1 Adding standard diagnostic performance metrics to a ml diagnosis model

Here we will repeat the logistic regression model on cancer diagnosis, but add in commonly used measures of clinical diagnostic test performance. These may be used when the predicted outcome may be classified as true or false.

In this example we'll break down our code into functions. This is good practice for longer programmes. The functions are usually presented in alphabetical order. The main code comes after all functions have been defined. This makes the flow of the programme, as defined in the main code (not by the order of the functions), easy to follow. As a general rule functions should be named by their action. Names should adequately describe everything the function does (the best functional programming has functions that do one thing).

The diagnostic measures covered are:

1) accuracy: proportion of test results that are correct

2) sensitivity: proportion of true +ve identified

3) specificity: proportion of true -ve identified

4) positive likelihood: increased probability of true +ve if test +ve

5) negative likelihood: reduced probability of true +ve if test -ve

6) false positive rate: proportion of false +ves in true -ve patients

7) false negative rate: proportion of false -ves in true +ve patients

8) positive predictive value: chance of true +ve if test +ve

9) negative predictive value: chance of true -ve if test -ve

10) precision = positive predictive value

11) recall = sensitivity

12) f1 = (2 * precision * recall) / (precision + recall)

Let's look at the code, and run it:

```
# import required modules

from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np

def calculate_diagnostic_performance (actual_predicted):
    """ Calculate diagnostic performance.

    Takes a Numpy array of 1 and zero, two columns: actual and predicted

    Note that some statistics are repeats with different names
    (precision = positive_predictive_value and recall = sensitivity).
    Both names are returned

    Returns a dictionary of results:

    1) accuracy: proportion of test results that are correct
    2) sensitivity: proportion of true +ve identified
    3) specificity: proportion of true -ve identified
    4) positive likelihood: increased probability of true +ve if test +ve
    5) negative likelihood: reduced probability of true +ve if test -ve
```

```
    6) false positive rate: proportion of false +ves in true -ve patients
    7) false negative rate:  proportion of false -ves in true +ve patients
    8) positive predictive value: chance of true +ve if test +ve
    9) negative predictive value: chance of true -ve if test -ve
    10) precision = positive predictive value
    11) recall = sensitivity
    12) f1 = (2 * precision * recall) / (precision + recall)
    13) positive rate = rate of true +ve (not strictly a performance measure)
    """
    # Calculate results
    actual_positives = actual_predicted[:, 0] == 1
    actual_negatives = actual_predicted[:, 0] == 0
    test_positives = actual_predicted[:, 1] == 1
    test_negatives = actual_predicted[:, 1] == 0
    test_correct = actual_predicted[:, 0] == actual_predicted[:, 1]
    accuracy = np.average(test_correct)
    true_positives = actual_positives & test_positives
    true_negatives = actual_negatives & test_negatives
    sensitivity = np.sum(true_positives) / np.sum(actual_positives)
    specificity = np.sum(true_negatives) / np.sum(actual_negatives)
    positive_likelihood = sensitivity / (1 - specificity)
    negative_likelihood = (1 - sensitivity) / specificity
    false_positive_rate = 1 - specificity
    false_negative_rate = 1 - sensitivity
    positive_predictive_value = np.sum(true_positives) / np.sum(test_positives)
    negative_predictive_value = np.sum(true_negatives) / np.sum(test_negatives)
    precision = positive_predictive_value
    recall = sensitivity
    f1 = (2 * precision * recall) / (precision + recall)
    positive_rate = np.mean(actual_predicted[:,1])

    # Add results to dictionary
    performance = {}
    performance['accuracy'] = accuracy
    performance['sensitivity'] = sensitivity
    performance['specificity'] = specificity
    performance['positive_likelihood'] = positive_likelihood
    performance['negative_likelihood'] = negative_likelihood
    performance['false_positive_rate'] = false_positive_rate
    performance['false_negative_rate'] = false_negative_rate
    performance['positive_predictive_value'] = positive_predictive_value
    performance['negative_predictive_value'] = negative_predictive_value
    performance['precision'] = precision
    performance['recall'] = recall
    performance['f1'] = f1
    performance['positive_rate'] = positive_rate

    return performance

def load_data ():
    """Load the data set. Here we load the Breast Cancer Wisconsin (Diagnostic)
    Data Set. Data could be loaded from other sources though the structure
    should be compatible with thi sdata set, that is an object with the
    following attribtes:
        .data (holds feature data)
        .feature_names (holds feature titles)
        .target_names (holds outcome classification names)
        .target (holds classification as zero-based number)
```

```
        .DESCR (holds text-based description of data set)"""

    data_set = datasets.load_breast_cancer()
    return data_set

def normalise (X_train,X_test):
    """Normalise X data, so that training set has mean of zero and standard
    deviation of one"""

    # 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)
    return X_train_std, X_test_std


def print_diagnostic_results (performance):
    """Iterate through, and print, the performance metrics dictionary"""

    print('\nMachine learning diagnostic performance measures:')
    print('-------------------------------------------------')
    for key, value in performance.items():
        print (key,'= %0.3f' %value) # print 3 decimal places
    return

def split_data (data_set, split=0.25):
    """Extract X and y data from data_set object, and split into tarining and
    test data. Split defaults to 75% training, 25% test if not other value
    passed to function"""

    X=data_set.data
    y=data_set.target
    X_train,X_test,y_train,y_test=train_test_split(
        X,y,test_size=split, random_state=0)
    return X_train,X_test,y_train,y_test

def test_model(model, X, y):
    """Return predicted y given X (attributes)"""

    y_pred = model.predict(X)
    test_results = np.vstack((y, y_pred)).T
    return test_results

def train_model (X, y):
    """Train the model """

    from sklearn.linear_model import LogisticRegression
    model = LogisticRegression(C=100,random_state=0)
    model.fit(X, y)
    return model

###### Main code #######

# Load data
data_set = load_data()
```

```
# Split data into training and test sets
X_train,X_test,y_train,y_test = split_data(data_set, 0.25)

# Normalise data
X_train_std, X_test_std = normalise(X_train,X_test)

# Train model
model = train_model(X_train_std,y_train)

# Produce results for test set
test_results = test_model(model, X_test_std, y_test)

# Measure performance of test set predictions
performance = calculate_diagnostic_performance(test_results)

# Print performance metrics
print_diagnostic_results(performance)

OUT:


Machine learning diagnostic performance measures:
-------------------------------------------------
accuracy = 0.937
sensitivity = 0.933
specificity = 0.943
positive_likelihood = 16.489
negative_likelihood = 0.071
false_positive_rate = 0.057
false_negative_rate = 0.067
positive_predictive_value = 0.966
negative_predictive_value = 0.893
precision = 0.966
recall = 0.933
f1 = 0.949
positive_rate = 0.608
```