Customering Customer Data

Dr Rob Collins

Version 0, 23rd August 2023

GNU Free Documentation License

Introduction

In this workshop we will be introducing an un-supervised Machine Learning tool - that of 'Customering'. Customering is intended to help companies better understand their customer base.

This particular workshop uses the Kaggle dataset - which is one of the simpler un-supervised Machine learning datasets.

Instructions for Students

This workbook includes some empty code blocks. In these cases you are required to code the cell the block based on your previous knowledge and the instructions given in the text. Alternatively, you can simply type **Source Code** and

In working through the following workbook, please do the following:

- Create a copy - keeper version of your own workbook
- **Read and Run the code** - to ensure you understand what it does
- **Comment out of the code blocks** - to check your understanding
- **Fix the broken code** - to fix any broken code
- **Fix the broken code** - to fix any broken code
- **Fix the broken code** - to fix any broken code
- **Fix the broken code** - to fix any broken code
- **Write code** - in the empty code blocks - where you are asked to do so
- **Write code** - in the empty code blocks - where you are asked to do so
- **Write code** - in the empty code blocks - where you are asked to do so

```
# Author: Jake Vanderplas <vanderplas@astro.washington.edu>
# License: BSD 3 clause (C) INRIA, INRIA-Saclay
# Updated by: Carol Willing
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from sklearn.cluster import (KMeans, MiniBatchKMeans)
from sklearn.metrics.pairwise import pairwise_distances_argmin
from sklearn.datasets import make_blobs

# Generate sample data
np.random.seed(0)
batch_size = 45
centers = [[1, 1], [-1, -1], [1, -1]]
n_clusters = len(centers)
X, labels_true = make_blobs(n_samples=3000, centers=centers, cluster_std=0.7)

# Compute clustering with K-Means
k_means = KMeans(init='k-means++', n_clusters=3, n_init=10)
t0 = time.time()
k_means.fit(X)
t_batch = time.time() - t0
k_means_labels = k_means.labels_
k_means_cluster_centers = k_means.cluster_centers_
k_means_labels_unique = np.unique(k_means_labels)

# Compute clustering with MiniBatchKMeans
mbk = MiniBatchKMeans(init='k-means++', n_clusters=3, batch_size=batch_size,
                      n_init=10, max_no_improvement=10, verbose=0)
t0 = time.time()
mbk.fit(X)
t_mini_batch = time.time() - t0
mbk_means_labels = mbk.labels_
mbk_means_cluster_centers = mbk.cluster_centers_
mbk_means_labels_unique = np.unique(mbk_means_labels)

# Plot result
fig = plt.figure(figure_size_factor*fig_size[0], figure_size_factor*fig_size[1])
for this_centroid, this_count, colors in zip(k_means_cluster_centers,
                                             np.bincount(k_means_labels),
                                             cm.spectral(np.linspace(0, 1, len(k_means_cluster_centers)))):
    ax = fig.add_subplot(1, 3, 1, frameon=False, axisbelow=True)
    ax.set_xticks([])
    ax.set_yticks([])
    ax.plot(X[k_means_labels == this_centroid, 0],
            X[k_means_labels == this_centroid, 1], '.', markersize=4, color=colors)
    if hexagonal_centers:
        ax.plot(this_centroid[0], this_centroid[1], 'h', markersize=14,
                markeredgecolor='k', color=colors, alpha=0.5)
    else:
        ax.plot(this_centroid[0], this_centroid[1], 'o', markersize=14,
                markeredgecolor='k', color=colors, alpha=0.5)
    if plot_legend:
        ax.legend(loc='upper right', numpoints=1, ncol=len(k_means_cluster_centers),
                  bbox_to_anchor=(1.0, .05), fancybox=True, shadow=False)
    ax.text(.99, .01, 'elapsed %.3ms\n%iiter\n%i\n%i\n%i
```