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
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:

<table>
<thead>
<tr>
<th>CBC</th>
<th>PSA</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>67</td>
<td>Normal</td>
</tr>
<tr>
<td>132</td>
<td>58</td>
<td>Normal</td>
</tr>
<tr>
<td>178</td>
<td>69</td>
<td>Cancer</td>
</tr>
<tr>
<td>188</td>
<td>46</td>
<td>Normal</td>
</tr>
<tr>
<td>183</td>
<td>68</td>
<td>Cancer</td>
</tr>
</tbody>
</table>

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

\[ f(\text{CBC}, \text{PSA}) = 0.8 \times \text{CBC} - \text{PSA} - 20 \]

- \( f(\text{CBC}, \text{PSA}) > 0 \) \implies \text{Cancer}
- \( f(\text{CBC}, \text{PSA}) < 0 \) \implies \text{Normal}
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

<table>
<thead>
<tr>
<th>Actual positive</th>
<th>Predicted positive</th>
<th>Predicted negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP = 10</td>
<td>FN = 3</td>
<td></td>
</tr>
<tr>
<td>FP = 2</td>
<td>TN = 12</td>
<td></td>
</tr>
</tbody>
</table>

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

```python
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

**True positive rate (TPR) (aka sensitivity)**
The proportion of positive examples that are predicted positive
▶ Fraction of cancer patients who are predicted to have cancer

\[
TPR = \frac{TP}{TP + FN} = \frac{10}{10 + 3} = 77\%
\]

**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

\[
TPR = \frac{FP}{FP + TN} = \frac{2}{2 + 12} = 14\%
\]
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
- Actual positive: TP = 9
- Actual negative: FP = 3

The True Positive Rate (TPR) is calculated as:

\[ TPR = \frac{TP}{TP + FN} = \frac{9}{9 + 4} = 69\% \]

### Predicted negative
- Actual positive: FN = 4
- Actual negative: TN = 15

The False Positive Rate (FPR) is calculated as:

\[ FPR = \frac{FP}{FP + TN} = \frac{3}{3 + 15} = 17\% \]
Decision tree

Linear classifiers are limited in how well they can match the training data. Another type of classifier is called a decision tree. 

Decision tree in Python

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

```python
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	n, 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

\[
\text{TPR} = \frac{TP}{TP + FN} = \frac{12}{12 + 1} = 92\%
\]

\[
\text{FPR} = \frac{FP}{FP + TN} = \frac{0}{0 + 17} = 0\%
\]

Great accuracy on training set!
Decision tree

\[
TPR = \frac{TP}{TP + FN} = \frac{9}{9 + 8} = 53%,
\]

\[
FPR = \frac{FP}{FP + TN} = \frac{1}{1 + 11} = 8%.
\]

Not so good on the test set...
A harder example
Decision tree (max_depth = 3)

X[1] <= 103.074
gini = 0.5
samples = 95
value = [47, 48]

X[1] <= 72.255
gini = 0.483
samples = 81
value = [33, 48]

True

False

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]

TPR (train) = \frac{TP}{TP + FN} = \frac{41}{41 + 6} = 87%,

FPR (train) = \frac{FP}{FP + TN} = \frac{9}{9 + 39} = 19%,

TPR (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

```
Deeper trees - max_depth = 4

X[1] <= 103.074
   gini = 0.5
   samples = 95
   value = [47, 48]
   True
   X[1] <= 72.255
      gini = 0.483
      samples = 81
      value = [33, 48]
      False
         gini = 0.0
         samples = 14
         value = [14, 0]

X[0] <= 154.321
   gini = 0.375
   samples = 36
   value = [27, 9]
   True
   X[0] <= 70.221
      gini = 0.231
      samples = 45
      value = [6, 39]
      False
         gini = 0.0
         samples = 14
         value = [14, 0]

X[0] <= 52.888
   gini = 0.333
   samples = 28
   value = [26, 2]
   True
   X[1] <= 63.281
      gini = 0.219
      samples = 8
      value = [1, 7]
      False
         gini = 0.0
         samples = 5
         value = [5, 0]

X[0] <= 97.128
   gini = 0.355
   samples = 26
   value = [6, 20]
   True
   X[1] <= 103.074
      gini = 0.5
      samples = 95
      value = [47, 48]
      False
         gini = 0.0
         samples = 14
         value = [14, 0]
```

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

TPR (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!