Practical 3: K-means

(Version 1.1)

1 Overview

This practical builds on Lecture 3. In that lecture we discussed several learning mechanisms that are based on probability theory. Here you will program one of these — K-means. (We will get to Naive Bayes next week.)

Once again you will make use of `scikit-learn`.

2 K-means for the iris data

Start by downloading and running `k-means-starter.py`. This is very simple, and just loads and plots the Iris data (yet again).

Your job is to implement the K-means algorithm that we covered in Lecture 3.

You can assume that there are 3 clusters, and to simplify things you should consider only the two features that `k-means-starter.py` extracts from the dataset — the values that are loaded into the variable `X`.

To select cluster centres, you can pick random values based on the range of values of those features. The code in `k-means-starter.py` extracts the max and min values of these features here:

```
x0_min, x0_max = X[:, 0].min(), X[:, 0].max()
x1_min, x1_max = X[:, 1].min(), X[:, 1].max()
```

After that, you just need to follow the update rules in the K-means algorithm to find the clusters.

3 Plotting results

Once you have a clustering, you should be able to plot it in the same way that the correct Iris clusters are plotted in `k-means-starter.py`.

Plot the two clusterings side by side.

How does yours compare?

Are there any obvious cases that are in the wrong cluster?

4 Measuring performance

As we discussed in Lecture 1, there are metrics for clustering. One that is appropriate here is the Rand index. This compares two clusterings and reports how similar they are. Since we have the ground truth data for the Iris dataset (which cluster each example really is in) we can use the Rand index to tell us how good a job our K-means is doing.

How good is your implementation? Mine rated 0.797.

Documentation on the `scikit-learn` implementation of the Rand index can be found at:

> `http://scikit-learn.org/stable/modules/generated/sklearn.metrics.adjusted_rand_score.html`

5 Comparing with `scikit-learn`

`scikit-learn` has an implementation of K-means, `sklearn.cluster.KMeans`. In fact, we have seen it before in Lab 1.

Use this version of K-means to cluster the Iris data.

You can consult the documentation on `sklearn.cluster.KMeans` at:

> `http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html`

or you can go back and look at the instructions for Lab 1.

Now, compare `sklearn.cluster.KMeans` to your version, both by plotting the results, and by using the Rand index. Which is better?

6 More

If you want to try other things:

1. One of the weaknesses of K-means is that the results can be very dependent on the initial choice of cluster centres. Random selection can therefore give poor clusters. The `kmeans++` approach picks cluster centres as follows. Pick the first center at random. Then pick subsequent points with a probability that is proportional to their squared distance to the center that is nearest to them.

   Implement this approach. A quick and dirty way to do this is to create a list of potential centres (picking integer value only), then assign probabilities to these points based on their squared distance from the closest center, and then sample from this list.

   To see any improvement, you will probably need more difficult data than the Iris data (which clusters quite easily). Use the `scikit-learn` function `sklearn.datasets.make_blobs` to generate data. (Look at the documentation for the function to find out how to use it.)

2. If you didn't already do this, generalize your code to produce a solution that works for any number of clusters (that is, make the number of clusters a parameter).

   You will need to use `sklearn.datasets.make_blobs` to generate data to test your code.

7 Version list

- Version 1.0, January 28th 2020.

- Version 1.1, January 11th 2021.