0401-solution-notebook

April 5, 2024

1 004 PCA & Clustering With Python

- Written by Alexandre Gazagnes
- Last update: 2024-02-01

1.1 About

Context :

You're an export project manager for a major food manufacturer. You are in charge of poultry departement. You have been asked to identify segments of countries within the company's database in order to target them with personalized marketing campaigns.

Data :

After a quick look on the internet, you find a very interesting dataset on the FAO website. It contains a list of countries with various indicators. You decide to use this dataset to identify segments of countries.

You can download the "raw" dataset here.

You can also use a preprocessed version of the dataset here.

Mission :

Your objective is to

- Take a quick tour of the data to understand the data set
- Clean up the dataset if necessary
- Perform clustering with Kmeans and Agglomerative Clustering, focusing on countries with large potential markets: populous countries, wealthy countries and/or countries with high import levels
- You need to be able to understand and explain the clusters you've created.

1.2 Preliminaries

1.2.1 System

These commands will display the system information:

Uncomment theses lines if needed.

[ ]:
```
# pwd
```

[ ]:
```
# cd ..
```

[ ]:
```
# ls
```

[ ]:
```
# cd ..
```

[ ]:
```
# ls
```

These commands will install the required packages:

[ ]:
```
# !pip install pandas matplotlib seaborn plotly scikit-learn
```

This command will download the dataset:

[ ]:
```
# !wget https://gist.githubusercontent.com/AlexandreGazagnes/
↪7cc1176dfe0e281c45e0119210187ae2/raw/
↪4e5c4caac35b88fcedea810f470957b7c5f9b5af/chicken_cleaned.csv
```

1.2.2 Import

Import data libraries:

[ ]:
```
import pandas as pd
import numpy as np
```

Import Graphical libraries:

[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
```

Import Machine Learning libraries:

[ ]:
```
# must to have (mandarory)
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans

# nice to have
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_score
from sklearn.metrics import davies_bouldin_score
from sklearn.datasets import load_iris
from scipy.cluster.hierarchy import dendrogram, linkage
```

1.2.3 Get the data

1st option : Download the dataset from the web

```
[ ]: url = "https://gist.githubusercontent.com/AlexandreGazagnes/
      ↪7cc1176dfe0e281c45e0119210187ae2/raw/
      ↪4e5c4caac35b88fcedea810f470957b7c5f9b5af/chicken_cleaned.csv"
     df = pd.read_csv(url)
     df.head()
```

2nd Option : Read data from a file

```
[ ]: # or

     # fn = "./chicken_cleaned.csv"
     # df = pd.read_csv(fn)
     # df.head()
```

3rd Option : Load a toy dataset

```
[ ]: # or

     # data = load_iris()
     # df = pd.DataFrame(data.data, columns=data.feature_names)
     # df["Species"] = data.target
     # df.head()
```

1.3 Quick Data Viz

Write a function to display the correlation matrix:

```
[ ]: # With a function



     def make_corr_heatmap(df):
         corr = df.select_dtypes(include="number").corr()
         mask = np.triu(corr)
         sns.heatmap(
             corr, annot=True, cmap="coolwarm", fmt=".2f", vmin=-1, vmax=1, mask=mask
         )
```

```
[ ]: make_corr_heatmap(df)
```

Make a pairplot of the numerical columns:

This visualization can be slow with large datasets. Use VIZ = True / False to enable / disable the visualization.

```
[ ]: VIZ = False  # Enable this with True
     if VIZ:
         sns.pairplot(df.select_dtypes(exclude="object"), corner=True)
```

1.3.1 Scale

[ ]:
```
df
```

[ ]:
```

```

Select only numerical columns:

[ ]:
```
X = df.select_dtypes(include="number")
X
```

Use SciKit Learn to scale the dataset:

[ ]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled
```

Rebuild a DataFrame with the scaled data:

[ ]:
```
X_scaled = pd.DataFrame(X_scaled, columns=X.columns)
X_scaled.head()
```

Check that data were scaled:

[ ]:
```
X_scaled.describe().round(2)
```

Of course you can compute the scaling manually:

[ ]:
```
X_scaled = (X - X.mean()) / X.std()
X_scaled.head()
```

[ ]:
```
X_scaled.describe().round(2)
```

1.4 Principal Component Analysis

1.4.1 Initialisation and fit

Initialize a PCA :

[ ]:
```
pca = PCA(n_components=6)
pca
```

Fit :

[ ]:
```
pca.fit(X_scaled)
```

Here is our new dataset :

[ ]:
```
X_proj = pca.transform(X_scaled)
X_proj
```

Use pandas to create a DataFrame :

```
[ ]: X_proj = pd.DataFrame(
         X_proj, columns=[f"PC{i}" for i in range(1, pca.n_components_ + 1)]
     )
     X_proj
```

1.4.2 Analyse the components

```
[ ]: pcs = pca.components_
     pcs
```

```
[ ]: components = pd.DataFrame(
         pcs, columns=X.columns, index=[f"PC{i}" for i in range(1, pca.n_components_␣
      ↪+ 1)]
     )
     components
```

Recompute the first value :

```
[ ]: value = X_proj.iloc[0, 0]
     value
```

```
[ ]: X_scaled.head(1)
```

```
[ ]: (
         (-0.37 * 0.66)
         + (-0.44 * 0.11)
         + (-1.1 * 0.34)
         + (-0.15 * 0.46)
         + (-0.46 * -0.1)
         + (0.11 * -0.46)
     )
```

```
[ ]: sum([i * j for i, j in zip(pcs[0], X_scaled.iloc[0])])
```

```
[ ]: components = components.T
     components
```

```
[ ]: sns.heatmap(components, cmap="coolwarm", vmax=1, vmin=-1, annot=True, fmt=".2f")
```

1.4.3 Plot explained variance

The explained variance ratio is pre-computed :

```
[ ]: pca.explained_variance_ratio_
```

We can plot it :

```
[ ]: sns.lineplot(y=pca.explained_variance_ratio_, x=components.columns, marker="o")
```

A better feature is the cumulative variance :

[ ]:
```
cum_var = pca.explained_variance_ratio_.cumsum()
cum_var
```

We can plot it :

[ ]:
```
x = ["PC0"] + components.columns.tolist()
y = [0] + cum_var.tolist()
sns.lineplot(y=y, x=x, marker="o")
```

1.4.4 Correlation graph

[ ]:
```
def correlation_graph(
    X_scaled,
    pca,
    dim: list = [0, 1],
):
    """Affiche le graphe des correlations

    Positional arguments :
        X_scaled : DataFrame | np.array : le dataset scaled
        pca : PCA : l'objet PCA déjà fitté

    Optional arguments :
        dim : list ou tuple : le couple x,y des plans à afficher, exemple [0,1]␣
↪pour F1, F2
    """

    # Extrait x et y
    x, y = dim

    # features
    features = X_scaled.columns

    # Taille de l'image (en inches)
    fig, ax = plt.subplots(figsize=(10, 9))

    # Pour chaque composante :
    for i in range(0, pca.components_.shape[1]):
        # Les flèches
        ax.arrow(
            0,
            0,
            pca.components_[x, i],
            pca.components_[y, i],
            head_width=0.07,
            head_length=0.07,
```

```
            width=0.02,
        )

        # Les labels
        plt.text(
            pca.components_[x, i] + 0.05,
            pca.components_[y, i] + 0.05,
            features[i],
        )

    # Affichage des lignes horizontales et verticales
    plt.plot([-1, 1], [0, 0], color="grey", ls="--")
    plt.plot([0, 0], [-1, 1], color="grey", ls="--")

    # Nom des axes, avec le pourcentage d'inertie expliqué
    plt.xlabel(
        "F{} ({}%)".format(x + 1, round(100 * pca.explained_variance_ratio_[x],␣
↪1))
    )
    plt.ylabel(
        "F{} ({}%)".format(y + 1, round(100 * pca.explained_variance_ratio_[y],␣
↪1))
    )

    # title
    plt.title("Cercle des corrélations (F{} et F{})".format(x + 1, y + 1))

    # Le cercle
    an = np.linspace(0, 2 * np.pi, 100)
    plt.plot(np.cos(an), np.sin(an))  # Add a unit circle for scale

    # Axes et display
    plt.axis("equal")
    plt.show(block=False)
```

[ ]:
```
correlation_graph(
    X_scaled,
    pca,
    dim=[0, 1],
)
```

[ ]:
```
correlation_graph(
    X_scaled,
    pca,
    dim=[0, 2],
)
```

[ ]:
```python
correlation_graph(
    X_scaled,
    pca,
    dim=[1, 2],
)
```

1.4.5 Factorial planes

[ ]:
```python
def factorial_planes(
    X_,
    pca,
    dim,
    labels: list = None,
    clusters: list = None,
    figsize: list = [12, 10],
    fontsize=14,
):
    """Affiche les plans factoriels"""

    x, y = dim

    dtypes = (pd.DataFrame, np.ndarray, pd.Series, list, tuple, set)
    if not isinstance(labels, dtypes):
        labels = []
    if not isinstance(clusters, dtypes):
        clusters = []

    # Initialisation de la figure
    fig, ax = plt.subplots(1, 1, figsize=figsize)

    if len(clusters):
        sns.scatterplot(data=None, x=X_[:, x], y=X_[:, y], hue=clusters)
    else:
        sns.scatterplot(data=None, x=X_[:, x], y=X_[:, y])

    # Si la variable pca a été fournie, on peut calculer le % de variance de␣
 ↪chaque axe
    v1 = str(round(100 * pca.explained_variance_ratio_[x])) + " %"
    v2 = str(round(100 * pca.explained_variance_ratio_[y])) + " %"

    # Nom des axes, avec le pourcentage d'inertie expliqué
    ax.set_xlabel(f"F{x+1} {v1}")
    ax.set_ylabel(f"F{y+1} {v2}")

    # Valeur x max et y max
    x_max = np.abs(X_[:, x]).max() * 1.1
    y_max = np.abs(X_[:, y]).max() * 1.1
```

```
    # On borne x et y
    ax.set_xlim(left=-x_max, right=x_max)
    ax.set_ylim(bottom=-y_max, top=y_max)

    # Affichage des lignes horizontales et verticales
    plt.plot([-x_max, x_max], [0, 0], color="grey", alpha=0.8)
    plt.plot([0, 0], [-y_max, y_max], color="grey", alpha=0.8)

    # Affichage des labels des points
    if len(labels):
        for i, (_x, _y) in enumerate(X_[:, [x, y]]):
            plt.text(
                _x, _y + 0.05, labels[i], fontsize=fontsize, ha="center",␣
↪va="center"
            )

    # Titre et display
    plt.title(f"Projection des individus (sur F{x+1} et F{y+1})")
    plt.show()
```

[ ]:
```
factorial_planes(X_proj.values, pca, [0, 1])
```

[ ]:
```
factorial_planes(
    X_proj.values, pca, [0, 1], labels=df.zone.values, figsize=(20, 16),␣
↪fontsize=6
)
```

1.5 Kmeans

1.5.1 First Naive Clustering

Initialize the Kmeans model with arbitrary number of clusters:

[ ]:
```
kmeans = KMeans(n_clusters=6, random_state=42)
kmeans
```

Fit the Kmeans model to the data:

[ ]:
```
kmeans.fit(X_scaled)
kmeans
```

[ ]:
```
labels = kmeans.predict(X_scaled)
labels
```

Use alphanumerical labels for the clusters:

[ ]:
```
labels_values = {
    0: "A",
```

```
    1: "B",
    2: "C",
    3: "D",
    4: "E",
    5: "F",
    6: "G",
    7: "H",
    8: "I",
    9: "J",
}

labels = [labels_values[l] for l in labels]
labels[:10]
```

Create a copy of the original dataset and add labels:

[ ]:
```
_df = df.copy()
_df["labels"] = labels
_df
```

What about A cluster?

[ ]:
```
_df.loc[_df.labels == "A"]
```

What about B cluster?

[ ]:
```
_df.loc[_df.labels == "B"]
```

Etc etc…

[ ]:
```
_df.loc[_df.labels == "C"]
```

[ ]:
```
_df.loc[_df.labels == "D"]
```

[ ]:
```
_df.loc[_df.labels == "E"]
```

[ ]:
```
_df.loc[_df.labels == "F"]
```

Build a boxplot for each cluster:

[ ]:
```
for col in _df.select_dtypes(include="number").columns:
    sns.boxplot(data=_df, y=col, hue="labels")
    plt.show()
```

Same but with plotly:

[ ]:
```
for col in _df.select_dtypes(include="number").columns:
    fig = px.box(_df, y=col, color="labels")
    fig.show()
```

1.5.2 Find the best number of clusters

Compute the inertia for different number of clusters:

[ ]:
```python
# WCSS : Within-Cluster-Sum-of-Squares

inertia_list = []
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(X_scaled)
    print(k, kmeans.inertia_)
    inertia_list.append(kmeans.inertia_)
```

Plot the inertia for different number of clusters:

[ ]:
```python
plt.plot(range(2, 11), inertia_list, marker="o")
```

Same but with davies bouldin score:

[ ]:
```python
# DB Score

db_list = []
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(X_scaled)
    db = davies_bouldin_score(X_scaled, kmeans.labels_)
    print(k, db)
    db_list.append(db)
```

[ ]:
```python
plt.plot(range(2, 11), db_list, marker="o")
```

Same but with silhouette score:

[ ]:
```python
# silhouette Score

silhouette_list = []
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(X_scaled)
    silhouette = davies_bouldin_score(X_scaled, kmeans.labels_)
    print(k, silhouette)
    silhouette_list.append(silhouette)
```

[ ]:
```python
plt.plot(range(2, 11), silhouette_list, marker="o")
```

1.5.3 Use the best number of clusters

Fit the Kmeans model to the data with the best number of clusters:

[ ]:
```
kmeans = KMeans(n_clusters=6, random_state=42)
kmeans.fit(X_scaled)
labels = kmeans.predict(X_scaled)
```

Have a look to labels

[ ]:
```
labels
```

Use string values for labels:

[ ]:
```
labels = [labels_values[l] for l in labels]
```

Update _df dataset with labels:

[ ]:
```
_df = df.copy()
_df["labels"] = labels
_df
```

Have a look to A cluster:

[ ]:
```
_df.loc[_df.labels == "A"]
```

Have a look to B cluster:

[ ]:
```
_df.loc[_df.labels == "B"]
```

Etc etc…

[ ]:
```
_df.loc[_df.labels == "C"]
```

[ ]:
```
_df.loc[_df.labels == "D"]
```

[ ]:
```
_df.loc[_df.labels == "E"]
```

[ ]:
```
_df.loc[_df.labels == "F"]
```

[ ]:
```
_df.loc[_df.labels == "G"]
```

[ ]:
```
_df.loc[_df.labels == "H"]
```

Use boxplot to visualize the clusters:

[ ]:
```
for col in _df.select_dtypes(include="number").columns:
    sns.boxplot(data=_df, y=col, hue="labels")
    plt.show()
```

The D cluster is interesting:

[ ]:
```
_df.loc[_df.labels == "D"]
```

[ ]:
```

```

1.6 OPTIONAL Agglomerative Clustering (Hierarchical Clustering)

Create a model with the best number of clusters:

[ ]:
```
agc = AgglomerativeClustering(n_clusters=6)
agc
```

Fit the model to the data:

[ ]:
```
agc.fit(X_scaled)
```

Have a look to labels :

[ ]:
```
agc_class = agc.labels_
agc_class[:100]
```

Use string values for labels:

[ ]:
```
agc_class = [labels_values[l] for l in agc_class]
agc_class
```

Update _df dataset with new labels:

[ ]:
```
_df["labels"] = agc_class
_df
```

Have a look to A cluster:

[ ]:
```
_df.loc[_df.labels == "A"]
```

Have a look to B cluster:

[ ]:
```
_df.loc[_df.labels == "B"]
```

Etc etc…

[ ]:
```
_df.loc[_df.labels == "C"]
```

[ ]:
```
_df.loc[_df.labels == "D"]
```

[ ]:
```
_df.loc[_df.labels == "E"]
```

[ ]:
```
_df.loc[_df.labels == "F"]
```

Use Plotly to visualize the clusters:

[ ]:
```
for col in _df.select_dtypes(include="number").columns:
    fig = px.box(_df, y=col, color="labels")
    fig.show()
```

Plot the dendrogram

[ ]:
```
plt.figure(figsize=(10, 5))
plt.xlabel("sample index")
plt.ylabel("distance")
z = linkage(X_scaled, method="ward")
dendrogram(
    z,
    leaf_rotation=90,
    p=5,
    color_threshold=10,
    leaf_font_size=10,
    truncate_mode="level",
)
plt.tight_layout()
```