*6CCS3ML1 (Machine Learning)*

# *Practical 1: Machine Learning Metrics*

(Version 1.4)

## 1 Overview

This practical builds on Lecture 1. In that lecture we discussed how to measure the performance of machine learning methods. In this practical we will use these ideas to explore the effectiveness of a couple of different machine learning techniques. To do this, you will make use of `scikit-learn`, a Python library that includes several different kinds of learner. Both Python 3 and `scikit-learn` should be installed on your machines.

For any questions and/or issues on software, please see here: `https://techwiki.nms.kcl.ac.uk`.

**Note:** If you are not so fluent in Python, you may well find that this exercise takes substantially longer than the practical session. If that is the case, I would urge you to complete the exercise after the lab, at least as far as Section 5.

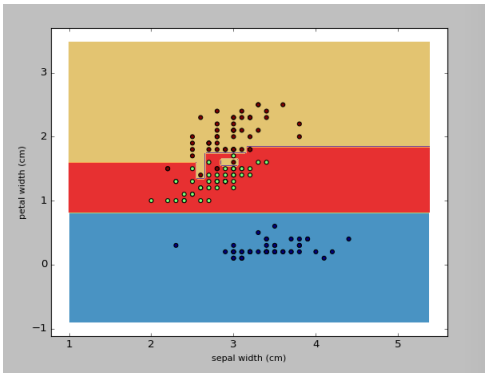
## 2 Decision trees

Decision trees are a simple form of classifier that we will cover in Lecture 2. We will start by using a decision tree on the Iris dataset.

Download

`classify-iris-simple.py`

from KEATS and run it. You should get output that looks like this:




This is the same dataset you saw in the lecture, pictured slightly differently. What you see are the points from the Iris dataset, where the background colours identify the classes into which other examples would be classified. In other words, any iris that had petal and sepal widths that put it in the blue region would be classified as *setosa*. What we see in this picture are the *decision boundaries*.

Does this decision tree look as though it fits the data well, or does it overfit? Why do you think this?

Take a look at the code. We start with:

```
iris = load_iris()
X = iris.data [:, [1, 3]]
y = iris.target
```

which loads the dataset, and pulls out two parts of it, the `data`, and the `target`. The `data` are the attribute values we will cluster. The `target` holds the class labels. Note that we only use two attribute values (those numbered 1 and 3, which are, of course the second and fourth in the list of values). We will explain why we only want two later.

The line:

```
iris_tree = tree.DecisionTreeClassifier(criterion="entropy").fit(X, y)
```

creates a decision tree, and learns how to classify the data. `fit(X, y)` is the method that does the learning. Running learning is often very simple in `scikit-learn`.

The next section of the code uses the classifier to create the background colours you see in the plot. The part where the classifier is used is here:

```
Z = iris_tree.predict(np.c_[xx.ravel(), yy.ravel()])
```

with `predict` being the method that predicts which class a new point — the argument of `predict` — will be in. The rest of this section of the code runs all the points in the background space through the classifier, and then paints the background of the plot the appropriate colour.

The last important bit is:

```
plt.scatter(X[:, 0], X[:, 1], c=y.astype(np.float))
```

This plots the training examples in a scatter plot.

The plot is the reason we restricted ourselves to just two features. With more features, we could not produce a simple two-dimensional plot. However, a tree which uses more features might well produce a more accurate classifier.

A cleverer version of the code can be found in:

`classify-iris.py`

This takes all possible pairs of features in the Iris dataset, does what `classify-iris-simple.py` does for each pair, and then plots all the results in one figure.

Now, try to do the same as `classify-iris-simple.py` does for a different dataset. We will use the Wisconsin Breast Cancer dataset. Like the Iris dataset, this is a standard set, but has a lot more features, and is a binary dataset. The fact it is binary makes some things easier.

You need to use:

```
from sklearn.datasets import load_breast_cancer
bc = load_breast_cancer()
```

to load the dataset into the variable `bc`. Given this, you should be able to build a decision tree for the breast cancer and then plot the decision boundaries. You may need to explore several sets of features to get a set that produces a good plot of the decision boundaries. (You might even want to use the method in `classify-iris.py` to try all pairs).

## 3 Cross-validation

The previous section had you build a decision tree from all the data in a dataset. After the last lecture, we know that this is not the right thing to do. Instead we want to use some of the data for training and some for testing. `scikit-learn` provides an easy way to do this.

The code:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    my_data, my_target, test_size=0.2, random_state=0)
```

will randomly split `my_data` and `my_target` so that 80% is in `X_train` and `y_train`, and 20% is in `X_test` and `y_test`. In addition:

```
dtree.score(X_test, y_test)
```

will produce an accuracy score for the decision tree `dtree` on the test data `X_test`, `y_test`.

Given this information, do the following:

1. Split the breast cancer data so that you use 90% for training and 10% for testing. What is the accuracy of the tree that you construct?

   Make sure that you use all the features in the dataset.

2. Now find the average score when you build and test 10 such decision trees. And compute the standard deviation of the scores so that you can report the standard error in the average.

3. Now try out the cross-validation function in `scikit-learn`. Here it is being used to run a 5-fold cross-validation on `data` and `target`.

   ```
   from sklearn import metrics
   cv_scores = cross_val_score(dtree, data, target, cv=5)
   print cv_scores.mean()
   print cv_scores.std()
   ```

   `dtree` is a decision tree object which is instantiated, in turn, with the result of each tree learnt from each fold.

   Run a 10-fold cross-validation on the breast cancer data. How do the average accuracy and standard deviation compare with the results you obtained earlier with the 10 trees from randomly assigned data?

4. Since this is not a module in using `scikit-learn` but a module in machine learning techniques, you should also write your own code to do the 10-fold cross-validation. For now you don't need to figure out the score for yourself (we'll get to that), just use the `score` method. Use the result of the previous question to check your result.

# 4 Metrics

As we know from the lecture, accuracy (what `score` provides) is not the only metric that is useful in machine learning. In this part of the practical you will compute many of the other metrics we discussed in the lecture. As above, the idea is not that you just use the `sklearn` functions, but that you write your own code to do this. This will do much more to help you understand (and learn) the concepts. It will also help your Python programming if that is something that you are not yet fluent in.

You should:

1. Compute the confusion matrix of a breast cancer decision tree. Start with last one you learnt for the cross-validation exercise. You should have a tree, call it `yr_tree`, a test set `X_test` and a set of true labels for the test data, `y_test`. The simplest way to compute the confusion matrix is to run through `X_test`, example by example, feeding the example into the classifier and comparing the result with the corresponding element of `y_test`. Depending on the result, you will have a true positive, a true negative, a false positive or a false negative. Running through the whole test set will give you the numbers you need for a confusion matrix.

2. Compute the precision of a decision tree. This is easy now that you have the true positives and false positives.

3. Compute the recall of a decision tree.

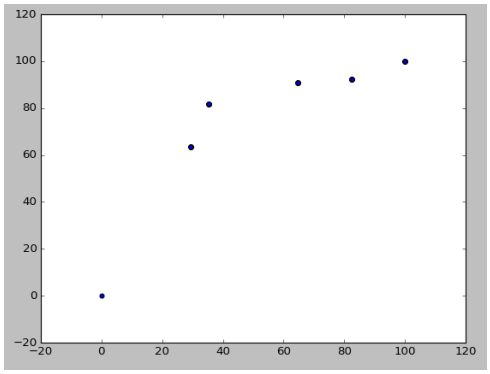
4. Compute the F1 score of a decision tree.

5. Now you can compute the mean and standard error of the precision and recall of all the decision trees that you learn during 10-fold cross-validation.

6. Compute the ROC curve of a breast cancer classifier. The Breast cancer decision trees don't give much of an ROC curve because they only make definite class predictions (so adjusting the threshold does nothing). So, we will use a $k$-nearest neighbour classifier, just like the one we discussed in the lecture:

   ```
   from sklearn.neighbors import KNeighborsClassifier
   my_neigh = KNeighborsClassifier(n_neighbors=5)
   ```

   Now, once you have trained this, if you use the method `predict_proba` rather than `predict` you will get the probability of the example being in each of the two classes. Using a threshold, you can decide what the right classification is, and compute a true positive, false positive pair. Varying the threshold between 0 and 1 will give you all the data you need for an ROC curve. To get the curve, plot the pairs of true and false positives as a scatter plot. There will not be many points, but it should be roughly the same shape as the one we saw in the lecture. I got one that looks like this:




If you want an ROC curve with axes like those above, normalise the values. If you want an ROC curve that has axes that only run from 0 to 100 (unlike the one above) you can fix that too. You could make your output even more like a standard ROC curve by connecting the points.

Of course, now that you have code to count true positives, true negatives, false positives and false negatives, you can easily compute accuracy as well as the metrics you computed above.

## 5 Nearest Neighbour

Now that you have written code to do cross-validation, and code to compute metrics, it should be simple to go back and do these on the $k$-nearest neighbour classifier. In particular:

1. Run a 10-fold cross-validation on the $k$-nearest neighbour classifier applied to the breast cancer dataset.
2. Compute the average precision, recall and F1 score across the 10-fold cross-validation.

   That is, compute precision, recall and F1 score for each fold, and compute the average and standard deviation over the 10 values.

This completes the compulsory part of the practical.

## 6 More

If you want to try other things, pick one or more of the following. Note that the last challenges you to write your own classifier.

1. Repeat the exercises from Section 4 using the `scikit-learn` methods for all of these. You can find the methods by looking at the API documentation:

   `http://scikit-learn.org/stable/modules/classes.html`

   Look under `sklearn.metrics`
2. Compute precision, recall, F1 and ROC curve for classifiers run on the Iris dataset.

   Because the Iris dataset has 3 classes, you may (depending on how you wrote it) need to rethink your code that counts false positives etc.

3. Write a k-Nearest Neighbour classifier and apply it to the Iris dataset. You should find it works rather well. The 1-NN classifier that I wrote has an accuracy of of 93% when trained on 80% of the data, and a 5-NN classifier has 96% accuracy.

# 7 Version list

- Version 1.0, January 14th 2020.
- Version 1.1, January 11th 2021.
- Version 1.2, January 11th 2022.
- Version 1.3, January 4th 2023.
- Version 1.4, January 8th 2024.