1 Using logistic regression to diagnose breast cancer

Here we will use the first of our machine learning algorithms to diagnose whether someone has a benign or malignant tumour. We are using a form of logistic regression. In common to many machine learning models it incorporates a regularisation term which sacrifices a little accuracy in predicting outcomes in the training set for improved accuracy in predicting the outcomes of patients not used in the training set.

We will use the Wisconsin Breast Cancer diagnosis data set, a classic 'toy' machine learning database. It is pre-loaded in scikit-learn, but is also available at:

`https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)`

The data set contains data on 562 patients, with 32 features for each patient.

Let's repeat what we've covered so far: loading data, splitting the data into training and test sets, and normalising the data. We'll print out the list of data fields and the patient classification outcomes.

Once you have had a look through this why not try changing the load data line to the iris data set we have seen before and see how the same code works there (where there are three possible outcomes).

Just replace the first line of the # Load dataset section with:

data_set = datasets.load_breast_cancer()

*A reminder that we are using the Anaconda distribution for scientific Python, which contains all the required packages, download from:

`https://www.anaconda.com/download/`

```
# import required modules

from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd
from sklearn.linear_model import LogisticRegression

# Load Dataset

data_set = datasets.load_breast_cancer()
X=data_set.data
y=data_set.target

# Show data fields
print ('Data fields data set:')
print (data_set.feature_names)

# Show classifications
print ('\nClassification outcomes:')
print (data_set.target_names)

# Create training and test data sets
X_train,X_test,y_train,y_test=train_test_split(
        X,y,test_size=0.25, random_state=0)

# Initialise a new scaling object for normalising input data
sc=StandardScaler()

# Set up the scaler just on the training set
sc.fit(X_train)

# Apply the scaler to the training and test sets
```

```
X_train_std=sc.transform(X_train)
X_test_std=sc.transform(X_test)

OUT:

Data fields data set:
['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']

Classification outcomes:
['malignant' 'benign']
```

Now let's see how easy it is to build and logistic regression model with scikit-learn. (We'll come back to that C=100 in a later section - that is the regularisation term which can help avoid over-fitting to the training data set and improve accuracy for predicting the outcomes for previously unseen data).

```
# Run logistic regression model from sklearn
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression(C=100,random_state=0)
lr.fit(X_train_std,y_train)
```

That's it! We can now use the model to predict outcomes for patients not yet seen.

```
y_pred=lr.predict(X_test_std)
```

Let's calculate a simple accuracy score (in the next module we'll look at a range of accuracy scores). This is simply the percentage of test cases correctly predicted.

```
correct = (y_test == y_pred).sum()
incorrect = (y_test != y_pred).sum()
accuracy = correct / (correct + incorrect) * 100

print('\nPercent Accuracy: %0.1f' %accuracy)

OUT:

Percent Accuracy: 93.7
```

That's not bad for our first model. We can look at the individual patients in more detail:

```
# Show more detailed results

prediction = pd.DataFrame()
prediction['actual'] = data_set.target_names[y_test]
prediction['predicted'] = data_set.target_names[y_pred]
prediction['correct'] = prediction['actual'] == prediction['predicted']

print ('\nDetailed results for first 20 tests:')
print (prediction.head(20))

OUT:

Detailed results for first 20 tests:
       actual  predicted  correct
0   malignant  malignant     True
```

```
1       benign     benign     True
2       benign     benign     True
3       benign     benign     True
4       benign     benign     True
5       benign     benign     True
6       benign     benign     True
7       benign     benign     True
8       benign     benign     True
9       benign     benign     True
10      benign     benign     True
11      benign     benign     True
12      benign     benign     True
13      benign  malignant    False
14      benign     benign     True
15  malignant  malignant     True
16      benign     benign     True
17  malignant  malignant     True
18  malignant  malignant     True
19  malignant  malignant     True
```

There we have our first machine learning model!