COMP 204: Python programming for life sciences

Intro to machine learning with scikit-learn
Part 3

Yue Li
based on material from Mathieu Blanchette, Christopher J.F. Cameron and Carlos G. Oliver

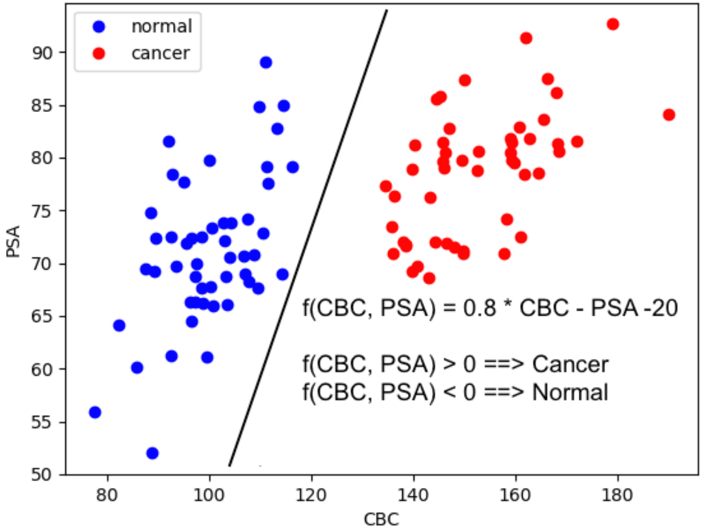
Prostate cancer prediction problem

Suppose you want to learn to predict if a person has a prostate cancer based on two easily-measured variables obtained from blood sample: Complete Blood Count (CBC) and Prostate-specific antigen (PSA). We have collected data from patients known to have or not have prostate cancer:

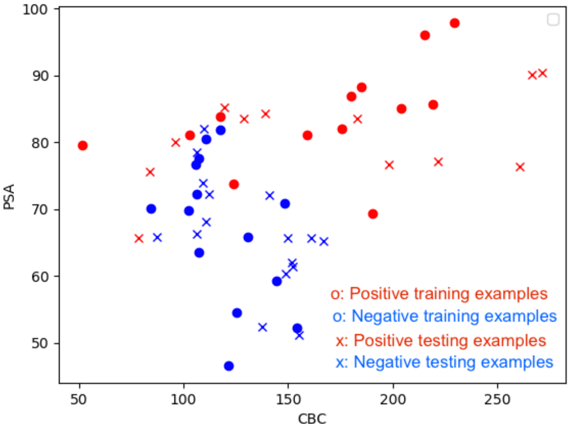
CBC	PSA	Status
142	67	Normal
132	58	Normal
178	69	Cancer
188	46	Normal
183	68	Cancer
...		

Goal: Train classifier to predict the class of new patients, from their CBC and PSA.

A perfect classifier



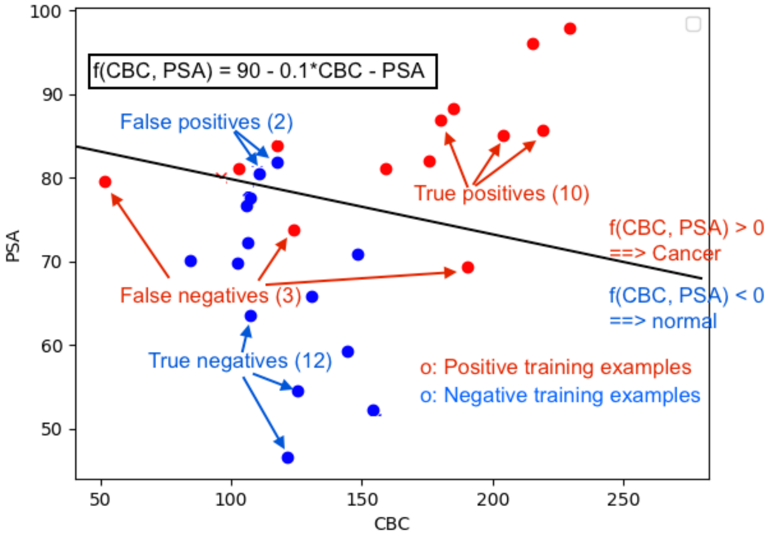
More realistic data



Here, it is impossible to cleanly separate positive and negative examples with a straight line.
$\rightarrow$ We will be bound to make classification errors.

More realistic data

Here: TP = 10, TN = 12, FP = 2, FN = 3.



True/false positives and negatives

True positive (TP)

Positive example that is predicted to be positive

- A person who is predicted to have cancer and actually has cancer

False positive (FP)

Negative example that is predicted to be positive

- A person who is predicted to have cancer and but doesn't have cancer

True negative (TN)

Negative example that is predicted to be negative

- A person who is predicted to not have cancer and actually doesn't have cancer

False negative (FN)

Positive example that is predicted to be negative

- A person who is predicted to not have cancer and but actually has cancer

Confusion matrices

Confusion matrix: A table of counts for TPs, FPs, TNs, and FNs

	predicted negative	predicted positive
actual negative	29	16
actual positive	14	36

In scikit-learn, we can get the confusion matrix for the LR by:

```
from sklearn.metrics import confusion_matrix

X_train, X_test, y_train, y_test = \
train_test_split(X, y, test_size=0.5, random_state=100)

lr_model = LogisticRegression(solver="liblinear")
lr_model.fit(X_train, y_train)
y_test_pred = lr_model.predict(X_test)

cm = confusion_matrix(y_test, y_test_pred)
```

True/false positive rates (pop quiz)

	predicted negative	predicted positive
actual negative	29	16
actual positive	14	36

True positive rate (TPR) (or sensitivity)

The proportion of positive examples that are predicted positive

- Fraction of cancer patients who are predicted to have cancer

$$TPR = \frac{TP}{TP + FN} = \frac{?}{? + ?} = 72\%$$

False positive rate (FPR)

The proportion of negative examples that are predicted to be positive

- Fraction of healthy patients who are predicted to have cancer

$$FPR = \frac{FP}{FP + TN} = \frac{?}{? + ?} = 35\%$$

True/false positive rates

	predicted negative	predicted positive
actual negative	29	16
actual positive	14	36

True positive rate (TPR) (or sensitivity)

The proportion of positive examples that are predicted positive

- Fraction of cancer patients who are predicted to have cancer

$$TPR = \frac{TP}{TP + FN} = \frac{36}{36 + 14} = 72\%$$

False positive rate (FPR)

The proportion of negative examples that are predicted to be positive

- Fraction of healthy patients who are predicted to have cancer

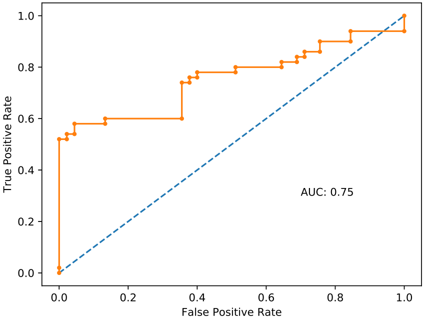
$$FPR = \frac{FP}{FP + TN} = \frac{16}{16 + 29} = 35\%$$

Receiver Operating Characteristic (ROC) curve

- We can create a table for TPR and FPR at each Threshold.
- Draw the ROC curve plots TPR (y-axis) versus FPR (x-axis)
- The area under the curve (AUC) is 75%

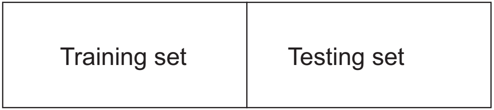
```
y_test_proba = lr_model.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_test_proba)
auc = roc_auc_score(y_test, y_test_proba[:,1])
```

TPR	FPR	Threshold
0.00	0.000000	1.920577
0.02	0.000000	0.920577
0.52	0.000000	0.696330
...	...	...
0.60	0.133333	0.654847
0.60	0.355556	0.611319
0.74	0.355556	0.483655
...	...	...
0.84	0.688889	0.370510
0.84	0.711111	0.367536
0.86	0.711111	0.364517
...	...	...
0.94	1.000000	0.260042
1.00	1.000000	0.115260



K-fold Cross Validation

- In our above example, we split the data into 50% training and 50% testing
- We train and evaluate the model using only half of the data.



- This is quite wasteful. How can we evaluate our model on every data point while training on the rest of the data points?
- Answer: K-fold cross-validation

Five-fold cross validation

Step 1. Randomly split the data $\mathcal{D}$ into 5 folds

$\mathcal{F}1$	$\mathcal{F}2$	$\mathcal{F}3$	$\mathcal{F}4$	$\mathcal{F}5$

Step 2. Training and prediction

Fold 1	$\mathcal{F}1$					Train on $\mathcal{D}$ - $\mathcal{F}1$, predict on $\mathcal{F}1$
Fold 2		$\mathcal{F}2$				Train on $\mathcal{D}$ - $\mathcal{F}2$, predict on $\mathcal{F}2$
Fold 3			$\mathcal{F}3$			Train on $\mathcal{D}$ - $\mathcal{F}3$, predict on $\mathcal{F}3$
Fold 4				$\mathcal{F}4$		Train on $\mathcal{D}$ - $\mathcal{F}4$, predict on $\mathcal{F}4$
Fold 5					$\mathcal{F}5$	Train on $\mathcal{D}$ - $\mathcal{F}5$, predict on $\mathcal{F}5$

Cross validation

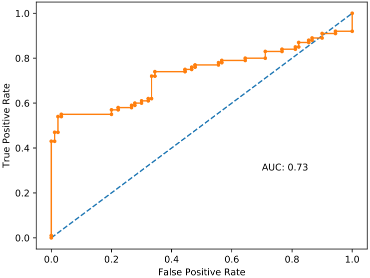
Step 3. Evaluate predictions on all 5 folds by ROC

Predicted probabilities	$\mathcal{F}1$	$\mathcal{F}2$	$\mathcal{F}3$	$\mathcal{F}4$	$\mathcal{F}5$

versus

True labels	$\mathcal{F}1$	$\mathcal{F}2$	$\mathcal{F}3$	$\mathcal{F}4$	$\mathcal{F}5$

ROC curve on ALL data points



Method comparisons

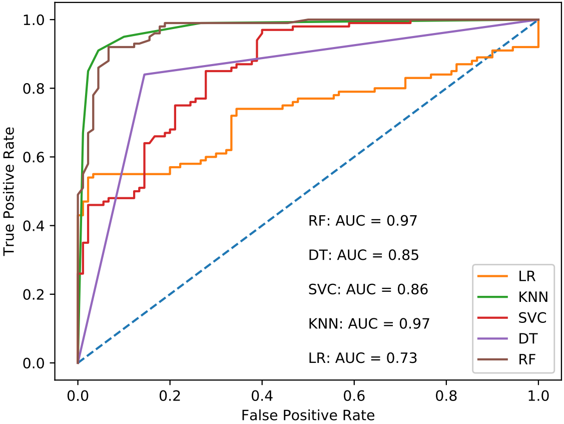
- There are many machine learning methods implemented in scikit-learn
- How do we know which one performs the test on *our data set*?
- To get the answer, we will need to compare these methods using cross validation
- Let's compare the five machine learning methods namely
  - Logistic regression (LR)
  - K-nearest neighbours (KNN)
  - Support vector machine classifier (SVC)
  - Decision tree classifier (DT)
  - Random forest (RF): an ensemble approach that averages predictions from many decision trees (default: 100 trees)
- Note: for each method (or class), we create an *object* of the method using their initializer method defined under that class (OOP reminder)
- Training and prediction follows the *generic* syntax

Method comparisons using scikit-learn

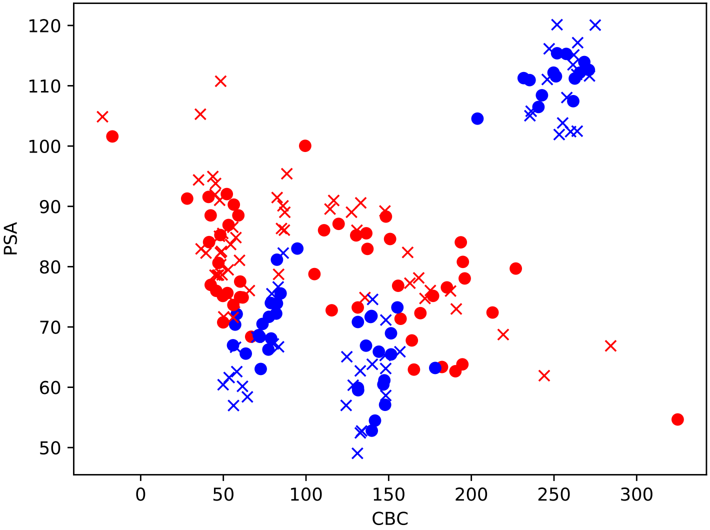
```python
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

models = [LogisticRegression(solver="liblinear"),
          KNeighborsClassifier(),
          SVC(probability=True, gamma='auto'),
          DecisionTreeClassifier(),
          RandomForestClassifier(n_estimators=100)]
perf = {}
for model in models:
    model_name = type(model).__name__
    print(model_name)
    label,pred = cross_validate(model, X_flat, Y)
    fpr, tpr, thresholds = roc_curve(label, pred)
    auc = roc_auc_score(label, pred)
    perf[model_name] = {'fpr':fpr,'tpr':tpr,'auc':auc}
```

ROC curves and AUC for all of the four methods

- The best method is KNN or RF (a tie; AUC: 0.97).
- LR did the worst because our data are not linearly separable
- In contrast, KNN, DT, and RF are non-linear methods
- SVC transforms the data to make them linearly separable

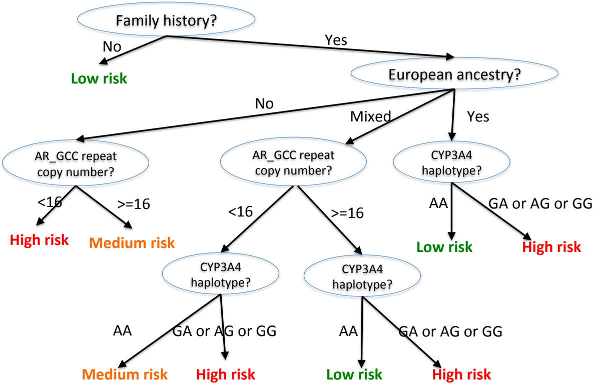


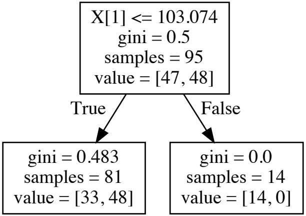
Non-linearly separable data

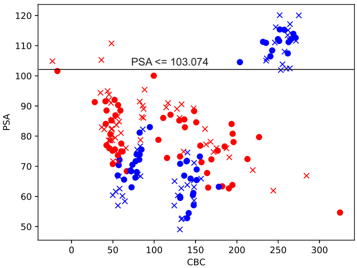


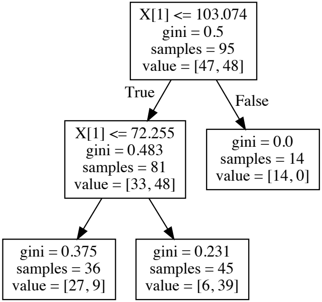
Decision tree

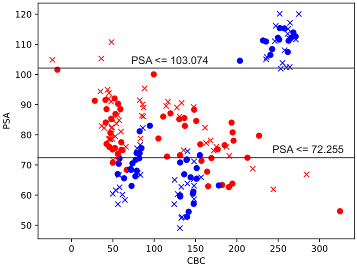
Linear classifiers are limited discriminate complex data structure.
Another type of classifier is called a decision tree (API).
We have seen decision tree as a rule-based approach.
How can we *learn* decision trees from the data?

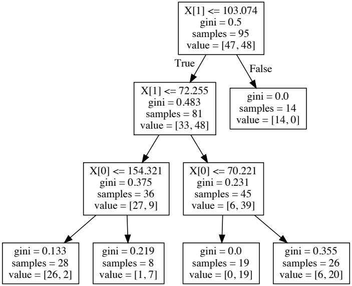


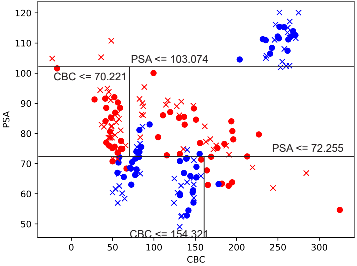

X[1] <= 103.074
gini = 0.5
samples = 95
value = [47, 48]
True
False
gini = 0.483
samples = 81
value = [33, 48]
gini = 0.0
samples = 14
value = [14, 0]

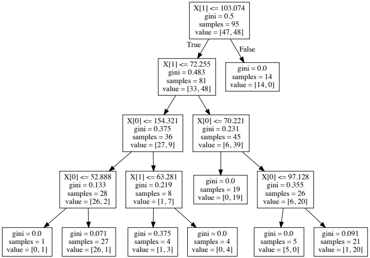

PSA <= 103.074
PSA
CBC

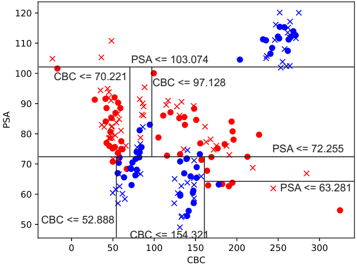

X[1] <= 103.074
gini = 0.5
samples = 95
value = [47, 48]
True
False
X[1] <= 72.255
gini = 0.483
samples = 81
value = [33, 48]
gini = 0.0
samples = 14
value = [14, 0]
gini = 0.375
samples = 36
value = [27, 9]
gini = 0.231
samples = 45
value = [6, 39]


PSA <= 103.074
PSA <= 72.255
PSA
CBC

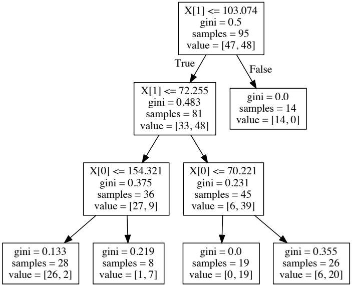

X[1] <= 103.074
gini = 0.5
samples = 95
value = [47, 48]
True
False
X[1] <= 72.255
gini = 0.483
samples = 81
value = [33, 48]
gini = 0.0
samples = 14
value = [14, 0]
X[0] <= 154.321
gini = 0.375
samples = 36
value = [27, 9]
X[0] <= 70.221
gini = 0.231
samples = 45
value = [6, 39]
gini = 0.133
samples = 28
value = [26, 2]
gini = 0.219
samples = 8
value = [1, 7]
gini = 0.0
samples = 19
value = [0, 19]
gini = 0.355
samples = 26
value = [6, 20]

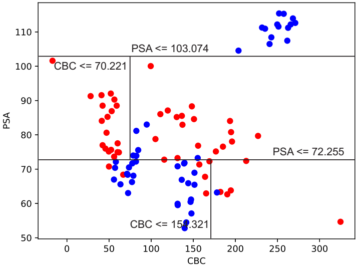

PSA <= 103.074
CBC <= 70.221
PSA <= 72.255
CBC <= 154.321
PSA
CBC
120
110
100
90
80
70
60
50
0
50
100
150
200
250
300


X[1] <= 103.074
gini = 0.5
samples = 95
value = [47, 48]
True
False
X[1] <= 72.255
gini = 0.483
samples = 81
value = [33, 48]
gini = 0.0
samples = 14
value = [14, 0]
X[0] <= 154.321
gini = 0.375
samples = 36
value = [27, 9]
X[0] <= 70.221
gini = 0.231
samples = 45
value = [6, 39]
X[0] <= 52.888
gini = 0.133
samples = 28
value = [26, 2]
X[1] <= 63.281
gini = 0.219
samples = 8
value = [1, 7]
gini = 0.0
samples = 19
value = [0, 19]
X[0] <= 97.128
gini = 0.355
samples = 26
value = [6, 20]
gini = 0.0
samples = 1
value = [0, 1]
gini = 0.071
samples = 27
value = [26, 1]
gini = 0.375
samples = 4
value = [1, 3]
gini = 0.0
samples = 4
value = [0, 4]
gini = 0.0
samples = 5
value = [5, 0]
gini = 0.091
samples = 21
value = [1, 20]


PSA <= 103.074
CBC <= 70.221
CBC <= 97.128
PSA <= 72.255
PSA <= 63.281
CBC <= 52.888
CBC <= 154.321
PSA
CBC
120
110
100
90
80
70
60
50
0
50
100
150
200
250
300

Confusion matrix for the decision tree at max_depth = 3





Training data:

	Predicted negative (PN)	Predicted positive (PP)
Negative (0)	40	7
Positive (1)	2	46

$\text{TPR} = 46/(46+2) = 0.96$

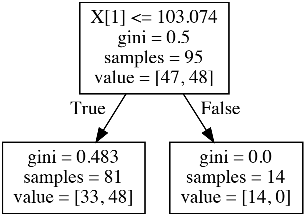
$\text{FPR} = 7/(7+40) = 0.15$

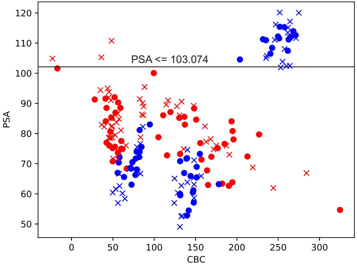
Decision tree in Python scikit-learn

Note: Requires installing graphviz: `pip install graphviz`

```python
from sklearn.metrics import confusion_matrix
from sklearn import model_selection,tree

depth = 3
clf = tree.DecisionTreeClassifier(max_depth=depth)
clf.fit(X_train, y_train)
p_train = clf.predict(X_train)
p_test = clf.predict(X_test)

#plot tree
dot_data = tree.export_graphviz(clf, out_file=None)
graph = graphviz.Source(dot_data)
graph.render("prostate_tree_depth_"+str(depth))

# calculate training and testing error
tn,fp,fn,tp = confusion_matrix(y_train,p_train).ravel()
print("Training data:",tn,fp,fn,tp)
tn,fp,fn,tp = confusion_matrix(y_test,p_test).ravel()
print("Test data:",tn,fp,fn,tp)
```

Decision tree (max_depth = 1)





Training data:

	PN	PP
0	14	33
1	0	48

$\text{TPR} = 48/(48+0) = 1.0$

$\text{FPR} = 33/(33+14) = 0.7$

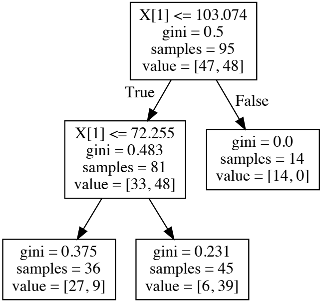
Test data:

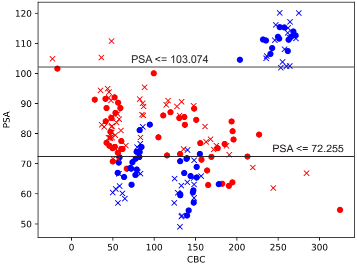
	PN	PP
0	13	30
1	3	49

$\text{TPR} = 49/(49+3) = 0.94$

$\text{FPR} = 30/(30+13) = 0.7$

Decision tree (max_depth = 2)





Training data:

	PN	PP
0	41	6
1	9	39

$\text{TPR} = 39/(39+9) = 0.81$

$\text{FPR} = 6/(6+41) = 0.13$

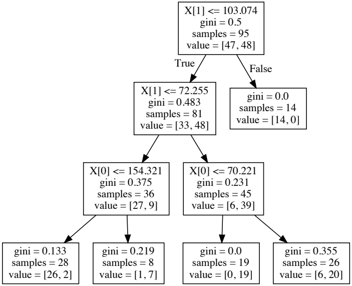
Test data:

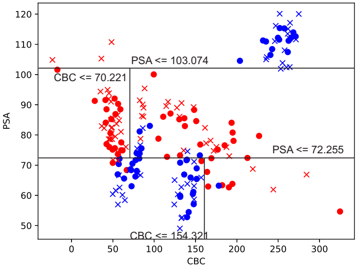
	PN	PP
0	36	7
1	8	44

$\text{TPR} = 44/(44+8) = 0.85$

$\text{FPR} = 7/(7+36) = 0.16$

Decision tree (max_depth = 3)





Training data:

	PN	PP
0	40	7
1	2	46

$\text{TPR} = 46/(46+2) = 0.96$

$\text{FPR} = 7/(7+40) = 0.15$

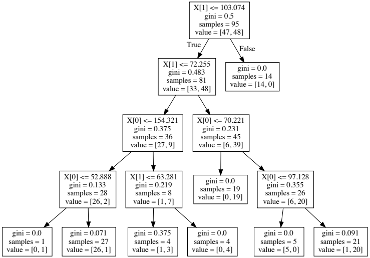
Test data:

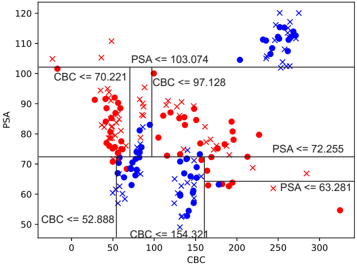
	PN	PP
0	35	8
1	5	47

$\text{TPR} = 47/(47+5) = 0.9$

$\text{FPR} = 8/(8+35) = 0.19$

Decision tree (max_depth = 4) - overfitting occurs





Training data:

	PN	PP
0	45	2
1	1	47

TPR = 47/(47+1) = 0.98
FPR = 2/(2+45) = 0.04

Test data:

	PN	PP
0	37	6
1	11	41

TPR = 41/(41+11) = 0.79
FPR = 6/(6+37) = 0.14

Decision tree (max_depth = 5 & 6) - more overfitting

Tree depth = 5

Training data:

	PN	PP
0	46	1
1	1	47

TPR = $47/(47+1) = 0.98$
FPR = $1/(1+46) = 0.02$

Test data:

	PN	PP
0	37	6
1	11	41

TPR = $41/(41+11) = 0.79$
FPR = $6/(6+37) = 0.14$

Tree depth = 6

Training data:

	PN	PP
0	47	0
1	0	48

TPR = $48/(48+0) = 1.0$
FPR = $0/(0+47) = 0.0$

Test data:

	PN	PP
0	37	6
1	11	41

TPR = $41/(41+11) = 0.79$
FPR = $6/(6+37) = 0.14$

ML - closing comments

Very powerful algorithms exist and are available in scikit-learn:

- Decision trees and decision forests
- Support vector machines
- Neural networks
- etc. etc.

These algorithms can be used for classification / regression based on all kinds of data:

- Arrays of numerical values
- Images, video, sound
- Text
- etc. etc.

Applications in life sciences

- Medical diagnostic
- Interpretation of genetic data
- Drug design, optimization of medical devices
- Modeling of ecosystems
- etc. etc.

Experiment with different approaches/problems!