ME4 Machine Learning - Tutorial 9

Lecture 9 introduced unsupervised learning, and the concepts of clustering and principal component analysis. This tutorial will give you practice with both of these methods, using Scikit-learn. For clustering you will test out what happens as you vary the options for the KMeans algorithm we looked at in the lecture. You can also test out some of the other tools available in the library, which is straightforward to do because the interface is the same for all. In principal component analysis you will test whether you can use the method to identify the key directions in a dataset, implementing the theory from the course.

Here we generate some random distributions; to keep consistent results, set the random seed to a constant number as discussed in earlier tutorials.

Note: this tutorial has parts which must be submitted as coursework. While for most of the tutorial content you are encouraged to collaborate and learn from each other, sharing answers to these assessed sections will constitute plagiarism and be treated accordingly. The tutors have been instructed not to provide help with these questions, however, you may ask generic questions to help your overall understanding.

1 Clustering

We will firstly generate some test data with the make_blobs() function in sklearn.datasets

```
from sklearn.datasets import make_blobs
...
X, y = make_blobs(n_samples=n_samples, centers=6)
```

This will generate 6 blobs. Set n_samples to 1500 in this.

We will apply the k-means algorithm, discussed in the lecture, to this dataset. This can be applied as

```
from sklearn.cluster import KMeans
...
km = KMeans(n_clusters=6)
```

The fit_predict() function will then predict what the clusters are. Produce a scatter plot of the data, with the predicted clusters in different colours (use 'c = ...' to set the colour to a particular category for each point). Produce a plot of the true clusters too (given in the variable y, output from make_blobs above) and compare. How well do you think the clustering algorithm performs? You can control the standard deviation of each blob with the 'cluster_std' argument in make_blobs (default is 1.0) -

how do you think you could make it more challenging to separate? Try this and see what the result is. Increase and decrease the number of clusters (vary both at the generation and fitting stage) and compare the results.

The centres (means) of the clusters will be stored in

```
km.cluster_centers_
```

Check that these are correct. You can also use

```
km.inertia_
```

to obtain the sum of the squared distances of samples to their closest cluster centre. How might you change the data to get a different result out from this?

Try other clustering routines, such as AffinityPropagation() and DBSCAN() in sklearn.cluster, and plot the results. Try making the data more clustered by reducing the standard deviation of each cluster, with cluster_std=0.5 in make_blobs(); this should help these algorithms. AffinityPropagation() may also converge better for a smaller number of points, e.g. 500.

2 Principal component analysis

We will now perform principal component analysis on a dataset. First of all generate 500 samples from a 2D normal distribution centred at (0, 0) with a standard deviation of 0.3 and 0.1 in x and y respectively, which is then rotated around by 23 degrees (you will need to use the get_cov() function we developed in earlier tutorials, along with np.random.multivariate_normal()). Plot the distribution and set the axes equal.

Then we can perform a principal component analysis. We use the following to fit the data:

```
from sklearn.decomposition import PCA
...
pca = PCA(n_components=2)
pca.fit(X)
```

The attribute pca.components_ contains the components. Plot lines showing the first and second component directions on the scatter graph in red and green respectively, starting from position (0, 0) each time. Calculate the angle of the first principal component from the values given (note that this may be out by 180 degrees – you can either account for this manually or the np.mod() function may be useful.). Is there a small discrepancy between this angle and the input 23 degrees? What methods do you think you could improve the accuracy of this number? There are two main ones you should be able to come up with. See if you can implement the improvement in practice. Note that this is not specifically an improvement to the method implementation, just the data being provided.

3 Submission

Consider that you are now a student on a Mechatronics course. The lecturer has given you a very complex electronic circuit to analyse, which has two outputs, y_1 and y_2 and two inputs x_1 and x_2.

The inputs are each continuous voltages between 0 and 10v, and you have been told that the system is stateless, i.e. the outputs have no dependence on what happened before.

You know that the output voltages are intended to indicate one of three states for the various inputs. Each of the three states is indicated by an (unknown) unique combination of the two output voltages (although note that there is noise in this). Analyse the training data provided (in [volts.csv]) to find out what you expect these voltages to be – submit the three pairs of values as part (A) – an example of the form of these three pairs could be [1.30V 5.27V], [1.97V 7.52V], [8.32V 1.89V], with each pair corresponding to a particular class. Then make a new output variable y which indicates the class based on the continuous voltages provided (which number you assign to which voltage combination is not important).

We want to train a classifier so that we can predict general output for this system. First use a random forest with 100 trees each and a maximum depth of 20. Plot the resulting decision function as a contour plot as used throughout the course, with the colour indicating the class as a function of the two inputs x_1 and x_2 in the range 0 to 10v. This is submission part (B).

Then utilise a nearest-neighbour approach (a single neighbour should be considered) to classify across the same domain. This is submission part (C).

For part (D), answer the following questions concisely (no more than 1-2 sentences each, and ideally just a few words):

1. Which method do you think performs best and why? Also indicate what you mean by 'performs best' too.

2. If the outputs did not fall into discrete classes, how could you have modelled the relationship between the inputs and outputs instead? Could this also be done in this case even though the outputs do fall into classes?

Submit the following parts: numerical values for the voltages (A), your decision function image for random forest (B), your decision function with nearest neighbour (C) and your answers for part (D). All responses should be marked with the labels (A)-(D). Place these all into a single PDF document containing your name in the filename, and your Imperial username (e.g. xyz117 -- not your CID) at the top of the first page, and submit this. Note that we reserve the right to award zero marks to any coursework not following the instructions given (e.g. submission of Word documents). Also note that short answers are explicitly requested and longer answers may lose marks. You should not submit code.