COMP 204

Intro to machine learning with scikit-learn (part two)

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

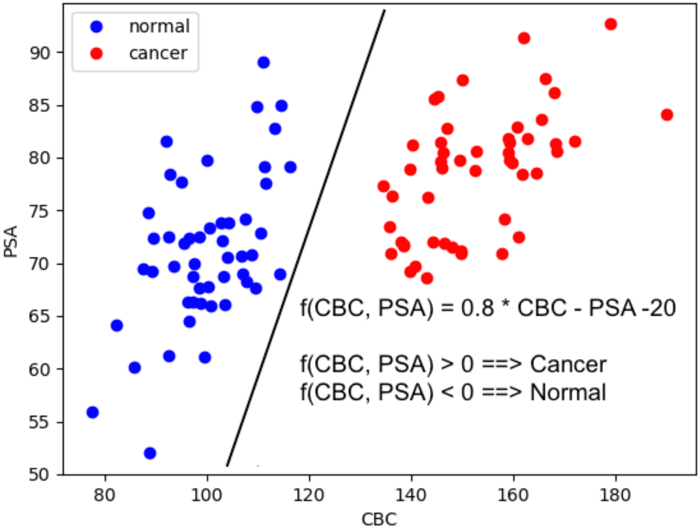
Return to our prostate cancer prediction problem

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:

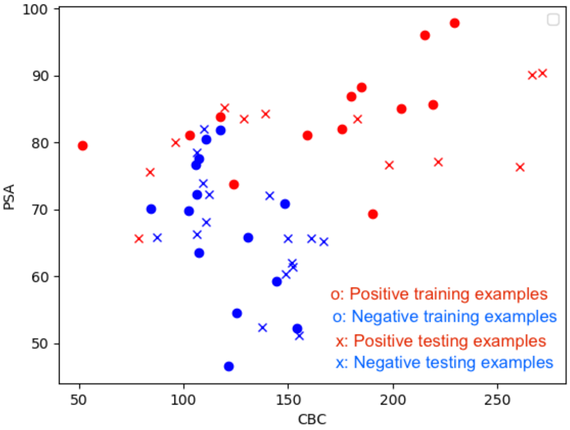
CBC	PSA	Status
142	67	Normal
132	58	Normal
178	69	Cancer
188	46	Normal
183	68	Cancer
...		

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.

→ We will be bound to make classification errors.

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

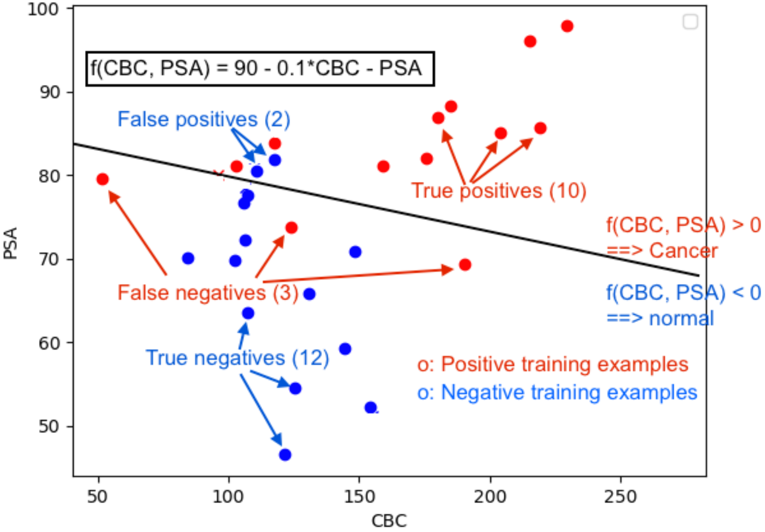
False negative (FN)

Positive example that is predicted to be negative

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

More realistic data

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



Confusion matrices

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

	Predicted positive	Predicted negative
Actual positive	TP = 10	FN = 3
Actual negative	FP = 2	TN = 12

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

```
from sklearn.metrics import confusion_matrix

clf = svm.SVC()
clf.fit(X_train, y_train)
preds = clf.predict(X_test)
tn, fp, fn, tp = confusion_matrix(y_test, preds).ravel()
```

True/false positive rates

Sensitivity: Pproportion of positive examples that are predicted to be positive

- Fraction of cancer patients who are predicted to have cancer

$$Sensitivity = \frac{TP}{TP + FN} = \frac{10}{10 + 3} = 77\%$$

Specificity: Proportion of negative examples that are predicted to be negative

- Fraction of healthy patients who are predicted to be healthy

$$Specificity = \frac{TN}{FP + TN} = \frac{12}{2 + 12} = 86\%$$

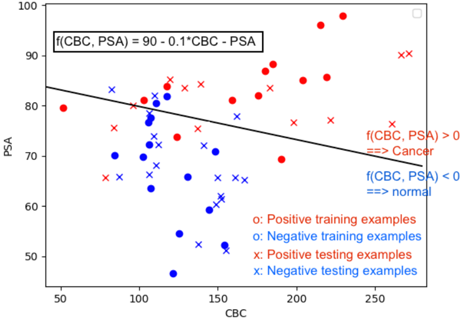
False-positive rate (FPR): Proportion of negative examples that are predicted to be positive

- Fraction of healthy patients who are predicted to have cancer

$$FPR = \frac{FP}{FP + TN} = 1 - specificity = \frac{2}{2 + 12} = 14\%$$

Accuracy on training vs testing sets

To get an unbiased estimation of the accuracy of a predictor, we need to evaluate it against our test data (not used for the training).



	Predicted positive	Predicted negative
Actual positive	TP $= 9$	FN $= 4$
Actual negative	FP $= 3$	TN $= 15$

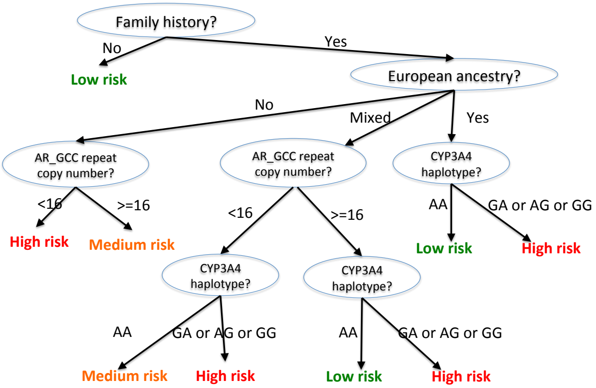
$Sens = \frac{TP}{TP+FN} = \frac{9}{9+4} = 69\%$, $FPR = \frac{FP}{FP+TN} = \frac{3}{3+15} = 17\%$

Decision tree

Linear classifiers are limited in how well they can match the training data.

Another type of classifier is called a decision tree.

http://scikit-learn.org/stable/modules/tree.html

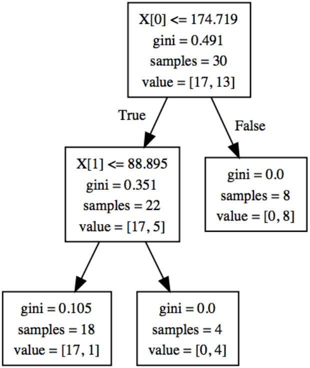


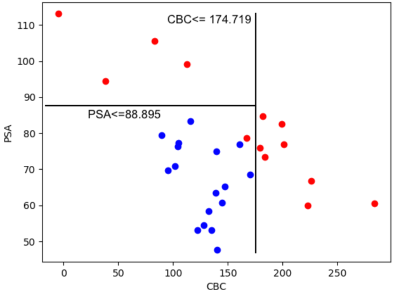
Decision tree in Python

Note: Requires installing graphviz by running "pip install graphviz"

```
import graphviz
from sklearn import model_selection
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

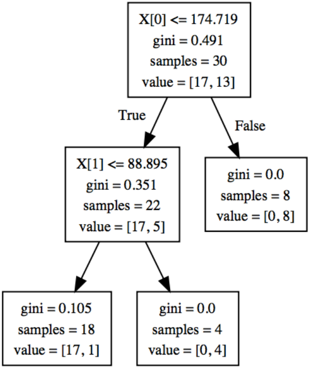


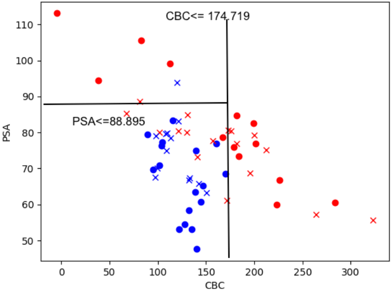


$$Sens = \frac{TP}{TP+FN} = \frac{12}{12+1} = 92\%, \; FPR = \frac{FP}{FP+TN} = \frac{0}{0+17} = 0\%$$

Great accuracy on training set!

Decision tree

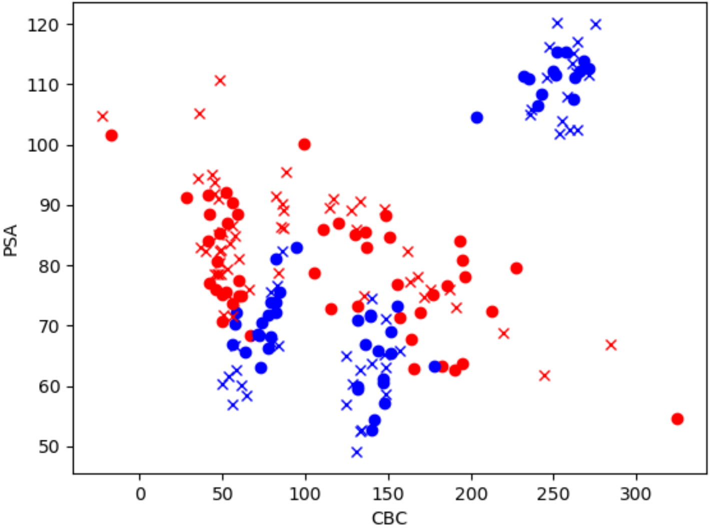




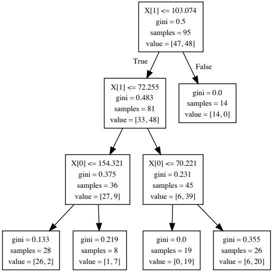
$$Sens = \frac{TP}{TP+FN} = \frac{9}{9+8} = 53\%, \ FPR = \frac{FP}{FP+TN} = \frac{1}{1+11} = 8\%$$

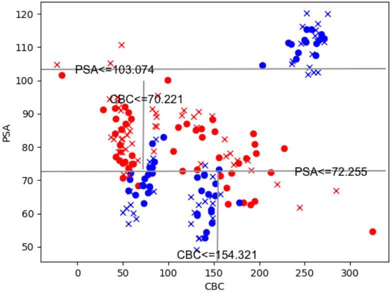
Not so good on the test set...

A harder example



Decision tree (max_depth = 3)

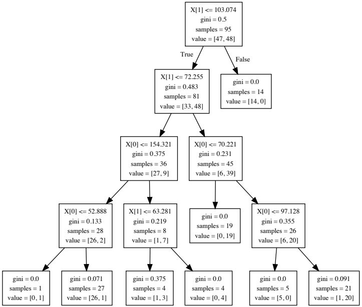


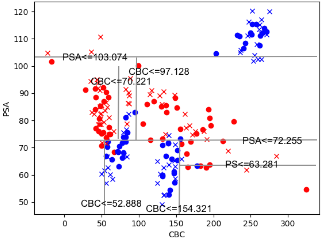


$$sens(train) = \frac{TP}{TP+FN} = \frac{41}{41+6} = 87\%,$$
$$FPR(train) = \frac{FP}{FP+TN} = \frac{9}{9+39} = 19\%$$

$$sens(test) = \frac{TP}{TP+FN} = \frac{36}{36+7} = 84\%,$$
$$FPR(test) = \frac{FP}{FP+TN} = \frac{8}{8+44} = 15\%$$

Deeper trees - max_depth = 4





$$sens(train) = \frac{TP}{TP+FN} = \frac{45}{45+2} = 96\%,$$
$$FPR(train) = \frac{FP}{FP+TN} = \frac{1}{1+47} = 2\%$$

$$sens(test) = \frac{TP}{TP+FN} = \frac{37}{37+6} = 86\%,$$
$$FPR(test) = \frac{FP}{FP+TN} = \frac{11}{11+41} = 21\%$$

Accuracy on training data is much higher than on testing data: overfitting! We've gone too far!

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!