Feature Clustering: A Simple Solution to Many Machine Learning Problems

Vincent Granville, Ph.D.
vincentg@MLTechniques.com
www.MLTechniques.com
Version 1.0, March 2023

1 Introduction

Feature clustering is an unsupervised machine learning technique to separate the features of a dataset into homogeneous groups. In short, it is a clustering procedure, but performed on the features rather than on the observations. Such techniques often rely on a similarity metric, measuring how close two features are to each other. In this article, I use the absolute value of the correlation between two features. An immediate consequence is that the technique is scale-invariant: it does not depend on the units of measurement in your dataset. Of course, in some instances, it makes sense to transform the data using a logit or log transform prior to using the technique, to turn a multiplicative setting into an additive one.

The technique can also be used for traditional clustering performed on the observations. In that case, it is useful in the presence of wide data: when you have a large number of features but a small number of observations, sometimes smaller than the number of features as in clinical trials. When applied to features, it allows you to break down a high-dimensional problem (the dimension is the number of features), into a number of low-dimensional problems. It can accelerate many algorithms – those with computing time growing exponentially fast with the dimension – and at the same time avoid issues related to the "curse of dimensionality". In fact it can be used as a data reduction technique, where feature clusters with a low average correlation (in absolute value) are removed from the data set.

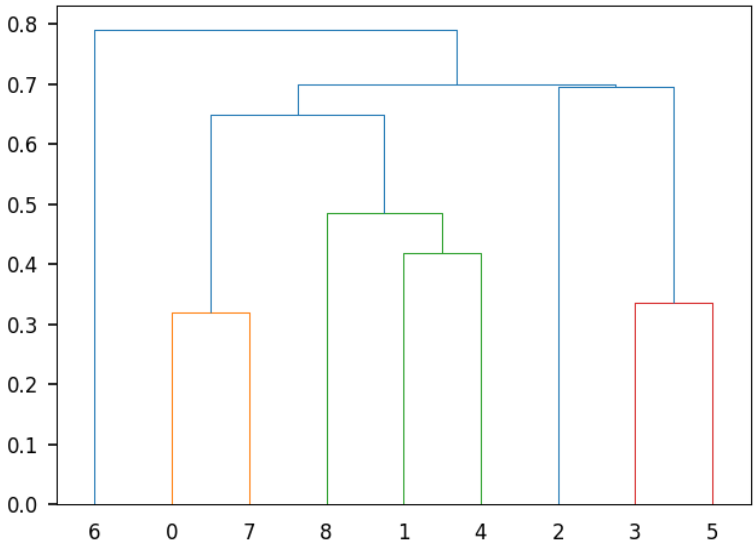

Figure 1: Feature clustering using Scipy; Y-axis is $1 - |\rho|$

Applications are numerous. In my case I used it in the context of synthetic data generation, especially with generative adversarial networks (GAN). The idea is is to identify clusters of related features, and apply a separate GAN to each of them, then put the synthetizations altogether back into one dataset. The benefits are faster processing with little to no loss in terms of capturing the full correlation structure present in the data set. It also increases the robustness and explainability of the method, making it less volatile during the successive epochs in the GAN model. For details, see chapter about GAN in my book on synthetic data [1], or this article.

I summarize the feature clustering results in section 2. I used the technique on a Kaggle dataset with 9 features, consisting of medical measurements. I offer two Python implementations: one based on hierarchical clustering in section 3.1, and one based on connected components (a fundamental graph theory algorithm) in section 3.2. In addition, the technique leads to a simple visualization of the 9-dimensional dataset, with one scatterplot and two colors: orange for diabetes and blue for non-diabetes. Here diabetes is the binary response

feature. This is because the largest feature cluster contains only 3 features, and one of them is the response. In any well-designed experiment, you would expect the response to always be in a large feature cluster.

No linear algebra or calculus is required: the method is essentially math-free. This is in contrast to principal component analysis (PCA) which relies on eigenvalues, and turns your features into meaningless, arbitrary linear combinations that are hard to interpret.

2 Method and case study

The topic of feature clustering is well covered in the literature, see for instance [2]. Here I provide simple methods. The first one consists of finding the connected components [Wiki] in the correlation matrix. The algorithm is a slight adaptation of a version used to detect connected components in nearest neighbor graphs. Two features are connected if their correlation is above some parameter named `threshold` in the Python code. A cluster of features is just a connected component of the undirected graph in question. The second method uses hierarchical clustering. It requires only a few lines of code.

	Pregnanci	Glucose	BloodPres	SkinThickr	Insulin	BMI	DiabetesP	Age	Outcome
Pregnanci	1.00	0.20	0.21	0.09	0.08	-0.03	0.01	0.68	0.26
Glucose	0.20	1.00	0.21	0.20	0.58	0.21	0.14	0.34	0.52
BloodPres	0.21	0.21	1.00	0.23	0.10	0.30	-0.02	0.30	0.19
SkinThickr	0.09	0.20	0.23	1.00	0.18	0.66	0.16	0.17	0.26
Insulin	0.08	0.58	0.10	0.18	1.00	0.23	0.14	0.22	0.30
BMI	-0.03	0.21	0.30	0.66	0.23	1.00	0.16	0.07	0.27
DiabetesP	0.01	0.14	-0.02	0.16	0.14	0.16	1.00	0.09	0.21
Age	0.68	0.34	0.30	0.17	0.22	0.07	0.09	1.00	0.35
Outcome	0.26	0.52	0.19	0.26	0.30	0.27	0.21	0.35	1.00

Figure 2: Correlation matrix for the medical dataset

I applied the method to the medical dataset with `threshold=0.4`. See its correlation matrix in Figure 2. Five feature clusters are detected: $\{0, 7\}, \{3, 5\}, \{1, 4, 8\}, \{2\}, \{6\}$ in that order as seen in the dendogram in Figure 1. For instance feature 0 corresponds to pregnancies, 1 to glucose, 2 to blood pressure, and so on, with feature 9 being the binary response: Outcome = 1 if patient has diabetes, or 0 otherwise.

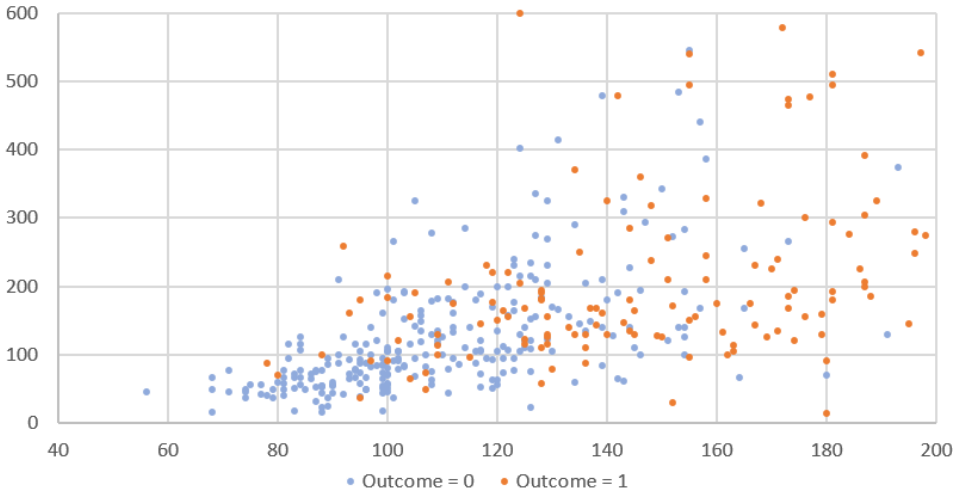

Figure 3: Scatterplot of feature 1 (horizontal axis) and 4 (vertical axis)

The medical dataset can be found on GitHub, here. I removed observations with missing values (encoded as 0 except for the response). A separate feature clustering step can be applied to these observations. Figure 1 shows the dendogram produced by the code in section 3.1. The Y-axis represents $1 - |\rho|$, where ρ is the correlation between 2 features. The 9 features are labeled 0 to 8, with the mapping between label and the feature name listed in Table 1. The 5 clusters, with 2 of them consisting of a single feature, are listed in table 2.

The scatterplot in Figure 3 illustrates the "best" possible visualization of the 9-dimensional dataset, using just 3 features: the response or feature 8 (orange = diabetes, blue = no diabetes), the Glucose feature (X-

axis) and the Insulin feature (Y-axis). These 3 features belong to the largest feature cluster that also includes the response, as shown in Table 2. It provides a compact summary of the medical dataset. Each dot in the scatterplot represents an observation. First, it is clear that diabetes and no-diabetes have very distinct locations. It makes discrimination relatively easy, given the reduction to a 3-dimensional space. Of course, there is some overlapping, which would be less pronounced if you add more relevant features. Then, the best predictors of diabetes are the two other features from the feature cluster in question: Glucose and Insulin. In short, this feature cluster consists of highly inter-correlated features, with Glucose and Insulin also noticeably correlated.

Finally, feature clustering via the correlation matrix is scale-invariant. You can also use this methodology for traditional clustering, by swapping features and observations. For instance, with wide data, that is, a small number of observations (say less than 10,000) with a large number of features, as in clinical trials.

Code	Feature name	Description
0	Pregnancies	Number of pregnancies
1	Glucose	Plasma glucose concentration
2	BloodPressure	Diastolic blood pressure in mm/Hg
3	SkinThickness	Triceps skinfold thickness in mm
4	Insulin	Insulin in U/mL
5	BMI	Body mass index
6	DiabetesPedigreeFunction	Risk based on family history
7	Age	Age of patient
8	Outcome	Patient had diabetes or not

Table 1: Feature mapping table, medical dataset

Cluster ID	Features (codes)	Features (names)
1	0, 7	Pregnancies, Age
2	3, 5	SkinThickness, BMI
3	1, 4, 8	Glucose, Insulin, Outcome
4	2	BloodPressure
5	6	DiabetesPedigreeFunction

Table 2: Feature clusters

3 Python implementations

I provide two implementations of the feature clustering procedure. The first one in section 3.1 uses hierarchical clustering with the Scipy library. The second implementation in section 3.2 is based on connected components.

3.1 Feature clustering via hierarchical clustering

Figure 1 illustrates the output of the code. In Figure 1, ρ is the correlation, the X-axis is the feature label. The code is available on GitHub, here.

```
# feature correlation with hierarchical clustering and dendograms
# featureClusteringScipy.py

import matplotlib.pyplot as plt
import numpy as np
import scipy.spatial.distance as ssd
import scipy.cluster.hierarchy as hcluster

correlMatrix = [
     [1.0000,0.1983,0.2134,0.0932,0.0790,-0.0253,0.0076,0.6796,0.2566],
     [0.1983,1.0000,0.2100,0.1989,0.5812,0.2095,0.1402,0.3436,0.5157],
     [0.2134,0.2100,1.0000,0.2326,0.0985,0.3044,-0.0160,0.3000,0.1927],
```

```
     [0.0932,0.1989,0.2326,1.0000,0.1822,0.6644,0.1605,0.1678,0.2559],
     [0.0790,0.5812,0.0985,0.1822,1.0000,0.2264,0.1359,0.2171,0.3014],
     [-0.0253,0.2095,0.3044,0.6644,0.2264,1.0000,0.1588,0.0698,0.2701],
     [0.0076,0.1402,-0.0160,0.1605,0.1359,0.1588,1.0000,0.0850,0.2093],
     [0.6796,0.3436,0.3000,0.1678,0.2171,0.0698,0.0850,1.0000,0.3508],
     [0.2566,0.5157,0.1927,0.2559,0.3014,0.2701,0.2093,0.3508,1.0000]]

simMatrix = correlMatrix - np.identity(len(correlMatrix))
distVec = ssd.squareform(simMatrix)
linkage = hcluster.linkage(1 - distVec)

plt.figure()
axes = plt.axes()
axes.tick_params(axis='both', which='major', labelsize=8)
for axis in ['top','bottom','left','right']:
   axes.spines[axis].set_linewidth(0.5)
with plt.rc_context({'lines.linewidth': 0.5}):
   dendro = hcluster.dendrogram(linkage,leaf_font_size=8)
plt.show()
```

3.2 Feature clustering via connected components

The following implementation is based on connected components. It provides the same results as the previous one based on hierarchical clustering and dendograms. The version presented here does not use recursivity, yet it is just as fast as any optimum algorithm to detect connected components. The code is also available here. A detailed description can be found in my book [1], in the section dealing with nearest neighbor distances.

```
# feature correlation with connected components
# featureClustering.py

correlMatrix = [
     [1.0000,0.1983,0.2134,0.0932,0.0790,-0.0253,0.0076,0.6796,0.2566],
     [0.1983,1.0000,0.2100,0.1989,0.5812,0.2095,0.1402,0.3436,0.5157],
     [0.2134,0.2100,1.0000,0.2326,0.0985,0.3044,-0.0160,0.3000,0.1927],
     [0.0932,0.1989,0.2326,1.0000,0.1822,0.6644,0.1605,0.1678,0.2559],
     [0.0790,0.5812,0.0985,0.1822,1.0000,0.2264,0.1359,0.2171,0.3014],
     [-0.0253,0.2095,0.3044,0.6644,0.2264,1.0000,0.1588,0.0698,0.2701],
     [0.0076,0.1402,-0.0160,0.1605,0.1359,0.1588,1.0000,0.0850,0.2093],
     [0.6796,0.3436,0.3000,0.1678,0.2171,0.0698,0.0850,1.0000,0.3508],
     [0.2566,0.5157,0.1927,0.2559,0.3014,0.2701,0.2093,0.3508,1.0000]]

dim = len(correlMatrix)
threshold = 0.4 # two features with |correl|>threshold are connected
pairs = {}

for i in range(dim):
   for j in range(i+1,dim):
      dist = abs(correlMatrix[i][j])
      if dist > threshold:
         pairs[(i,j)] = abs(correlMatrix[i][j])
         pairs[(j,i)] = abs(correlMatrix[i][j])

# connected components algo to detect feature clusters on feature pairs

#---
# PART 1: Initialization.

point=[]
NNIdx={}
idxHash={}

n=0
for key in pairs:
   idx = key[0]
```

```
    idx2 = key[1]
    if idx in idxHash:
        idxHash[idx]=idxHash[idx]+1
    else:
        idxHash[idx]=1
    point.append(idx)
    NNIdx[idx]=idx2
    n=n+1


hash={}
for i in range(n):
    idx=point[i]
    if idx in NNIdx:
        substring="~"+str(NNIdx[idx])
    string=""
    if idx in hash:
        string=str(hash[idx])
    if substring not in string:
        if idx in hash:
            hash[idx]=hash[idx]+substring
        else:
            hash[idx]=substring
    substring="~"+str(idx)
    if NNIdx[idx] in hash:
        string=hash[NNIdx[idx]]
    if substring not in string:
        if NNIdx[idx] in hash:
            hash[NNIdx[idx]]=hash[NNIdx[idx]]+substring
        else:
            hash[NNIdx[idx]]=substring

#---
# PART 2: Find the connected components

i=0;
status={}
stack={}
onStack={}
cliqueHash={}

while i<n:

    while (i<n and point[i] in status and status[point[i]]==-1):
        # point[i] already assigned to a clique, move to next point
        i=i+1

    nstack=1
    if i<n:
        idx=point[i]
        stack[0]=idx; # initialize the point stack, by adding $idx
        onStack[idx]=1;
        size=1  # size of the stack at any given time

        while nstack>0:
            idx=stack[nstack-1]
            if (idx not in status) or status[idx] != -1:
                status[idx]=-1 # idx considered processed
                if i<n:
                    if point[i] in cliqueHash:
                        cliqueHash[point[i]]=cliqueHash[point[i]]+"~"+str(idx)
                    else:
                        cliqueHash[point[i]]="~"+str(idx)
                nstack=nstack-1
                aux=hash[idx].split("~")
                aux.pop(0) # remove first (empty) element of aux
```

```
            for idx2 in aux:
               # loop over all points that have point idx as nearest neighbor
               idx2=int(idx2)
               if idx2 not in status or status[idx2] != -1:
                  # add point idx2 on the stack if it is not there yet
                  if idx2 not in onStack:
                     stack[nstack]=idx2
                     nstack=nstack+1
                  onStack[idx2]=1

#---
# PART 3: Save results.

clusterID = 1
for clique in cliqueHash:
   cluster = cliqueHash[clique]
   cluster = cluster.replace('~', ' ')
   print("Feature Cluster number %2d: features %s" %(clusterID, cluster))
   clusterID += 1
clusteredFeature = {}
for feature in range(dim):
   for clique in cliqueHash:
      if str(feature) in cliqueHash[clique]:
         clusteredFeature[feature] = True
for feature in range(dim):
   if feature not in clusteredFeature:
      cluster = " "+str(feature)
      print("Feature Cluster number %2d: features %s" %(clusterID, cluster))
      clusterID += 1
```

References

[1] Vincent Granville. *Synthetic Data and Generative AI*. MLTechniques.com, 2022. [Link]. 1, 4

[2] Hui Liu et al. A new model using multiple feature clustering and neural networks for forecasting hourly $PM_{2.5}$ concentrations. *Engineering*, 6:944–956, 2020. [Link]. 2