School of Computing and Information Systems
The University of Melbourne
COMP30027
MACHINE LEARNING (Semester 1, 2019)

Practical exercises: Week 4

Today, we will be using `scikit-learn` to classify some data, and to evaluate some classifiers.

1. Load the *Iris* dataset[1] as follows:

```
>>> from sklearn import datasets
>>> iris = datasets.load_iris()
>>> iris
```

   (a) Identify the contents of the complex data type `iris`, for example `iris.DESCR` contains a long description of the dataset, which you can `print()`.

   (b) The common terminology in `scikit-learn` is that the array defining the attribute values is called `X` and the array defining the gold–standard ("ground truth") labels is called `y`; create these variables for the *Iris* data.

   (c) Confirm that `X` is a 2-dimensional array, with a row for each instance and a column for each attribute. (Hint: read[2] about the `shape` property in `numpy`.)

2. Let's build a 0-R classifier ("majority class classifier"). In `scikit-learn`, this is a `DummyClassifier`[3]:

```
>>> from sklearn.dummy import DummyClassifier
>>> zero_r = DummyClassifier(strategy='most_frequent')
>>> zero_r.fit(X, y)
```

   (a) Confirm that this is a typical 0-R classifier by checking its predictions on the training data: `zero_r.predict(X)` — which class has it chosen?

   (b) The default evaluation metric associated with a `DummyClassifier` is accuracy, which you can observe using `score()`, for example: `zero_r.score(X, y)`
   This strategy — building a model, and then evaluating on the data that we used to build the model — gives us something called "training accuracy", and is generally frowned upon in the Machine Learning community. Why do you suppose this is? (We'll examine some better techniques later.)

   (c) Contrast the 0-R classifier with the "weighted random classifier", which makes random predictions according to the distribution of classes in the training data; `(strategy='stratified')` — check its predictions, and evaluate its training accuracy. Does it have a higher accuracy, on average, than 0-R, or a lower accuracy? (You should run `score()` at least 10 times.)

3. Let's consider a couple of other classifiers: a Decision Tree, and 1-R[4] (which is really just a limited `DecisionTreeClassifier` in `scikit-learn`):

```
>>> from sklearn.tree import DecisionTreeClassifier
>>> one_r = DecisionTreeClassifier(max_depth=1)
>>> one_r.fit(X, y)
>>> dt = DecisionTreeClassifier(max_depth=None)
>>> dt.fit(X, y)
```

[1] Note that there are some small differences between this dataset and the one we were looking at last week, most notably, the errors that we needed to fix are not present.

[2] This is worth doing, because one day you might want to re-shape a `numpy` array.

[3] `scikit-learn` uses this terminology to help remind you not to use these sorts of classifiers when trying to solve real problems; they are easy **baseline** classifiers, however.

[4] Note that this `scikit-learn` implementation is slightly different to the lecture version, because it doesn't count errors — rather it uses the Gini coefficient or the Information Gain to determine the best attribute.

(a) Find the training accuracy of the two classifiers.

(b) The `feature_importances_` attribute is adequate for completely describing the 1-R classifier. Which attribute is being used to classify the data?

(c) (Harder) Check the predicted labels for each instance to discern the values for this attribute that each class maps to.

(d) The default splitting criterion for these Decision Trees is the **Gini coefficient**. Read up on the difference between this and the **Information Gain** — do you expect the behaviour of this model to change by using the alternative splitting criterion? Try it, and confirm your expectations.

4. A better mechanism for evaluating a classifier is based on randomly partitioning the data into a training set and test set (the "holdout" method). There is an in-built utility for this in `scikit-learn`, but it can be in one of two places:

```
>>> from sklearn.model_selection import train_test_split # Newer versions
>>> from sklearn.cross_validation import train_test_split # Older versions
>>> X_train, X_test, y_train, y_test = train_test_split(X, y)
```

(a) Train the three classifiers (0-R, 1-R, Decision Tree) on the training data, rather than the full data set. `score()` is too specific to be used in most situations; another way to find the training accuracy is by comparing the predictions to the gold-standard labels as follows:

```
>>> from sklearn.metrics import accuracy_score
>>> accuracy_score(zero_R.predict(X_train),y_train))
```

Calculate the training accuracy of the classifiers on the held-out training data. How does it compare to the training accuracies you calculated before? Why is this?

(b) Instead of calculating the accuracy with respect to the training set, train your classifiers on the training data (using `fit()`) and then evaluate them (by calculating accuracy) according to their predictions on the test data. How different are the training accuracies and test accuracies? Hypothesise what could be causing these differences.

(c) By default, `train_test_split` uses 75% of the data as training, and 25% as test. This can be changed by passing an argument, for example, `test_size=0.5` means that we use 50% as training[5] and 50% as test. Try some different values (perhaps multiple times) to see if you can observe the **trade-off** inherent in the model using this evaluation strategy.

5. (Stratified) M–fold cross-validation is so popular, `scikit-learn` has a utility that applies it directly[6]. For example, 10–fold cross-validation of the 0-R classifier proceeds as follows:

```
>>> from sklearn.model_selection import cross_val_score # Newer versions
>>> from sklearn.cross_validation import cross_val_score # Older versions
>>> cross_val_score(zero_R, X, y, cv=10)
```

(a) This method returns an array of the calculated evaluation metric (by default, accuracy) across the folds. Write a wrapper function which averages these values, so as to come up with a single score for the classifier.

(b) How does the estimate of the accuracy of the various classifiers using cross-validation compare to the training accuracies and holdout accuracies you calculated above?

[5]The default behaviour is that the remainder of the data is used as training; this too can be altered, if you wish.

[6]There are also simpler methods like `StratifiedKFold()` to generate the partitions, which you can then use to train and test the model yourself, if you wish.