PCA for dim. reduction of geochemical data

Authors: Matt Divers, Savanna van Mesdag
Affiliation: University of Glasgow, Department of Geographical and Earth Sciences

This notebook imports .csv geochemical data, converts to a data array, and performs dimensional reduction by principal component analysis (PCA) using sklearn singular value decomposition (SVD) based method. The output is then visualised to evaluate any emerging patterns.

To run the notebook on new data seamlessly, the .csv should be structured as follows:

- A row for each datapoint.
- The first column for location (or some other descriptor).
- A column per measured molecule (for geochemical data), or other dimensionality.

If your data is in another format (including file format), the code will need to be amended to suit.

As a prerequisite for running this notebook, we recommend setting up an **anaconda** environment for Python 3 using the following commands:

> conda create -n geochem python=3.8
> conda activate geochem
>
> conda install -c conda-forge scikit-learn
> conda install -c conda-forge pandas
> conda install -c anaconda matplotlib
> conda install -c anaconda seaborn
> conda install -c anaconda jupyter
>
> cd "your/file/path/here"
>
> jupyter notebook

Import packages

In [ ]:
```
# set matplotlib backend
%matplotlib inline

import matplotlib # visualisation, plotting etc [1]
import matplotlib.pyplot as plt # visualisation, plotting etc [1]
import seaborn as sns # more plotting, better linkage with pandas [2]

import pandas as pd # load .csv and dataframe management [3]
import numpy as np # matrix mathematics [4]

import sklearn as skl # machine learning, i.e. PCA etc... [5]
from sklearn.decomposition import PCA # PCA [5]
```

Load data

In [ ]:
```
df = pd.read_csv('Plant_Chemistry_Major_and_Trace_Elements.csv') # read data as par
display(df) # display the table
```

Above, we can see all the data as a table. Each row represents a sample, and each column represents a dimension to the data. We want only the numerical data, so we reformat the data as a NumPy array and drop the first column representing location:

In [ ]:
```
# create np array of data excluding location
data = np.array(df.iloc[:, 2 : ]) # df.iloc[EVERY ROW, 3rd COLUMN : to LAST COLUMN]
print(data.shape) # print the shape of the data array
print(data.dtype) # print the data type
```

All looks well, data has 41 samples and 13 dimensions (i.e., the measured molecules). The data is also in floating point format, which is what is required as an input for scikit-learn PCA.

In [ ]:
```
# let's preserve a variable for number of samples (n_row) and number of dimensions
n_row, n_col = data.shape # as above, we see data.shape outputs a tuple. So we assi
```

In [ ]:
```
# plot data as heatmap
plt.figure(figsize = (8,6), dpi=600) # create figure
plt.title('Raw data as heatmap') # set title
# plot data as image
im = plt.pcolormesh(data, cmap='viridis', vmin=0, vmax=100) # use vmin and vmax to
plt.colorbar(mappable=im, ax=plt.gca(), label='weight %') # add colorbar

# get current axis
ax = plt.gca()

ax.set_box_aspect(1) # alter aspect ratio

# x ticks settings
x_tcks = np.arange(0, n_col, 1) # create an array of 1 to n dimensions
x_tck_labels = []
for comp in df.iloc[:,2:].columns: # use compositions as labels from data frame col
    x_tck_labels.append(comp)

ax.set_xticks(x_tcks) # create x tick for each dimension
ax.set_xticklabels(x_tck_labels, rotation = 90, ha = 'left') # add label for each

# y tick settings
y_tcks = np.arange(0, n_row, 1) # create an array of 1 to n samples
y_tck_labels = df.iloc[:, 0]

ax.set_yticks(y_tcks) # create x tick for each dimension
ax.set_yticklabels(y_tck_labels, ha = 'right', va = 'bottom', fontsize = 8) # add l

plt.grid(True, color='black')

fig = plt.gcf()
fig.savefig(f'figs/RawDataHeatmap.tif', format='tif')

plt.close(fig=fig)
```

From above, we see can start to see which molecules show the most significant variation. Now let's try to statistically simplify the complexity of the problem, and try to identify correlation in the dataspace.

Sklearn PCA

Initial run, we do not define a smaller number of components than those in the data. This will allow us to determine the optimum heuristically with a scree plot.

In [ ]:
```
# since scikit-learn PCA is an SVD-based implementation, we must first center the d
data_pca = skl.preprocessing.scale(data, axis=1) # set axis equal to the dimension
```

In [ ]:
```
pca = PCA(random_state = 0) # create our linear decomposition model, random state f
pca.fit(data_pca) # fit model to data

exp_var = pca.explained_variance_ratio_*100 # get explained variance (as a %)
csum_var = np.cumsum(exp_var) # cumulative explained variance

print(f'Total explained variance = {exp_var.sum()}') # check explained variance sum
```

We want to determine an optimal number of components in the data. This is done heuristically with a scree plot below:

In [ ]:
```
plt.figure(dpi=600)
plt.title('Cumulative explained variance plot')

plt.plot(exp_var, marker = 'o', label = 'successive') # plot non cumulative explain
plt.plot(csum_var, marker = 'x', label = 'cumulative') # plot cumulative explained

plt.xticks(np.arange(0,n_col,2), np.arange(1,n_col+1,2))

plt.xlabel('Principal components')
plt.ylabel('Explained variance (%)')

plt.legend() # add legend

plt.savefig(f'figs/PCA_ScreePlot.tif', format='tif')

plt.close(fig=plt.gcf())
```

In [ ]:
```
print(f'2 component explained variance = {csum_var[1]} %') # cumulative sum of 2 co
print(f'3 component explained variance = {csum_var[2]} %') # cumulative sum of 3 co
print(f'4 component explained variance = {csum_var[3]} %') # cumulative sum of 4 co
print(f'Maximum variance beyond 4 components = {exp_var[4:].max()} %') # if PCA is
```

We are looking for the "elbow" in the scree plot to determine the optimal number of components. This may exist at a few points (2, 3, or 4 components). We see that component 3 explains ~5% of variance, making it significant enough. 4th component and beyond only describe <1% of the total variance. Therefore, **4 components are optimal**.

Re-run PCA with optimal *n_comps*

In [ ]:
```
n_comps = 4 # choose optimal number of components
```

In [ ]:
```
pca = PCA(n_components = n_comps, random_state = 0) # create our dim reduction mode
pca.fit(data_pca) # fit model to data

exp_var = pca.explained_variance_ratio_*100 # get explained variance (as a %)
csum_var = np.cumsum(exp_var) # cumulative explained variance

print(f'Total explained variance = {exp_var.sum()}') # check explained variance sum
```

In [ ]:
```
scores = pca.transform(data_pca) # get score matrix
print(scores.shape)

loads = pca.components_# get loadings matrix
print(loads.shape)
```

Now our reduced dimensional model is divided into a scores matrix, and loadings matrix. Scores matrix is of shape [number of samples, number of principal components], whilst loadings matrix is of shape [number of components, number of input dimensions]. In this case, the scores matrix describes positive/negative correlation each sample has with each principal component, and the loadings matrix describes the positive/negative correlation each principal component has with each dimension.

We can consider the **loadings matrix** a **reduction in the number of samples** (i.e., if we only had *n_comps* number of "samples", which "samples" would explain most of the variation), whilst the **scores matrix** represents a **dimensionality reduction** - each sample can now be desceribed in only *n_comps* number of dimensions, rather than the original input number of dimensions.

Visualisation

First, let's plot the loadings matrix as though it were *n_comps* number of new samples. We still have the original number of dimensions, but now we have *n_comps* number of "samples" which explain almost the same amount of variation.

In [ ]:
```
#### visualise the loadings matrix
# plot data as heatmap
plt.figure(figsize = (8,6), dpi=600) # create figure
plt.title('PCA loadings matrix') # set title

plt.pcolormesh(loads, cmap = 'viridis', vmin=loads.min(), vmax=loads.max()) # plot
plt.colorbar(location='bottom', label='loading value') # add colorbar

# get current axis
ax = plt.gca()

ax.set_box_aspect(n_comps/n_col) # alter aspect ratio
ax.invert_yaxis() # flip y axis

# x ticks settings
x_tcks = np.arange(0, n_col, 1) # create an array of 1 to 13 (n dimensions)
x_tck_labels = []
for comp in df.iloc[:,2:].columns: # use compositions as labels from data frame col
    x_tck_labels.append(comp)

ax.set_xticks(x_tcks) # create x tick for each dimension
ax.set_xticklabels(x_tck_labels, rotation = 90, ha = 'left') # add label for each

# y tick settings
```

```
y_tcks = np.arange(0, n_comps, 1) # create an array of 1 to 3 (n components)
y_tck_labels = []
for i in range(1,n_comps+1):
    y_tck_labels.append(f'PC {i}')

ax.set_yticks(y_tcks) # create x tick for each dimension
ax.set_yticklabels(y_tck_labels, ha = 'right', va = 'top', fontsize = 14) # add lab

plt.grid(True, color='black')

fig = plt.gcf()
fig.savefig(f'figs/PCA4_LoadingsHeatmap.tif', format='tif')

plt.close(fig=fig)
```

This visualisation helps us understand the assumptions of PCA. The columns "MgO" to "BaO" exist in trace abundance only, and as such have very little weight when it comes to supplying variance to the dataset. The first 3 components instead are explaining:

- 1. High SiO2 but low CaO, everthing else very moderate.
- 1. High SiO2 and high CaO, slightly lower Al2O3, everything else very moderate.
- 1. Moderate amounts of everything except high Fe2O3.

Note: "high"/"low" is relative to the rest of the dataset, not as weight %.

Now let's see how that affects the distribution of our components in each of the original samples:

In [ ]:
```
# plot data as heatmap
plt.figure(figsize = (4,8), dpi=600) # create figure
plt.title('PCA scores matrix') # set title
# plot data as image
im = plt.pcolormesh(scores, cmap='viridis', vmin=scores.min(), vmax=scores.max())
plt.colorbar(mappable=im, ax=plt.gca(), label='PCA scores') # add colorbar

# get current axis
ax = plt.gca()

ax.set_box_aspect(n_row/(n_comps*2)) # alter aspect ratio

# x tick settings
x_tcks = np.arange(0, n_comps, 1) # create an array of 1 to 3 (n components)
x_tck_labels = []
for i in range(1, n_comps+1):
    x_tck_labels.append(f'PC {i}')

ax.set_xticks(x_tcks) # create x tick for each dimension
ax.set_xticklabels(x_tck_labels, rotation = 45, ha = 'left') # add label for each

# y tick settings
y_tcks = np.arange(0, n_row, 1) # create an array of 1 to n samples
y_tck_labels = df.iloc[:, 0]

ax.set_yticks(y_tcks) # create x tick for each dimension
ax.set_yticklabels(y_tck_labels, ha = 'right', va = 'bottom', fontsize = 8) # add l

plt.grid(True, color='black')

fig = plt.gcf()
fig.savefig(f'figs/PCA4_ScoresHeatmap.tif')
```

```
plt.close(fig=fig)
```

Finally let's try to better represent the dimensionally reduced dataspace with scatter plots and histograms:

In [ ]:
```
#### creating a colorblind friendly color map
# create a float RGBA (red, green, blue, alpha) color for each category
colors_cbf = np.array((np.array([35/255, 187/255, 20/255, 1]), np.array([216/255, 
                   np.array([255/255, 193/255, 7/255, 1]), np.array([0/255, 77/255,
                   np.array([102/255, 179/255, 225/255, 1]), np.array([77/255, 11/
```

In [ ]:
```
# we create a key system for the location name only, by using .split()
location = [] # create an empty list to append to

for comm in df.loc[:, 'Community']: # for each entry in the 'Community' column of t
    location.append(comm.split(sep='_')[0]) # the separator is '_'

# get list of unique locations
loc_unique = np.unique(location)
# create list of matplotlib markers for each category
marker_lst = ['o', '^', 's', '*', 'D', 'X']

for i in range(0, len(loc_unique)): # for each number of categories
    print(f'{loc_unique[i]} : {marker_lst[i]}') # print unique place name : marker
```

In [ ]:
```
#### Visualise the scores
# histogram of scores
fig, (ax1,ax2,ax3,ax4) = plt.subplots(1,4, figsize = (8,2))

ax1.hist(scores[:,0], bins = 25, color = 'red', edgecolor = 'black') # create hist
ax1.set_title('PCA score 1')

ax2.hist(scores[:,1], bins = 25, color = 'green', edgecolor = 'black') # create his
ax2.set_title('PCA score 2')

ax3.hist(scores[:,2], bins = 25, color = 'blue', edgecolor = 'black') # create hist
ax3.set_title('PCA score 3')

ax4.hist(scores[:,3], bins = 25, color = 'gray', edgecolor = 'black') # create hist
ax4.set_title('PCA score 4')

plt.tight_layout()

plt.show()

# scatter plots
fig, ((ax1,ax2,ax3),(ax4,ax5,ax6)) = plt.subplots(2,3, figsize = (9,5))

fig.suptitle('PCA scores')

sns.scatterplot(x=scores[:,0], y=scores[:,1], alpha = 0.75, hue = location, ax = ax
                palette=colors_cbf, edgecolor = 'black', legend=False, style=locati
ax1.set_xlabel('PC 1')
ax1.set_ylabel('PC 2')

sns.scatterplot(x=scores[:,0], y=scores[:,2], alpha = 0.75, hue = location, ax = ax
                palette=colors_cbf, edgecolor = 'black', legend=False, style=locati
ax2.set_xlabel('PC 1')
ax2.set_ylabel('PC 3')

sns.scatterplot(x=scores[:,0], y=scores[:,3], alpha = 0.75, hue = location, ax = ax
```

```
                palette=colors_cbf, edgecolor = 'black', legend=False, style=locati
ax3.set_xlabel('PC 1')
ax3.set_ylabel('PC 4')

sns.scatterplot(x=scores[:,1], y=scores[:,2], alpha = 0.75, hue = location, ax = ax
                palette=colors_cbf, edgecolor = 'black', legend=False, style=locati
ax4.set_xlabel('PC 2')
ax4.set_ylabel('PC 3')

sns.scatterplot(x=scores[:,1], y=scores[:,3], alpha = 0.75, hue = location, ax = ax
                palette=colors_cbf, edgecolor = 'black', legend=False, style=locati
ax5.set_xlabel('PC 2')
ax5.set_ylabel('PC 4')

sns.scatterplot(x=scores[:,2], y=scores[:,3], alpha = 0.75, hue = location, ax = ax
                palette=colors_cbf, edgecolor = 'black', legend=False, style=locati
ax6.set_xlabel('PC 3')
ax6.set_ylabel('PC 4')

plt.tight_layout()

fig = plt.gcf()
fig.savefig('figs/PCA4_ScoresScatter_subplots.tif', format='tif')

plt.close(fig=fig)
```

In [ ]:
```
# generate a custom legend for plt.legend()
leg = [] # create empty list
for i in range(0, len(loc_unique)): # for each unique location
    leg.append(matplotlib.lines.Line2D([], [], marker = marker_lst[i], color = colo
```

In [ ]:
```
# create plot for each combination of components
for a in range(0, n_comps): # from 0 to n_comps
    for b in range(0, n_comps): # from 0 to n_comps again
        if a<=b: # remove repeats and straight lines
            pass
        else: # otherwise create scatter plot
            plt.figure(dpi=600)
            ax = plt.gca()
            sns.scatterplot(x=scores[:,a], y=scores[:,b], alpha = 0.75, hue = locat
                            palette=colors_cbf, edgecolor = 'black', legend=False,
                            markers=marker_lst)
            # add custom legend
            plt.legend(leg, loc_unique, loc = (1.04, 0.18), markerscale = 1.5, font
                       labelspacing = 1)
            # set x and y labels
            plt.xlabel(f'PC {a+1}', fontsize=12, fontweight='bold')
            plt.ylabel(f'PC {b+1}', fontsize=12, fontweight='bold')

            ax.set_box_aspect(1)

            plt.savefig(f'figs/scores_scatter_PC{a+1}vsPC{b+1}.tif', format='tif')

            plt.close(fig=plt.gcf())
```

In [ ]:
```
#### now with "correct" aspect ratio
# create plot for each combination of components
for a in range(0, n_comps): # from 0 to n_comps
    for b in range(0, n_comps): # from 0 to n_comps again
        if a<=b: # remove repeats and straight lines
            pass
        else: # otherwise create scatter plot
            plt.figure(dpi=600)
```

```
            ax = plt.gca()
            ax.set_box_aspect((scores[:,b].max()-scores[:,b].min())/(scores[:,a].ma
            sns.scatterplot(x=scores[:,a], y=scores[:,b], alpha = 0.75, hue = locat
                            palette=colors_cbf, edgecolor = 'black', legend=False,
                            markers=marker_lst)
            # add custom legend
            plt.legend(leg, loc_unique, loc = (1.04, 0.18), markerscale = 1.5, font
                       labelspacing = 1)
            # set x and y labels
            plt.xlabel(f'PC {a+1}', fontsize=12, fontweight='bold')
            plt.ylabel(f'PC {b+1}', fontsize=12, fontweight='bold')

            plt.savefig(f'figs/scores_scatter_CorrectedAspectRatio_PC{a+1}vsPC{b+1}

            plt.close(fig=plt.gcf())
```

In [ ]:
```
# change backend for interactive plotting
%matplotlib notebook

fig = plt.figure(figsize=(8,8)) # create figure
ax = fig.add_subplot(projection='3d', ) # add subplot for 3d plotting

scatter = ax.scatter(scores[:,0], scores[:,1], scores[:,2], c=scores[:,3], cmap='vi
cb = plt.colorbar(mappable=scatter, ax=ax, location='bottom')
cb.set_label(label='PC 4', weight='bold')

# set axis limits
ax.set_xlim(scores.min(), scores.max())
ax.set_ylim(scores.min(), scores.max())
ax.set_zlim(scores.min(), scores.max())

# add axis labels
ax.set_xlabel('PC 1', fontweight='bold')
ax.set_ylabel('PC 2', fontweight='bold')
ax.set_zlabel('PC 3', fontweight='bold')

plt.show()
```

In [ ]:
```
%matplotlib notebook

fig = plt.figure(figsize=(8,8)) # create figure
ax = fig.add_subplot(projection='3d', ) # add subplot for 3d plotting

# set axis limits
ax.set_xlim(scores.min(), scores.max())
ax.set_ylim(scores.min(), scores.max())
ax.set_zlim(scores.min(), scores.max())

# add axis labels
ax.set_xlabel('PC 1', fontweight='bold')
ax.set_ylabel('PC 2', fontweight='bold')
ax.set_zlabel('PC 3', fontweight='bold')

idx = 0
for loc in location:
    loc_idx = 0
    for locU in loc_unique:
        if loc == locU:
            ax.scatter(scores[idx,0], scores[idx,1], scores[idx,2], color=colors_cb
                       marker=marker_lst[loc_idx], edgecolor='black', depthshade=Tr
            idx+=1
        else:
            loc_idx+=1
```

```
#scatter = ax.scatter(scores[:,0], scores[:,1], scores[:,2], c=scores[:,3], cmap='v
#cb = plt.colorbar(mappable=scatter, ax=ax, location='bottom')
#cb.set_label(label='PC 4', weight='bold')

plt.legend(leg, loc_unique, markerscale = 2, fontsize = 10, labelspacing = 1.25)

plt.show()
```

The scatter plots and histograms above highlight the fact that there is not enough samples for a robust cluster analysis. There are several areas where points are deviating from one another; however, it is hard to know if these points are noise in the data, or if with a larger dataset these would form groups of their own.

Export data

Convert loadings and score matrices back into a dataframe with the appropriate headers, and save as .csv.

In [ ]:
```
# x ticks settings
x_tcks = np.arange(0, n_col, 1) # create an array of 1 to 13 (n dimensions)
x_tck_labels = []
for comp in df.iloc[:,2:].columns: # use compositions as labels from data frame col
    x_tck_labels.append(comp)

# y tick settings
y_tcks = np.arange(0, n_comps, 1) # create an array of 1 to 3 (n components)
y_tck_labels = []
for i in range(1, n_comps+1):
    y_tck_labels.append(f'PC {i}')

# create pandas dataframe for exporting
df_loads = pd.DataFrame(data = loads, index = y_tck_labels, columns = x_tck_labels)
df_loads
```

In [ ]:
```
df_loads.to_csv('PlantChemistry_MajorTraceElements_PCA4_loadings.csv') # save as cs
```

In [ ]:
```
df_scores = pd.DataFrame(data = scores, columns = y_tck_labels) # create df with da
df_scores['Community'] = df.loc[:,'Community'] # add locations column

df_scores
```

In [ ]:
```
df_scores.to_csv('PlantChemistry_MajorTraceElements_PCA4_scores.csv') # save as csv
```

References

Plotting

> [1] J.D. Hunter. 2007. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, **9**(3), pp.90-95, doi:10.1109/MCSE.2007.55
>
> [2] M.L. Waskom. 2021. seaborn: statistical data vizualisation. *The Open Journal*, **6**(60), pp. 3021, doi:10.21105/joss.03021

Data management

[3] The pandas development team. 2020. pandas-dev/pandas: Pandas. *Zenodo*, doi:10.5281/zenodo.3509134

[4] C.R. Harris, K.J. Millman, S.J. van der Walt, *et al*. 2020. Array programming with NumPy. *Nature* **585**, pp. 357–362, doi:10.1038/s41586-020-2649-2

Principal component analysis

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, & E. Duchesnay. 2011. Scikit-learn: Machine Learning in {P}ython, *Journal of Machine Learning Research*, **12**, pp. 2825-2830.