Published in More Python

Dmitriy Follow

Feb 20, 2019 · 5 min read · ✦ · Listen

Principal Component Analysis and k-means Clustering to Visualize a High Dimensional Dataset

Dimensionality reduction by PCA and k-means clustering to visualize patterns in data from diet, physical examinations, and hospital laboratory reports.

Key insights:

- **There are clusters in the National Health and Nutrition Exam Survey (combined diet, medical, and exam datasets, 2013- 2014) which are only visible via dimensionality reduction.**
- **PCA in conjunction with k-means is a powerful method for visualizing high dimensional data.**

I recently learned about principal component analysis (PCA) and I was eager to try to put it into practice, so I downloaded data from the National Health and Nutrition Examination Survey and began my analysis. The data contained nearly 200 features (columns) and there was no way in hell I could get a broad overview of all of them through traditional methods

of visualization. Luckily, this is what doing PCA is all about. You take a ton of features, project them onto a lower-dimensional space, reduce them down to just a few important principal ones, and visualize them. Alternatively, it's possible to use these reduced components in a machine learning pipeline, but that's a topic for a different post.

To better understand the magic of PCA, let's dive right in and see how I did it with my dataset in three basic steps.

Step 1: Reduce Dimensionality

In this step, we will find the optimal number of components which capture the greatest amount of variance in the data. In my case, as seen in Fig. 1 below, that number is three.

```
# Imports
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
```



```
# Standardize the data to have a mean of ~0 and a variance of 1
X_std = StandardScaler().fit_transform(df)

# Create a PCA instance: pca
pca = PCA(n_components=20)
principalComponents = pca.fit_transform(X_std)

# Plot the explained variances
features = range(pca.n_components_)
plt.bar(features, pca.explained_variance_ratio_, color='black')
plt.xlabel('PCA features')
plt.ylabel('variance %')
plt.xticks(features)

# Save components to a DataFrame
PCA_components = pd.DataFrame(principalComponents)
```

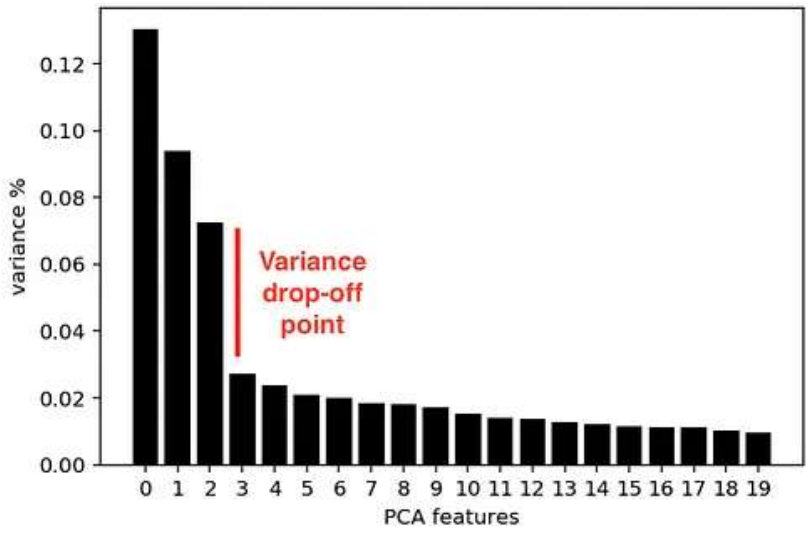

Figure 1. Scree plot showing variance drop-off after the third component.

Fig. 1 shows that the first three components explain the majority of the variance in our data. For this visualization use case, we will quickly plot just the first two. We do this to notice if there are any clear clusters.


```
plt.scatter(PCA_components[0], PCA_components[1], alpha=.1,
color='black')
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
```

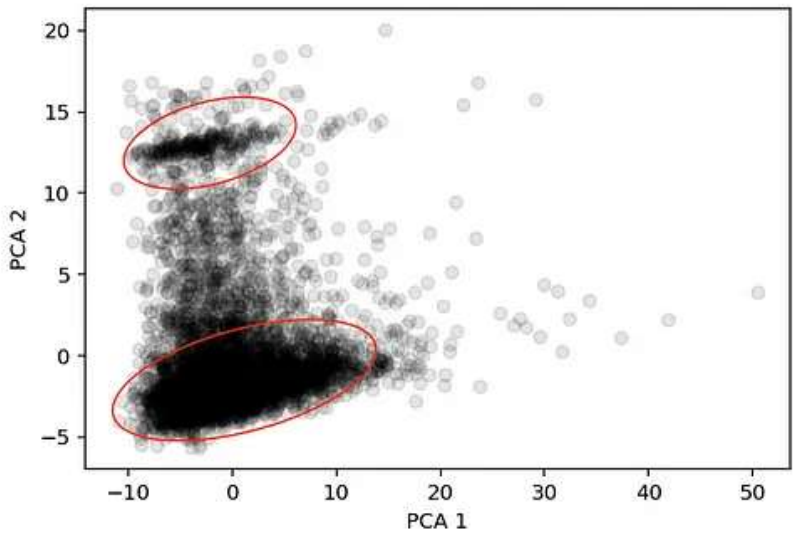

Figure 2. Scatter plot of the first two components of the PCA model.

Fig. 2 shows at least two distinguishable clusters. This factoid tells us that the observations in the dataset can be grouped. Because each observation in the data is a diet, lab, and physical exam for one person, we could say that the clusters represent different groups of people. It's important to note that we do not have a target variable by which to label these groups, so we do not know exactly what these labels are. In a utopian situation, this type of analysis would let us see the sample population segregated by health condition. Fig. 2 does not show all the meaningful principal components, however. To visualize the rest of the reduced dataset with much greater granularity, we will use k-means clustering.

Step 2: Find the Clusters

In this step, we will use k-means clustering to view the top three PCA components. To do this, we will first fit these principal components to the k-means algorithm and determine the best number of clusters. Determining the ideal number of clusters for our k-means model can be done by measuring the sum of the squared distances to the nearest cluster center aka inertia. Much like the scree plot in fig. 1 for PCA, the k-means scree plot below indicates the percentage of variance explained, but in slightly different terms, as a function of the number of clusters.

```
ks = range(1, 10)
inertias = []

for k in ks:
    # Create a KMeans instance with k clusters: model
    model = KMeans(n_clusters=k)

    # Fit model to samples
    model.fit(PCA_components.iloc[:,:3])

    # Append the inertia to the list of inertias
    inertias.append(model.inertia_)

plt.plot(ks, inertias, '-o', color='black')
plt.xlabel('number of clusters, k')
plt.ylabel('inertia')
plt.xticks(ks)
plt.show()
```

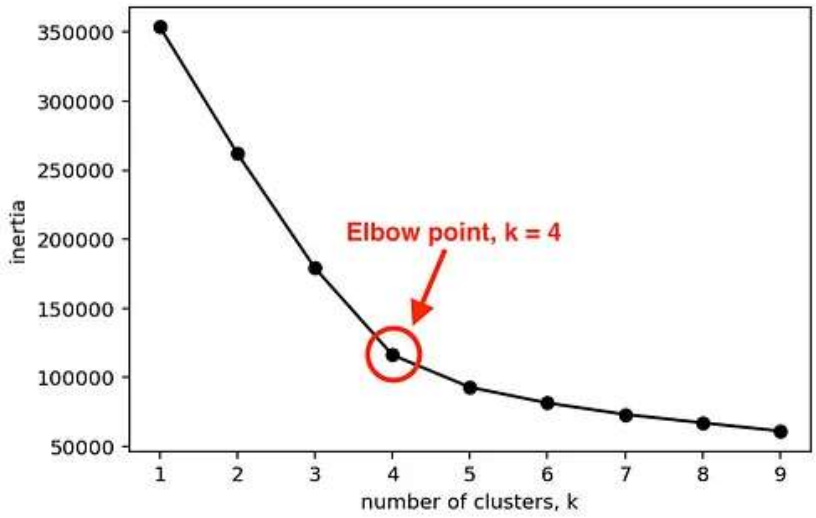

Figure 3. Scree plot showing a slow decrease of inertia after k = 4.

Fig. 3 shows that after 4 clusters(at the elbow), the change in the value of inertia is no longer significant, and most likely, neither is the variance of the rest of the data after the elbow point. Therefore we can discard everything after k=4 and proceed to the last step in the process.

Step 3: Visualize and Interpret the Clusters

I did this project with a basic question in mind: can people be grouped based on features like physical examination results, complete blood counts, and diet records? Reducing all those features down to principal components and then visualizing the clusters in those principal components using k-means hints that the answer to my question is most likely yes.

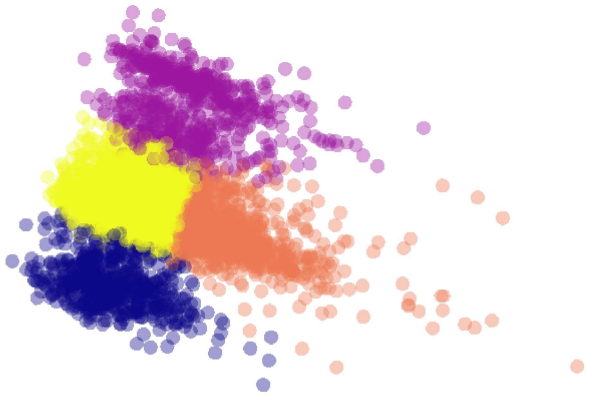

Figure 4. Interactive 3-D visualization of k-means clustered PCA components. Go ahead, interact with it.

Figure 4 was made with Plotly and shows some clearly defined clusters in the data. I did not label the dataset, so we do not know the names of the clusters. This does not mean that we couldn't go back and label these groups, however. Now that we know how many clusters there are in our data, we have a better sense of how many groups we can label the population with. As an example, it's possible to come up with a model that grades well-being in this population on four grades. Introducing these labels back into the reduced dataset on the unique id of each sample will allow us to visualize them by cluster.

The ability to notice otherwise unseen patterns and to come up with a model to generalize those patterns onto observations is precisely why tools like PCA and k-means are essential in any data scientist's toolbox. They allow us to see the big picture while we pay attention to the details.

Follow me if you enjoyed this and want to see more.

How to Use BERTopic and SentenceTransformer to Group Google Search Results into Topics

A tutorial on finding commonalities between different articles on websites using BERTopic and SentenceTransformer.

python.plainenglish.io

Machine Learning Data Science Healthcare Biology