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

Practical exercises: Week 6

Today, we will expect you to be referring to the API[1] for `scikit-learn` (`http://scikit-learn.org/stable/modules/classes.html`) — you should also refer to previous weeks' exercises where necessary.

1. We will use the *Car Evaluation* dataset from the UCI Machine Learning Repository (`https://archive.ics.uci.edu/ml/machine-learning-databases/car/car.data`).

   (a) Load the data into a suitable format for `scikit-learn`, for example:

   ```
   >>> for line in f:
   ...     atts = line[:-1].split(",")
   ...     X.append(atts[:-1])
   ...     y.append(atts[-1])
   ```

   (b) How many instances are there in this collection? How many attributes, and of what type(s)? What is the class we're trying to predict, and how many values does it take?

   (c) Are there any missing attribute values? Is there any evidence that this is an artificially–constructed dataset?

   (d) What happens if we try to build a classifier (using `fit()`) using this data?

2. Unfortunately, `scikit-learn` isn't set up to deal with our attributes in this format.

   (a) Write some functions that transform our **categorical** attributes into **numerical** attributes, by (perhaps arbitrarily) assigning each categorical value to an integer, for example:

   ```
   >>> def convert_class(raw):
   ...   if raw=="unacc": return 0
   ...   elif raw=="acc": return 1
   ...   elif raw=="good": return 2
   ...   elif raw=="vgood": return 3
   ```

   (b) Load the dataset again, this time as integers. Observe that we can actually build a model using this data.

   (c) Split the data into training and test sets[2].

3. Read up on different implementations of the **Naive Bayes** classifier in `sklearn.naive_bayes`. Which one do you think is most suitable for the dataset we have?

   (a) Train the (default) Naive Bayes model and determine its accuracy on the held–out test set.

   (b) Compare the accuracies of all three different kinds of Naive Bayes classifier (perhaps on a few different train–test splits). Does this accord with your expectations?

   (c) By default, this implementation of Naive Bayes uses **Laplace smoothing**. Turn this off, and see what happens — what is the significance of the reported accuracy?

   (d) What happens if you increase the smoothing parameter instead? Calculate the accuracy for a range of values from 5 to 500. For the very large values, examine the predicted classes for the test instances — what is happening?

[1] Note that there are some small differences between the version installed in the labs and the latest stable version.

[2] You should probably get in the habit of converting to `numpy` arrays, even though `scikit-learn` will often do this for you.

4. The transformation of the data in Q2 implicitly creates ordinal attributes. At first glance, such a strategy does seem reasonable in light of the given values (such as `small`, `med`, `big`).
A different strategy would be to replace each categorical attribute having m values with m **binary** attributes, in `scikit-learn` this uses the `OneHotEncoder`[3]:

```
>>> from sklearn.preprocessing import OneHotEncoder
>>> ohe = OneHotEncoder()
>>> ohe.fit(X)
>>> X_trans = ohe.transform(X).toarray()
```

Note that this transformation should be done before we split the data into training and test sets. (Why?)

   (a) Check the shape of `X_trans` — how many attributes do we have now? Does this correspond to your expectations?
   (b) Split the dataset comprised of one–hot attributes into train and test sets. Compare the accuracies of the three Naive Bayes models using ordinal attributes with the three models using one–hot attributes: are you surprised? What can we infer?

5. Recall that we built a `DecisionTreeClassifier` in Week 4.

   (a) Do you think the dataset comprised of ordinal attributes, or the dataset comprised of one–hot attributes would be more appropriate for a typical **Decision Tree**? Check the test accuracy of the default Decision Tree model built on each of these datasets.
   (b) How does the accuracy of the Decision Tree models compare with Naive Bayes on these datasets? Why might this be?
   (c) (n.b. This step might not be possible in the labs.) The main strategy for visualising a Decision Tree in `scikit-learn` is through the `export_graphviz()` method. Read up on the method, and explore whether the trees created in the previous question are indeed different.
   (d) Try altering the value of the `max_depth` parameter, between 1 and `None`[4]. Visualise the resulting trees, if you can. Compare the estimated training and test accuracies; is there any evidence that the algorithm is over–fitting (or under–fitting) for trees of certain depths?

6. The **filtering** approach to Feature Selection in `scikit-learn` can be done using `SelectKBest`, for example:

```
from sklearn.feature_selection import SelectKBest, chi2
x2 = SelectKBest(chi2, k=3)
x2.fit(X_train,y_train)          # these two statements can be combined into
X_train = x2.transform(X_train) # a single statement via fit_transform()
X_test = x2.transform(X_test)
```

   (a) What happens to the `shape` of `X_train` and `X_test` now?
   (b) Find out what the best features[5] were for your dataset, according to χ^2:

```
for feat_num in x2.get_support(indices=True):
    print(feat_num)
```

   (c) We'll be re-visiting these ideas again — on more interesting data — in later weeks.

[3] Note that the previous step, of replacing the strings with integers, is a necessary intermediate step for using the `OneHotEncoder`.
[4] It is perhaps not–so–obvious how deep the tree can possibly become, depending on the dataset that we are examining!
[5] Remember, everything is a number, so we need to do some work to get the name back.