COMP 204

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

Mathieu Blanchette

Today - Machine learning in Python

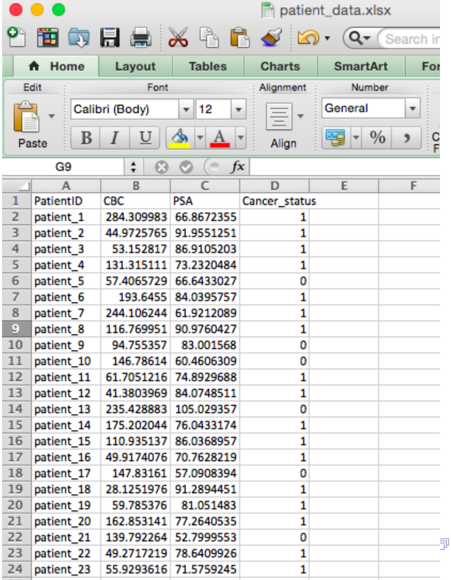
scikit-learn is a Python module that includes most basic machine learning approaches. We will learn how to use it.

Pandas is a Python module that allows reading, writing, and manipulating tabular data.

Pandas and scikit-learn work great together.

Reading in data from Excel file

With Pandas, we can easily import tabular data from a variety of formats.



PatientID	CBC	PSA	Cancer_status
patient_1	284.309983	66.8672355	1
patient_2	44.9725765	91.9551251	1
patient_3	53.152817	86.9105203	1
patient_4	131.315111	73.2320484	1
patient_5	57.4065729	66.6433027	0
patient_6	193.6455	84.0395757	1
patient_7	244.106244	61.9212089	1
patient_8	116.769951	90.9760427	1
patient_9	94.755357	83.001568	0
patient_10	146.78614	60.4606309	0
patient_11	61.7051216	74.8929688	1
patient_12	41.3803969	84.0748511	1
patient_13	235.428883	105.029357	0
patient_14	175.202044	76.0433174	1
patient_15	110.935137	86.0368957	1
patient_16	49.9174076	70.7628219	1
patient_17	147.83161	57.0908394	0
patient_18	28.1251976	91.2894451	1
patient_19	59.785376	81.051483	1
patient_20	162.853141	77.2640535	1
patient_21	139.792264	52.7999553	0
patient_22	49.2717219	78.6409926	1
patient_23	55.9293616	71.5759245	1

Reading in data from Excel file

With Pandas, we can easily import tabular data from a variety of formats.

```
import numpy as np
import pandas as pd

# parse Excel '.xls' file
xls = pd.ExcelFile("patient_data.xlsx")
# extract first sheet in Excel file
data = xls.parse(0)
print(data)
"""
        PatientID         CBC         PSA  Cancer_status
0       patient_1  284.309983   66.867236              1
1       patient_2   44.972576   91.955125              1
2       patient_3   53.152817   86.910520              1
...
"""
```

Processing data frame

With Pandas, we can easily import tabular data from a variety of formats.

```python
# extract CBC and PSA columns
# X are the features from which we want to make a prediction
X = data[["CBC","PSA"]].values # X is a numpy ndarray
print(X)
"""
[[284.3099833   66.8672355 ]
 [ 44.97257649  91.9551251 ]
 [ 53.15281695  86.91052025]
 [131.31511091  73.23204844]
 [ 57.40657286  66.6433027 ]
 ...
 """

# extract cancer_status
y = data["Cancer_status"].values
print(y) # [1 1 1 1 0 1 1 1 0...]
print(X.shape, y.shape)  # (190, 2) (190,)
```

Split training and testing data

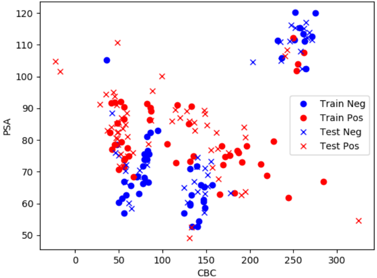
In supervised learning, it is essential to leave aside some data to evaluate the predictor after it will be trained.

This is achieved by splitting the data into a training set and a test set.

```
from sklearn import model_selection
# split data into training and test datasets
X_train, X_test, y_train, y_test = \
    model_selection.train_test_split(X, y, \
                                     test_size = 0.5,\
                                     shuffle = True,\
                                     random_state = 1)
print(X_train.shape, y_train.shape)  # (95, 2) (95,)
print(X_test.shape, y_test.shape)    # (95, 2) (95,)
```

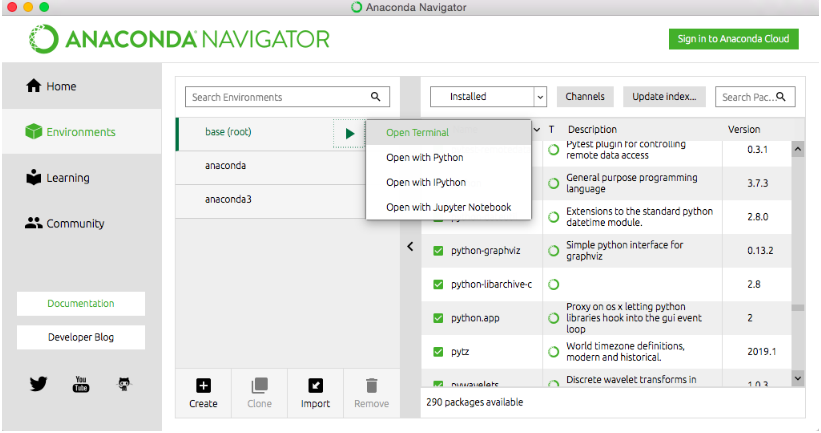
Plotting train/test data

```
import matplotlib.pyplot as plt
plt.plot(X_train[y_train==0,0],X_train[ y_train==0,1],\
         "ob",label="Train Neg")
plt.plot(X_train[y_train==1,0],X_train[ y_train==1,1],\
         "or",label="Train Pos")
plt.plot(X_test[y_test==0,0],X_test[ y_test==0,1],\
         "xb",label="Test Neg")
plt.plot(X_test[y_test==1,0],X_test[ y_test==1,1],\
         "xr",label="Test Pos")
plt.xlabel("CBC")
plt.ylabel("PSA")
plt.legend()
plt.savefig("tree_train_test.png")
```



Installing new Python modules

For the next step, we need Python modules that are not part of Anaconda by default. To install them:



In terminal, type:

conda install graphviz

and then

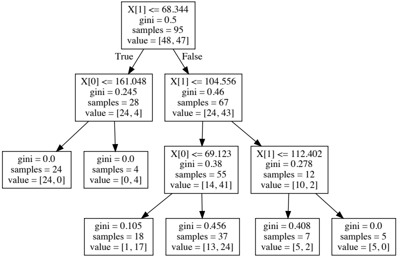
conda install python-graphviz

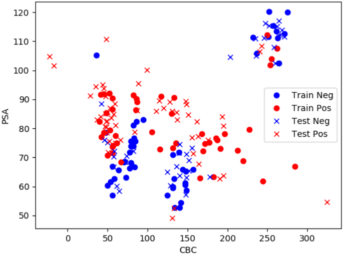
Creating a decision tree predictor

```
from sklearn import tree
import graphviz
# Create an object of class DecisionTreeClassifier
classifier = tree.DecisionTreeClassifier(max_depth=3)

# Build the tree
classifier.fit(X_train, y_train)

# Plot the tree
dot_data = tree.export_graphviz(classifier, out_file=None)
graph = graphviz.Source(dot_data)
graph.render("prostate_tree_depth3")
```


X[1] <= 68.344
gini = 0.5
samples = 95
value = [48, 47]
True
False
X[0] <= 161.048
gini = 0.245
samples = 28
value = [24, 4]
X[1] <= 104.556
gini = 0.46
samples = 67
value = [24, 43]
gini = 0.0
samples = 24
value = [24, 0]
gini = 0.0
samples = 4
value = [0, 4]
X[0] <= 69.123
gini = 0.38
samples = 55
value = [14, 41]
X[1] <= 112.402
gini = 0.278
samples = 12
value = [10, 2]
gini = 0.105
samples = 18
value = [1, 17]
gini = 0.456
samples = 37
value = [13, 24]
gini = 0.408
samples = 7
value = [5, 2]
gini = 0.0
samples = 5
value = [5, 0]


PSA
CBC
Train Neg
Train Pos
Test Neg
Test Pos
120
110
100
90
80
70
60
50
0
50
100
150
200
250
300

Using the trained predictor to make predictions

```python
from sklearn.metrics import confusion_matrix
predictions_train = classifier.predict(X_train)
predictions_test = classifier.predict(X_test)
print(predictions_test) # [1 1 0 1 1 0 1 0 ...]

# evaluate the predictions on the training set
conf_mat_train = confusion_matrix(y_train, predictions_train)
train_tn, train_fp, train_fn, train_tp = conf_mat_train.ravel()
print(conf_mat_train)
print("Sensitivity (train) =", train_tp/(train_tp+train_fn))
print("Specificity (train) =", train_tn/(train_tn+train_fp))
# [[34 14]
# [ 2 45]]
# Sensitivity (train) = 0.9574468085106383
# Specificity (train) = 0.7083333333333334

# evaluate the predictions on the test set
conf_mat_test = confusion_matrix(y_test, predictions_test)
test_tn, test_fp, test_fn, test_tp = conf_mat_test.ravel()
print(conf_mat_test)
print("Sensitivity (test) =", test_tp/(test_tp+test_fn))
print("Specificity (test) =", test_tn/(test_tn+test_fp))
# [[23 16]
# [ 6 50]]
# Sensitivity (test) = 0.8928571428571429
# Specificity (test) = 0.5897435897435898
```

Overfitting

There are big differences between the accuracies measured on the training and testing set:

Training:

	Pred Neg	Pred Pos
True Neg	46	2
True Pos	0	47

Testing:

	Pred Neg	Pred Pos
True Neg	27	12
True Pos	11	45

Predictor is much better on the training data than on the test data.
This is called *overfitting*.
Only the performance measured on the test data is representative of what we should expect on future examples.

More classifiers

Scikit-learn has a large number of different types of classifiers. See full list at:

https://scikit-learn.org/stable/supervised_learning.html

```
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)]

for model in models:
    print(type(model).__name__)
    model.fit(X_train, y_train)
    predictions_test = model.predict(X_test)
    conf_mat_test=confusion_matrix(y_test,predictions_test)
    test_tn,test_fp,test_fn,test_tp = conf_mat_test.ravel()
    print(conf_mat_test)
    print("Sensitivity(test) =",test_tp/(test_tp+test_fn))
    print("Specificity(test) =",test_tn/(test_tn+test_fp))
```

Conclusions

- Python + Scikit-learn allows easy use of many types of machine learning approaches for supervised learning.
- Accuracy of classification needs to be assessed using both sensitivity and specificity.
- Overfitting: Sens/Spec assessed on training set are generally overestimates of how the predictor will perform in new examples
- Sens/Spec assessed on test data (not used for training) are representative of accuracy that can be expected on new examples.