K-mean and PCA-Human Activity Recognition

Siyuan Zhang

1 Resources

Student: Siyuan Zhang
Language: Python
Code: https://github.com/Siyuan-gwu/K-mean-and-PCA-Human-Activity-Recognition
Instruction: The instruction to run the code is on github (readme)
Resource:

1. https://www.kaggle.com/ruslankl/k-means-clustering-pca
2. https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a
3. https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c

Code: https://github.com/Siyuan-gwu/K-mean-and-PCA-Human-Activity-Recognition

```
import random
import numpy as np
import pandas as pd
from IPython.display import display
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.metrics import homogeneity_score, completeness_score, \
v_measure_score, adjusted_rand_score, adjusted_mutual_info_score, silhouette_score
```

2 Dataset details

There are two dataset, train and test. Dataset contains 30 volunteers within an age bracket of 10-48 years. Each person performed six activities (walking, walking_upstairs, walking_downstairs, sitting, standing and laying).

Inspect the Dataset

```
data = pd.read_csv('train.csv')


print (data.sample(5))
```

```
        rn         activity  ...  angle.Y.gravityMean  angle.Z.gravityMean
1875  5353           LAYING  ...                0.432               -0.744
3152  9034  WALKING_UPSTAIRS  ...               0.303                0.296
2343  6697           LAYING  ...               -0.522               -0.483
3338  9536  WALKING_UPSTAIRS  ...               0.308                0.197
565   1606  WALKING_UPSTAIRS  ...               0.281                0.233

[5 rows x 563 columns]
```

Print the shape of dataset

```
print (str(data.shape))
```

```
shape of data set: (3609, 563)
```

Print activities

```
labels = data['activity']
data = data.drop(['rn', 'activity'], axis = 1)
labels_keys = labels.unique().tolist()
labels = np.array(labels)
print('Activity labels: ' + str(labels_keys))
```

```
Activity labels: ['STANDING', 'SITTING', 'LAYING', 'WALKING', 'WALKING_DOWNSTAIRS', 'WALKING_UPSTAIRS']
```

Normalize the dataset

```
# min-max normalization
def MaxMinNormalization(x):
    """[0,1] normaliaztion"""
    x = (x - np.min(x)) / (np.max(x) - np.min(x))
    return x
data = MaxMinNormalization(data)
print (data.sample(5))
```

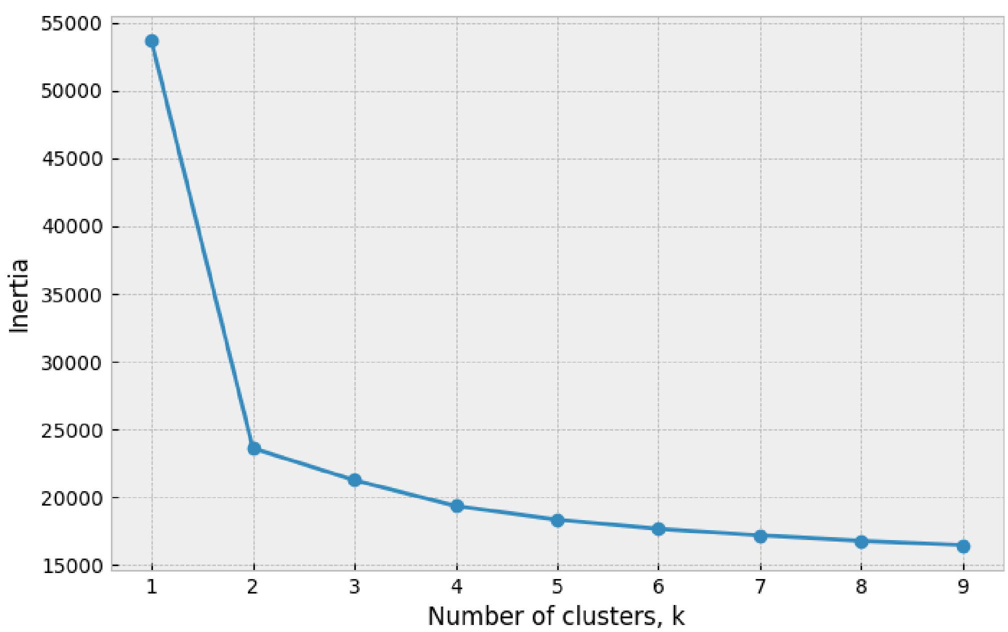
```
      tBodyAcc.mean.X  ...  angle.Z.gravityMean
1957         0.650741  ...             0.283900
314          0.640033  ...             0.385983
1786         0.658979  ...             0.360081
2241         0.710049  ...             0.547994
847          0.604613  ...             0.468664
```

Check for the optimal k

```
# check the optimal k value
ks = range(1, 10)
inertias = []

for k in ks:
    model = KMeans(n_clusters=k)
    model.fit(data)
    inertias.append(model.inertia_)

plt.figure(figsize=(8,5))
plt.style.use('bmh')
plt.plot(ks, inertias, '-o')
plt.xlabel('Number of clusters, k')
plt.ylabel('Inertia')
plt.xticks(ks)
plt.show()
```



We can notice that the optimal value of k is around 2, which means two clusters.

K-means algorithm

```
def k_means(n_clust, data_frame, true_labels):
    k_means = KMeans(n_clusters=n_clust, random_state=123, n_init=30)
    k_means.fit(data_frame)
    c_labels = k_means.labels_
    df = pd.DataFrame({'clust_label': c_labels, 'orig_label': true_labels.tolist()})
    ct = pd.crosstab(df['clust_label'], df['orig_label'])
    y_clust = k_means.predict(data_frame)
    display(ct)
    print('% 9s' % 'inertia  homo    compl   v-meas   ARI     AMI     silhouette')
    print('%i   %.3f   %.3f   %.3f   %.3f   %.3f    %.3f'
          % (k_means.inertia_,
             homogeneity_score(true_labels, y_clust),
             completeness_score(true_labels, y_clust),
             v_measure_score(true_labels, y_clust),
             adjusted_rand_score(true_labels, y_clust),
             adjusted_mutual_info_score(true_labels, y_clust),
             silhouette_score(data_frame, y_clust, metric='euclidean')))


k_means(2, data, labels)
```

Result

```
orig_label      0     1
  FutureWarning)
clust_label
0            1970     0
1               2  1637
inertia  homo    compl   v-meas   ARI     AMI     silhouette
/Users/zhangsiyuan/PycharmProjects/project3/venv/lib/python3.
  FutureWarning)
23608   0.384   0.994   0.553   0.332   0.383    0.474
```

From the result matrix, we can know that 2 clusters have a very high accuracy.

Change the labels into binary, 0 means not moving, 1 means moving

```
labels_binary = labels.copy()
for i in range(len(labels_binary)):
```

```
    if (labels_binary[i] == 'STANDING' or labels_binary[i] == 'SITTING' or labels_binary[i] == 'LAYING'):
        labels_binary[i] = 0
    else:
        labels_binary[i] = 1
labels_binary = np.array(labels_binary.astype(int))


k_means(2, data, labels_binary)
```

```
orig_label      0     1
clust_label
0            1970     0
1               2  1637
inertia  homo    compl   v-meas   ARI     AMI     silhouette
/Users/zhangsiyuan/PycharmProjects/project3/venv/lib/python3.7
  FutureWarning)
23608   0.994   0.994   0.994   0.998   0.994    0.474
```

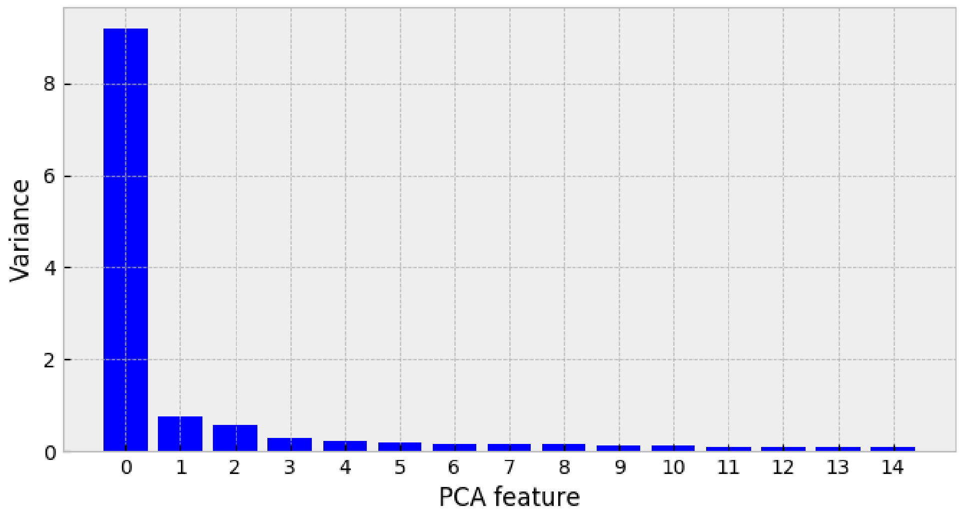
The result is the same.

PCA part

Check for the optimal value of k

```
#PCA
pca = PCA(random_state=123)
pca.fit(data)
features = range(pca.n_components_)


plt.figure(figsize=(8,4))
plt.bar(features[:15], pca.explained_variance_[:15], color='lightskyblue')
plt.xlabel('PCA feature')
plt.ylabel('Variance')
plt.xticks(features[:15])
plt.show()
```



We notice that 1 feature seems to be best fit to our problem.

PCA algorithm

```
def pca_transform(n_comp):
    pca = PCA(n_components=n_comp, random_state=123)
    global data_reduced
    data_reduced = pca.fit_transform(data)
    print('Shape of the new Data df: ' + str(data_reduced.shape))


pca_transform(1)
k_means(2, data_reduced, labels_binary)
```

```
orig_label      0     1
clust_label
0            1970     0
1               2  1637
inertia  homo    compl   v-meas   ARI     AMI     silhouette
/Users/zhangsiyuan/PycharmProjects/project3/venv/lib/python3.7
  FutureWarning)
3184    0.994   0.994   0.994   0.998   0.994   0.826
```

We can know that the "Silhouette" seems better than before.

3 Algorithm Description

K-means algorithm

The way k-means algorithm work is as follows:

1. Specify number of clusters K.
2. Initialize centroids by first shuffling the dataset and then randomly selecting K data points for the centroids without replacement.
3. Keep iterating until there is no change to the centroids. i.e assignment of data points to clusters isn't changing.
4. Compute the sum of the squared distance between data points and all centroids.
5. Assign each data point to the closest cluster (centroid).
6. Compute the centroids for the clusters by taking the average of the all data points that belong to each cluster.

The approach kmeans follows to solve the problem is called Expectation-Maximization. The E-step is assigning the data points to the closest cluster. The M-step is computing the centroid of each cluster.

The objective function is:

$$J = \sum_{i=1}^{m} \sum_{k=1}^{K} w_{ik} \|x^i - \mu_k\|^2$$

where wik=1 for data point xi if it belongs to cluster k; otherwise, wik=0. Also, μk is the centroid of xi's cluster.

PCA algorithm

Principal component analysis (PCA) is a technique to bring out strong patterns in a dataset by suppressing variations. It is used to clean data sets to make it easy to explore and analyze. The algorithm of Principal Component Analysis is based on a few mathematical ideas namely:

1. Variance and Covariance
2. Eigen Vectors and Eigen values

Algorithm steps:

1. Get your data
2. Give your data a structure
3. Standardize your data
4. Get covariance of matrix Z
5. Calculate Eigen Vectors and Eigen Values
6. Sort the Eigen Vectors

7. Calculate the new features
8. Drop unimportant features from the set

4 Runtime

For k-means:

n: number of points
k: number of clusters
i: number of iterations
d: number of attributes

The time complexity of k-means is $O(n * k * i * d)$

For PCA:

Suppose we have n data points, and each represented with p features.
Covariances matrix computation is $O(p^2 * n)$, its eigen-value decomposition is $O(p^\wedge 3)$.
So the time complexity of PCA is $O(p^2 * n + p^\wedge 3)$