Worksheet 3 Week 15

February 13, 2021

1 Introduction

This week we will look at some unsupervised clustering algorithms. In this worksheet, we will start off by implementing k-means from scratch. We go on to look at using elbow plots to select a good value of k.

We go on to complare the behaviour of k-means with hierarchical clustering and Gaussian mixture models.

NB: if you find implementing the k-means algorithm form scratch challenging, **don't worry**! Have a go, but the rest of the worksheet is not dependent on doing this, so you can skip that question and come back to it when you finish the rest of the worksheet.

2 K-means clustering

In the lecture, we saw that k-means is an unsupervised clustering algorithm. Recall that the algorithm runs as follows:

Given a set of datapoints drawn from $\Omega = \mathbb{R}^n$:

1. Randomly partition the set of datapoints into k sets.
2. For each set P calculate its mean vector:
$$\hat{x}_P = \left(\frac{\sum_{\vec{x} \in P} x_1}{|P|}, \ldots, \frac{\sum_{\vec{x} \in P} x_i}{|P|} \ldots, \frac{\sum_{\vec{x} \in P} x_n}{|P|} \right)$$
3. For each datapoint evaluate the squared Euclidean distance from each of the mean vectors e.g. $||\vec{x} - \hat{x}_P||^2$. Reallocate the datapoint to the partition set the mean of which it is closest to.
4. If the partition sets remain unchanged then stop. Else go to 2.

2.1 Implementing k-means

[ ]:
```
# The following code creates some artificial data and plots it
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
import numpy as np

X, y = make_blobs(centers=3,n_samples=100, cluster_std=2, random_state=100)
```

```
fig, ax = plt.subplots()
ax.scatter(X[:,0], X[:,1])
```

Implement a function `kmeans` that takes a value k and the data X, clusters the data, and returns the centroids and the labels of the data

[ ]:
```
def kmeans(k, X):
    # randomly assign labels to the data
    ##TODO##

    # set up a while loop that will run until the data labels no longer change
    while True:
        # Calculate the centroids of the data
        ##TODO##

        # For each datapoint:
        for i, x in enumerate(X):
            #Calculate the squared Euclidean distance to each centroid
            ##TODO##

            # Assign new labels based on distance to the centroid
            ##TODO##

        # If all the new labels are equal to the old labels,
        # break out of the while loop
        ##TODO##

        # Assign the values of the new labels to the variable labels
        ##TODO##

    # return the centres and the labels.
    return centres, labels

# Plot the centroids on the data. Are they as you would expect?
centres, labels = kmeans(3, X)
ax.scatter(centres[:,0],centres[:,1])
fig
```

2.2 Using the k-means function from scikit-learn

Scikit-learn has k-means built in. We import it using the command `from sklearn.cluster import KMeans`. Look at the documentation for KMeans (https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#). The KMeans estimator has 4 attributes. What are they?

The attributes are: 1. 2. 3. 4.

Which attribute would you use if you wanted to look at the labels assigned to the datapoints? What if you wanted to look at the centroids? What would you use the attribute `inertia_` for?

2.2.1 Generating elbow plots

We will run k means over our toy dataset for multiple values of k and generate an elbow plot. To do this we can use the attribute `inertia_`. This attribute measures the within-cluster sum of squares, or the variance of each cluster, and the k means algorithm works to minimize this quantity. The within-cluster sum of squares is defined as:

$$\sum_{j=1}^{k} \sum_{x \in P_j} ||x - \mu_j||^2$$

To generate the elbow plot, we run k means for values of k from 1 to 10, and plot the inertia at each point. If there is a clear 'elbow' in the plot, then this gives us the optimal value of k. Do you see a clear 'elbow' in the plot? If so, what is the optimal value of k?

[ ]:
```
# Import KMeans from sklearn.cluster
##TODO##

# Optional: write your own function to calculate the inertia
# (otherwise you can just use the attribute inertia_)
def inertia(X, labels, centroids):
    ##TODO## (Optional)

# Set up a variable to store the inertias
inertias = []

# Loop over values of k from 1 to 10
for k in range(1, K+1):
    # Instantiate the KMeans class with k clusters
    ##TODO##

    # Fit the model to the data
    ##TODO##

    # Store the value of the inertia for this value of k
    ##TODO##

# Plot the elbow
plt.figure()
plt.plot(range(1, K+1), inertias, 'bx-')
plt.xlabel('k')
plt.ylabel('Inertia')
plt.title('The elbow method showing the optimal k')
```

3 Clustering the iris dataset

The Iris flower data set or Fisher's Iris data set is a multivariate data set introduced by the British statistician and biologist Ronald Fisher in 1936. The data set consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). There are four features corresponding to the length and the width of the sepals and petals, in centimetres. Typically, the Iris data set is used as a classification problem, but by considering only the 4-D input feature space we can also apply clustering algorithms to it.

[ ]:
```
# Import the iris dataset, and save the data into a variable X
# (take a look at the documentation here:
# https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.
 ↪html)
##TODO##
```

Let's begin by assuming that since there are 3 types of iris, then there may be 3 clusters. Instantiate a k-means classifier with 3 clusters, and fit it to the data. Print out the centroids. You can visualise the resulting clusters by generating scatter plots projected on 2 dimensions. Try generating scatter plots for various combinations of features.

Extra question Generate one large plot with subplots for each combination of features.

[ ]:
```
# Fit the iris dataset
##TODO##

# Make a scatter plot of the data on the first two axes
# Experiment with looking at different axes
##TODO##
```

[ ]:
```
# Optional: Plot all combinations of data in one large plot with subplots for␣
 ↪each combination of features
##TODO##
```

Generate an elbow plot for this data set. To what extent does this elbow plot support the assumption that there are three clusters present in the data?

[ ]:
```
# Generate an elbow plot for this dataset
##TODO##

# Plot the elbow
##TODO##
```

4 Hierarchical clustering

In this question we investigate the use of hierarchical clustering on the Iris data set. SciPy (pronounced 'Sigh Pie') is a Python-based ecosystem of open-source software for mathematics, science,

and engineering. We start by importing packages `dendrogram` and `linkage`.

```
[ ]: from scipy.cluster.hierarchy import dendrogram, linkage
```

The following will generate a dendogram for the iris data set:

```
[ ]: linked = linkage(X, 'single')
     labelList = range(len(X))
     plt.figure(figsize=(10, 7))
     dendrogram(linked,labels=labelList)
     plt.show()
```

Recall from the lectures that there are a number of ways of measuring the distance between clusters. For example:

- Minimum distance: $d(S,T) = \min\{d(x,y) : x \in S, y \in T\}$
- Average distance: $d(S,T) = \frac{1}{|S||T|}\sum_{(x,y)} d(x,y)$
- Maximum distance: $d(S,T) = \max\{d(x,y) : x \in S, y \in T\}$
- Centroid distance: $d(S,T) = d(\frac{\sum_{x\in S} x}{|S|}\frac{\sum_{y\in T} y}{|T|})$

The parameter `'single'` in linkage refers to minimum distance. This can be change to `'average'` for average distance, `'complete'` for maximum distance and `'centroid'` for centroid distance. Generate the dendogram for each of these cases. Comment on which metrics are most consistent with the assumption of 3 clusters in the iris data set.

```
[ ]: # Generate dendrograms for each distance metric
     ##TODO##
```

The metrics most consistent with the assumption of 3 clusters are:

5 Gaussian Mixture models

In this question we investigate the use of Gaussian clustering on the Iris data set.

```
[ ]: from sklearn.mixture import GaussianMixture as GMM
     gmm = GMM(n_components=3)
     gmm.fit(X)
```

We can extract the parameters for the learnt Gaussian distributions as follows:

```
[ ]: print(gmm.means_)
     print(gmm.covariances_)
```

How do the means for the three distributions compare with the centroids from a 3-cluster k-means on this dataset?

[ ]:
```
# Compare the means from the GMM clusters with the means from
# the k-means clusters
##TODO##
```

Use the command `print(gmm.weights_)` to look at the weights for each distribution. What do these weights tell us about the composition of the three clusters?

[ ]:
```
##TODO##
```

Generate scatter plots for different 2-D combinations of the features.

[ ]:
```
##TODO##
```