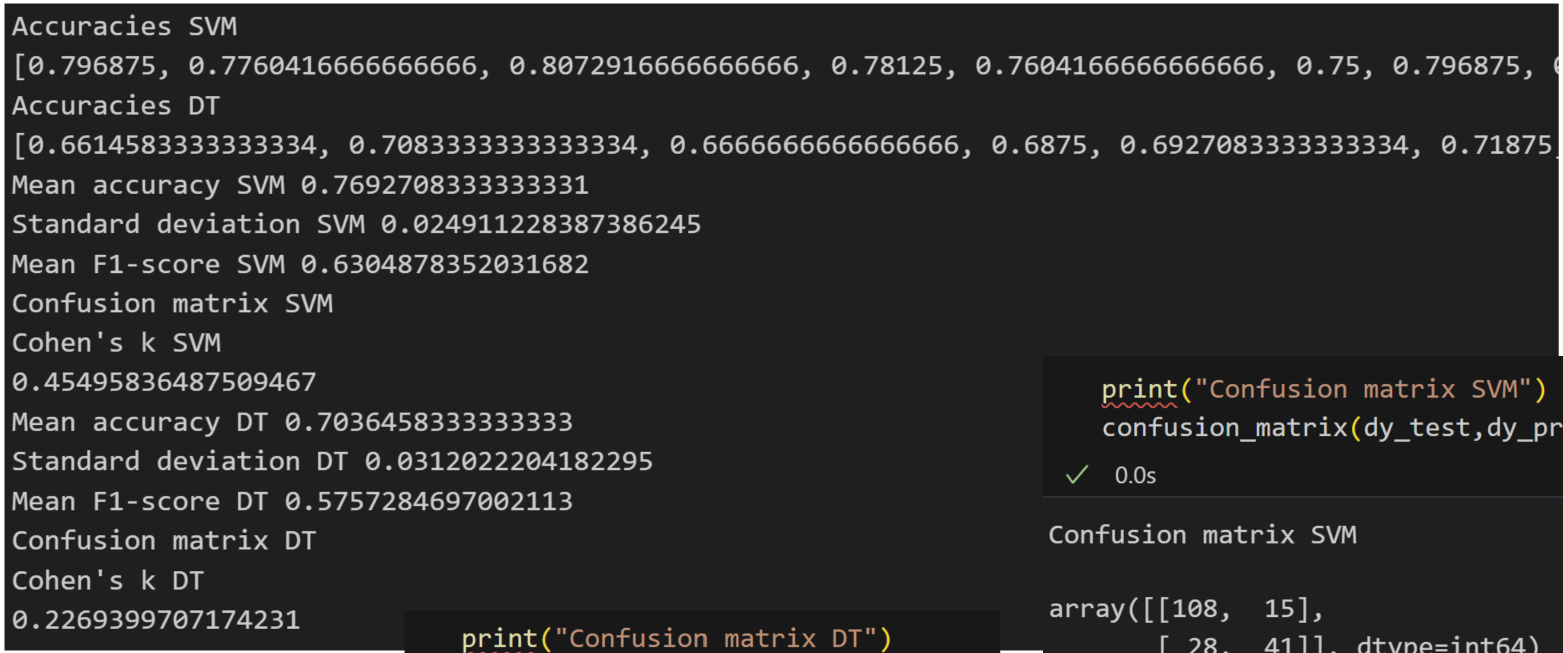


```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import f1_score
from sklearn import metrics

repeats = 30

accuraciesSVM = []
f1SVM = []

accuraciesDT = []
f1DT = []

data = pd.read_csv("diabetes.csv")

#take first n-1 columns for x and last column for y
dx = data.iloc[:, :-1]
dy = data.iloc[:, -1]

for i in range(0, repeats):
    #random generation of training and test, 75-25%
    dx_train, dx_test, dy_train, dy_test = train_test_split(dx, dy, test_size = 0.25)
```

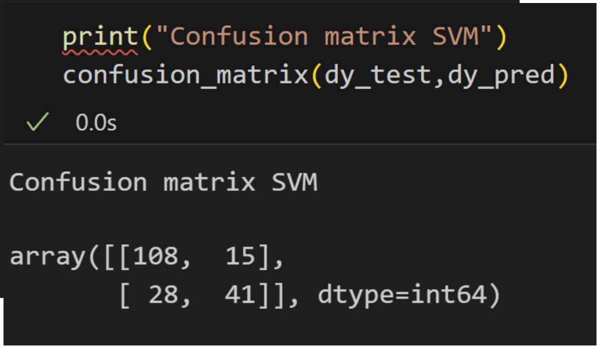


```
    # SVM training
    svmm = svm.SVC(kernel='linear', probability = True)
    svmm.fit(dx_train,dy_train)
    dy_pred = svmm.predict(dx_test)
    accuraciesSVM.append(accuracy_score(dy_test, dy_pred))
    f1SVM.append(f1_score(dy_test, dy_pred))

    # DT training
    dtm = DecisionTreeClassifier()
    dtm.fit(dx_train,dy_train)
    dy_pred2 = dtm.predict(dx_test)
    accuraciesDT.append(accuracy_score(dy_test, dy_pred2))
    f1DT.append(f1_score(dy_test, dy_pred2))

print("Accuracies SVM")
print(accuraciesSVM)
print("Accuracies DT")
print(accuraciesDT)

print("Mean accuracy SVM", np.mean(accuraciesSVM))
print("Standard deviation SVM", np.std(accuraciesSVM))
print("Mean F1-score SVM", np.mean(f1SVM))

print("Confusion matrix SVM")
confusion_matrix(dy_test,dy_pred)

print("Cohen's k SVM")
print(cohen_kappa_score(dy_test,dy_pred))


print("Mean accuracy DT", np.mean(accuraciesDT))
print("Standard deviation DT", np.std(accuraciesDT))
print("Mean F1-score DT", np.mean(f1DT))

print("Confusion matrix DT")
confusion_matrix(dy_test,dy_pred2)

print("Cohen's k DT")
print(cohen_kappa_score(dy_test,dy_pred2))
```

Results



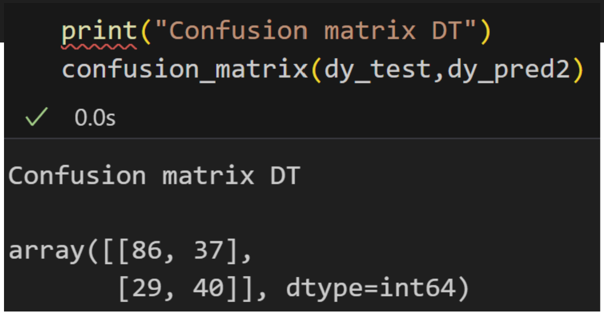
```
Accuracies SVM
[0.796875, 0.7760416666666666, 0.8072916666666666, 0.78125, 0.7604166666666666, 0.75, 0.796875, 
Accuracies DT
[0.6614583333333334, 0.7083333333333334, 0.6666666666666666, 0.6875, 0.6927083333333334, 0.71875
Mean accuracy SVM 0.7692708333333331
Standard deviation SVM 0.024911228387386245
Mean F1-score SVM 0.6304878352031682
Confusion matrix SVM
Cohen's k SVM
0.45495836487509467
Mean accuracy DT 0.7036458333333333
Standard deviation DT 0.0312022204182295
Mean F1-score DT 0.5757284697002113
Confusion matrix DT
Cohen's k DT
0.2269399707174231
```



```
print("Confusion matrix SVM")
confusion_matrix(dy_test,dy_pred)
```

✓ 0.0s

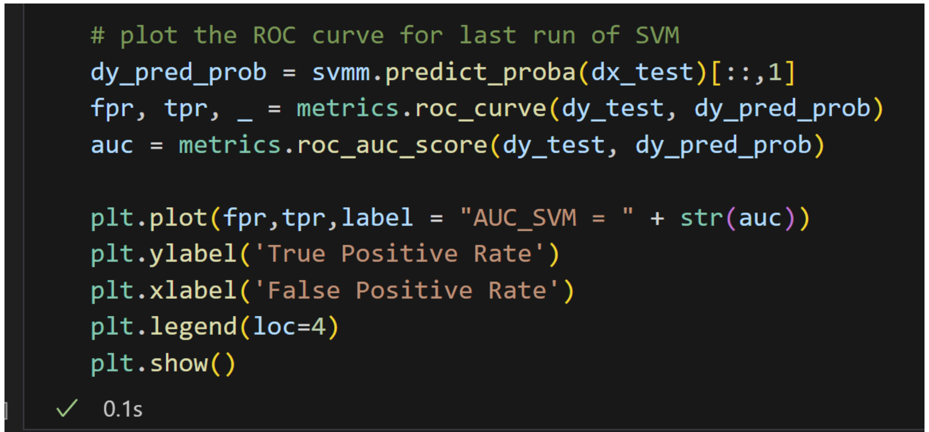
```
Confusion matrix SVM

array([[108,  15],
       [ 28,  41]], dtype=int64)
```



```
print("Confusion matrix DT")
confusion_matrix(dy_test,dy_pred2)
```

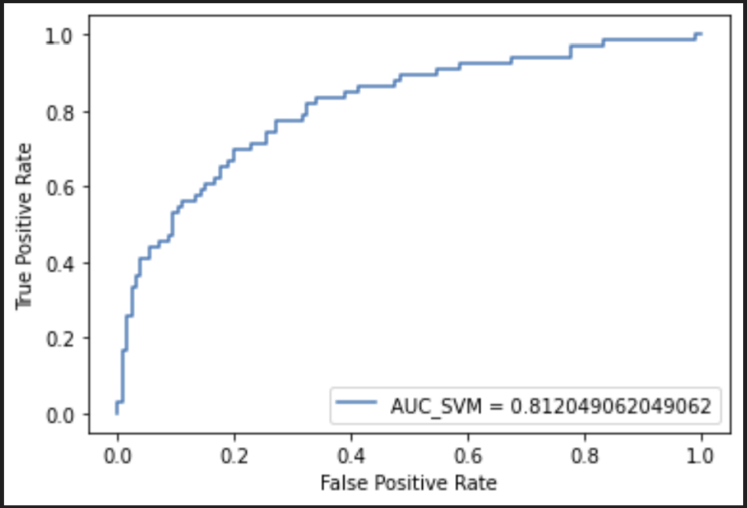
✓ 0.0s

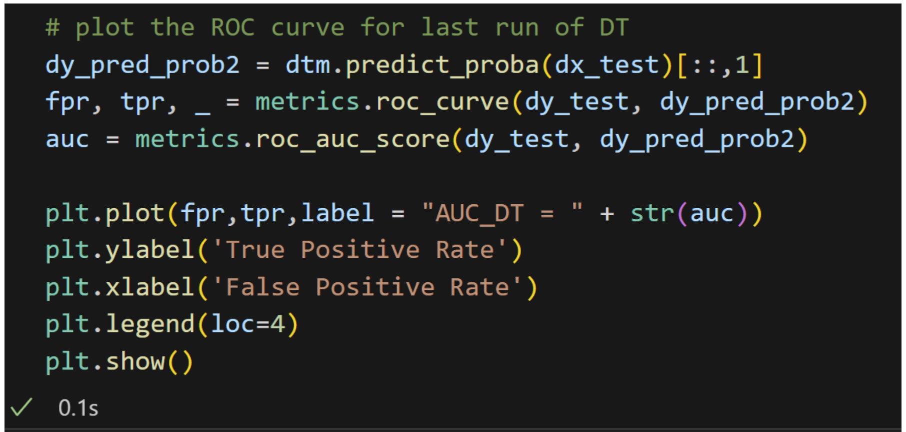
```
Confusion matrix DT

array([[86, 37],
       [29, 40]], dtype=int64)
```



```
# plot the ROC curve for last run of SVM
dy_pred_prob = svmm.predict_proba(dx_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(dy_test, dy_pred_prob)
auc = metrics.roc_auc_score(dy_test, dy_pred_prob)

plt.plot(fpr,tpr,label = "AUC_SVM = " + str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```

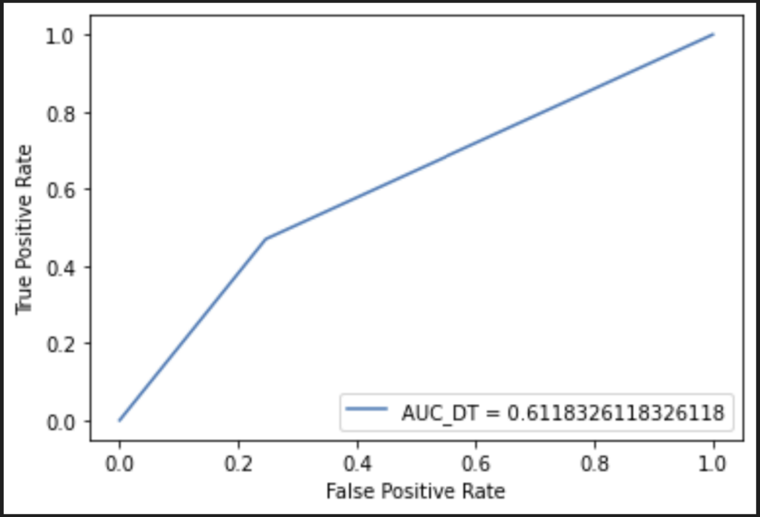
✓ 0.1s

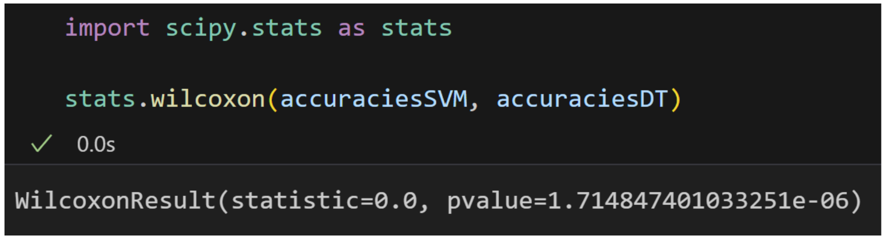




```
# plot the ROC curve for last run of DT
dy_pred_prob2 = dtm.predict_proba(dx_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(dy_test, dy_pred_prob2)
auc = metrics.roc_auc_score(dy_test, dy_pred_prob2)

plt.plot(fpr,tpr,label = "AUC_DT = " + str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```

✓ 0.1s





```
import scipy.stats as stats

stats.wilcoxon(accuraciesSVM, accuraciesDT)
```

✓ 0.0s

```
WilcoxonResult(statistic=0.0, pvalue=1.714847401033251e-06)
```

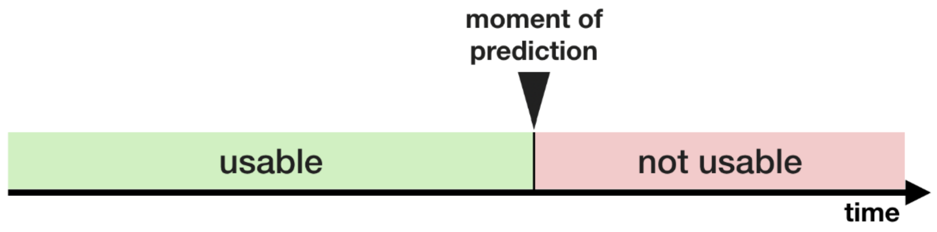
Data leakage

- When training data contains information that will not be found in deployment.

- Train-test contamination
  - When you perform a data pre-processing action before splitting into training-test-validation
  - e.g. data imputation of missing values, scaling

Target leakage

- Predictors that were added to the data set after the moment of prediction



kaggle.com/learn

Target

Age	Fever	Headache	Bacterial infection	Antibiotic
34	Y	Y	Y	Y
30	Y	N	N	N
23	N	Y	N	N

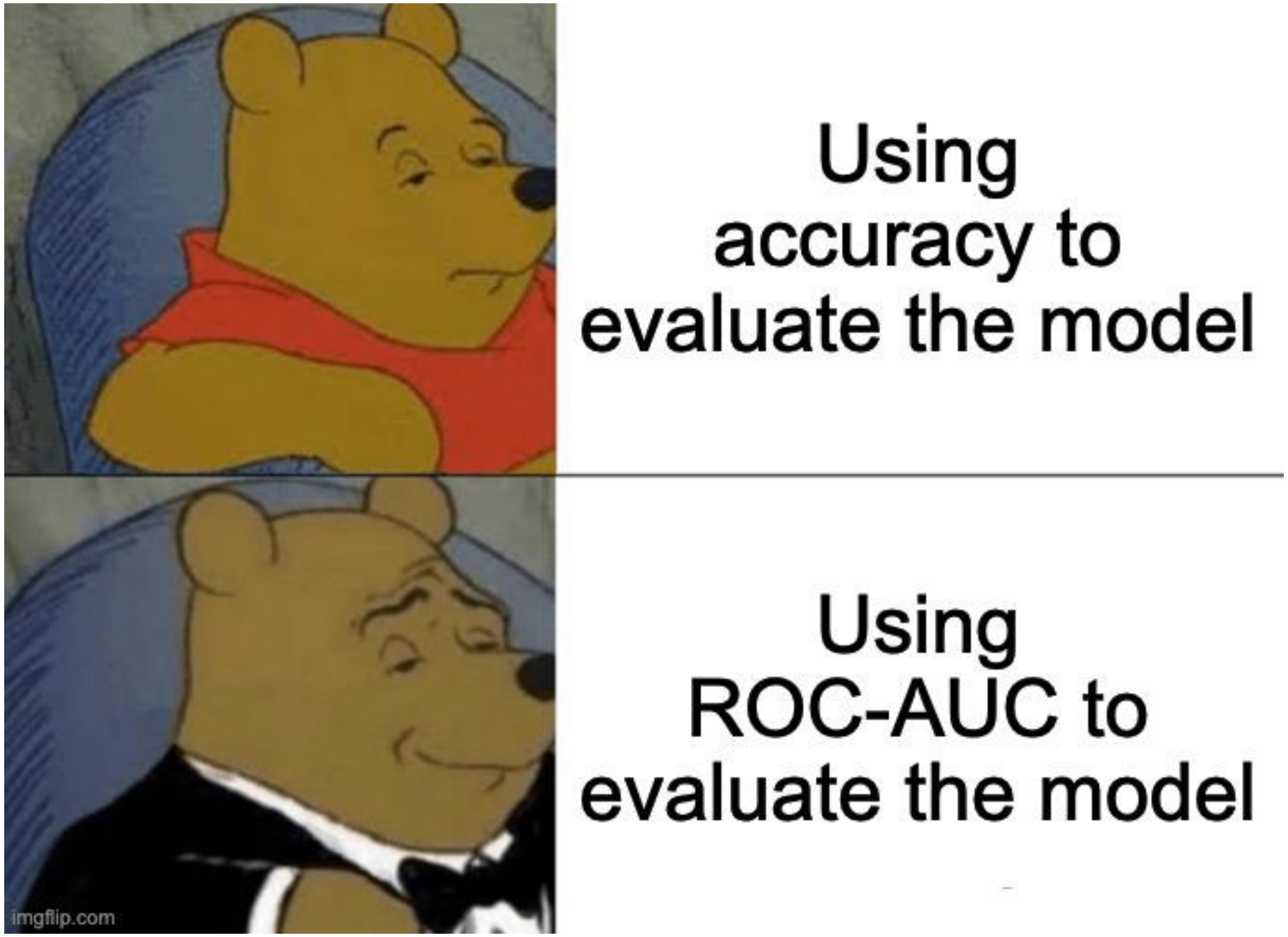
Homework 1/2

- Take one problem (Wisconsin, car or customer) and models from last homework (NN, DT, Ensembles)
- If the task is classification, for each model:
  - Conduct cross-validation by random subsampling with 30 runs.
  - Obtain mean accuracy, standard deviation, F1-score and the confusion matrix.
  - Compute the agreement between predictions and real targets with Cohen's k.
  - Obtain the ROC curve and the AUC value.
  - Apply the Wilcoxon test to compare the predictions of the most accurate two classifiers.

Homework 2/2

- If the task is regression, for each model:
  - Conduct cross-validation cross-validation by random subsampling with 30 runs.
  - Obtain MSE, RMSE, MAE, MAPE, R^2 and Adjusted R^2.
  - Apply the Wilcoxon test to compare the predictions of the least erroneous two regressors.
  - Cohen's k cannot be applied, as the data is not qualitative.

- (**Optional**) Construct at least 3 classifiers for the Iris, Wisconsin breast cancer and Wine data sets and try the application of Friedman's test to rank their performance.


Using accuracy to evaluate the model
Using ROC-AUC to evaluate the model
imgflip.com