DS620 Machine Learning and Deep Learning

HOS05A Principal Component Analysis

04/12/2021 Developed by Minh Nguyen

04/22/2021 Reviewed by Shanshan Yu

School of Technology & Computing @City University of Seattle (CityU)

Learning objectives

- Principal Component Analysis

Resources

- Galarnyk, M. (2021, February 4). PCA using Python (scikit-learn) - Towards Data Science. Medium. https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60
- scikit-learn: machine learning in Python — scikit-learn 0.24.1 documentation. (n.d.). Scikit-Learn. https://scikit-learn.org/stable/index.html
- Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. O'Reilly Media, Inc.
- Brownlee, J. (2018). Linear Algebra for Machine Learning. Machine Learning Mastery.

Introduction

In real life, Machine Learning problem can often involve thousands or even millions of columns of data. This not only greatly reduce the training time but also difficult to find a suitable model for such type of data. However, there are technique that can be used to reduce the number of columns of a dataset or in other words, reduce the dimensions of the data. Principal Component Analysis is one of the famous dimensionality reductions techniques where data with m-columns (features) is projected into a subspace with m or fewer columns, whilst retaining the essence of the original data. The PCA method can be described and implemented using the tools of linear algebra.

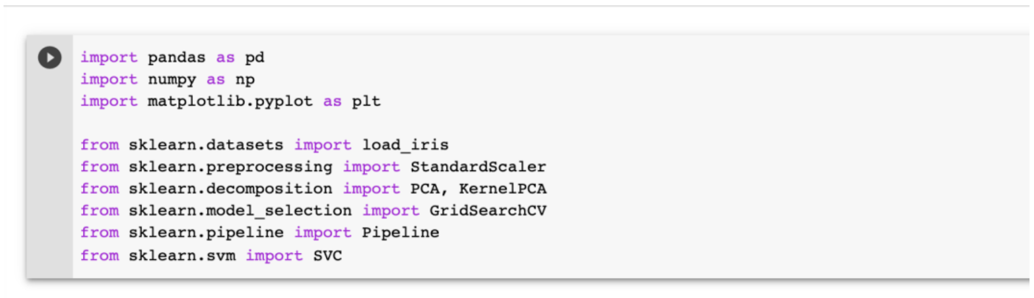
Dimensionality reduction is implemented during the Fine-tuning steps of your model. The goal of this phase is to optimize both the model and the data fed into the model.

Overview of a Machine Learning project

1. Get the data.
2. Discover and visualize the data to gain insights.
3. Prepare the data for Machine Learning algorithms.
4. Select a model and train it.
5. **Fine-tune your model.**
6. Present your solution.
7. Launch, monitor, and maintain your system.

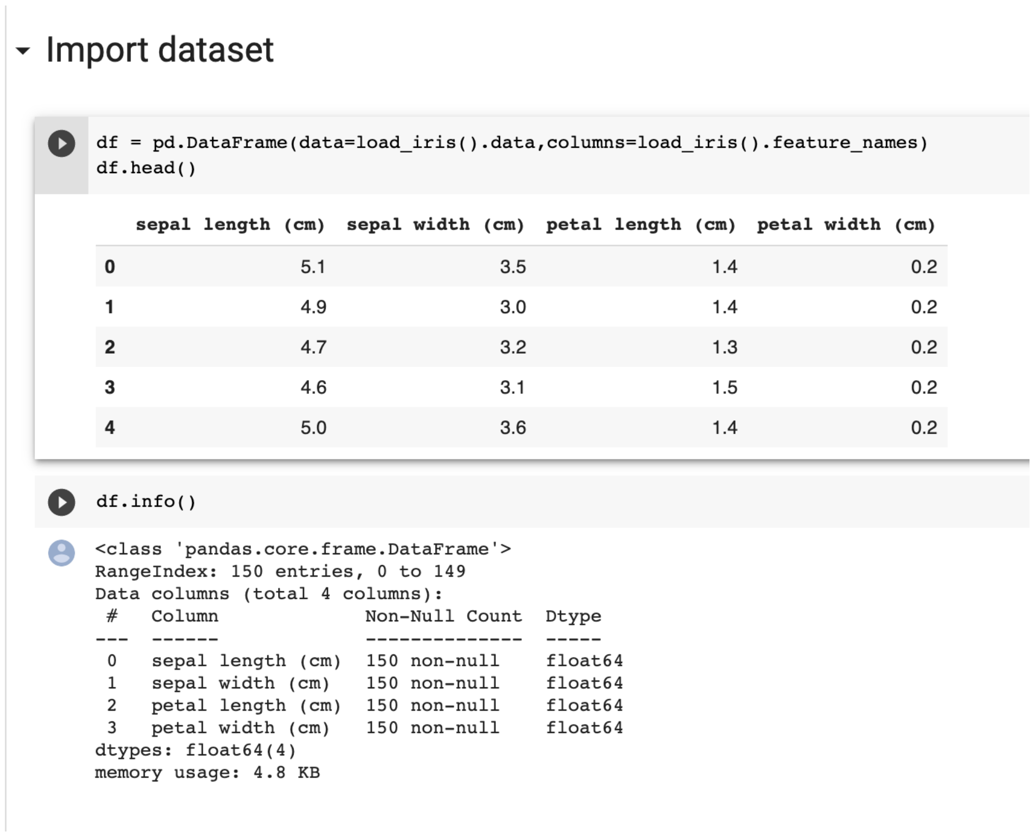
Preparing development environment

1. From Google Colab, create a new notebook, name it “PCA.ipynb”
2. Type the following codes to import libraries.



```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA, KernelPCA
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
```

3. Prepare the data.



Import dataset

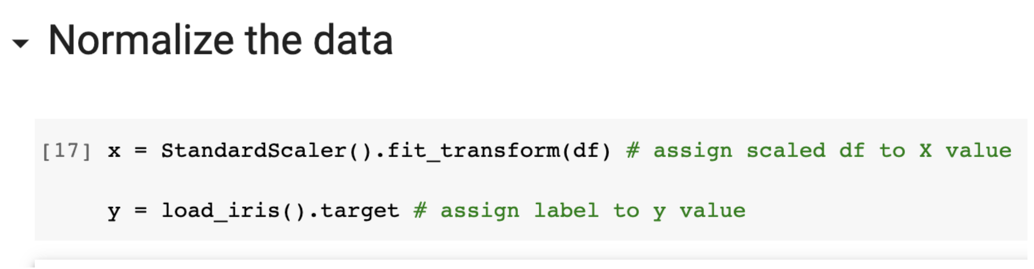
```
df = pd.DataFrame(data=load_iris().data,columns=load_iris().feature_names)
df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 4 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
dtypes: float64(4)
memory usage: 4.8 KB
```

4. Normalize the data.



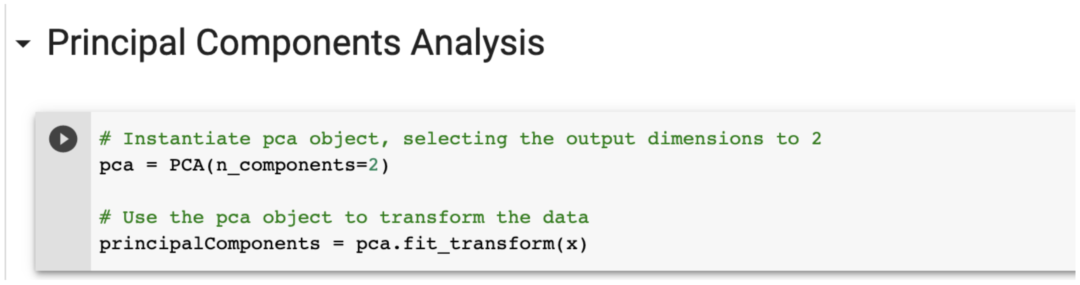
Normalize the data

```
[17] x = StandardScaler().fit_transform(df) # assign scaled df to X value

     y = load_iris().target # assign label to y value
```

I. Principal Component Analysis

Think of it as compressing the features matrix into a smaller one like how your computer do when zipping multiple files together. The result is a matrix with less column but still holds the similar values for the machine learning model. To use PCA, simply instantiate the PCA object and pass in the desired output column dimension to the n_components parameter.

- Type the following code for PCA.ipynb

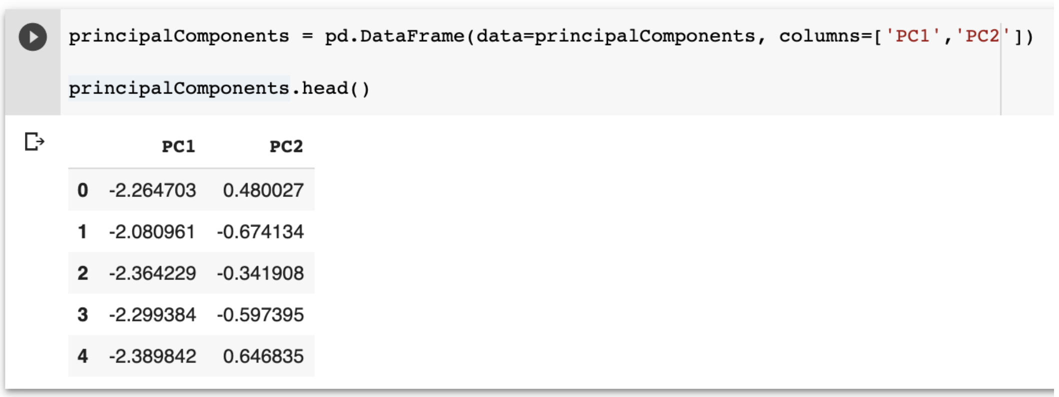


Principal Components Analysis

```
# Instantiate pca object, selecting the output dimensions to 2
pca = PCA(n_components=2)

# Use the pca object to transform the data
principalComponents = pca.fit_transform(x)
```

After we've created a new features matrix with 2 columns, let's create a new data frame to better visualize it.

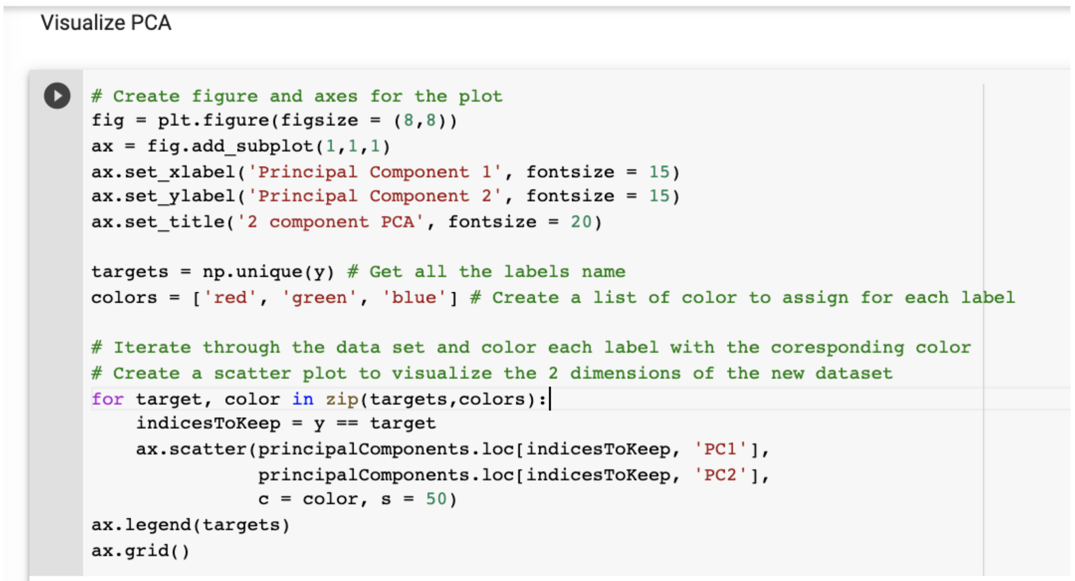
- Create the principal components data frame.



```
principalComponents = pd.DataFrame(data=principalComponents, columns=['PC1','PC2'])

principalComponents.head()
```

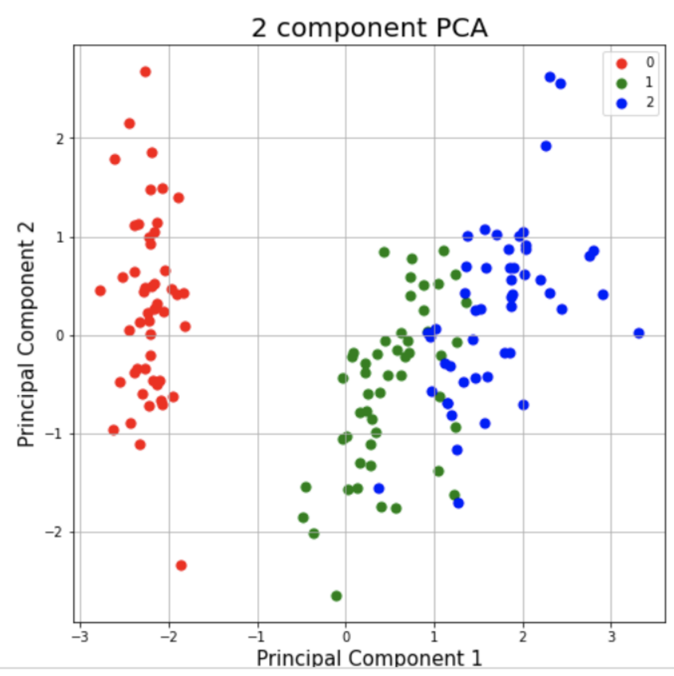
	PC1	PC2
0	-2.264703	0.480027
1	-2.080961	-0.674134
2	-2.364229	-0.341908
3	-2.299384	-0.597395
4	-2.389842	0.646835

- Now, let's visualize the data point in the new feature space of 2 dimensions.



Visualize PCA

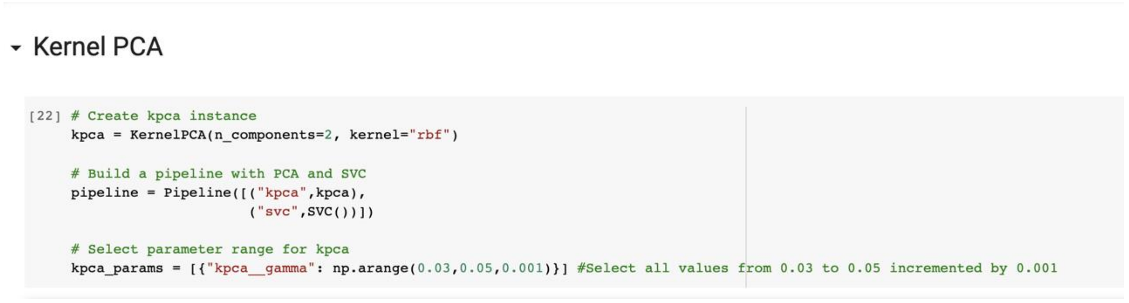
```
# Create figure and axes for the plot
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)

targets = np.unique(y) # Get all the labels name
colors = ['red', 'green', 'blue'] # Create a list of color to assign for each label

# Iterate through the data set and color each label with the coresponding color
# Create a scatter plot to visualize the 2 dimensions of the new dataset
for target, color in zip(targets,colors):
    indicesToKeep = y == target
    ax.scatter(principalComponents.loc[indicesToKeep, 'PC1'],
               principalComponents.loc[indicesToKeep, 'PC2'],
               c = color, s = 50)
ax.legend(targets)
ax.grid()
```



II. Kernel PCA

Kernel Trick is a concept that we’ve already learned from Support Vector Machine chapter, the rbf kernel can also be used for dimensionality reduction. In SVM, Kernel Trick maps instances into a high-dimensional space. The same concept can be applied to PCA making it possible to perform complex nonlinear projections for dimensionality reduction. The hyperparameter gamma is also used as the regularization parameter. Like other hyperparameter, this often requires tunning and trying between various values in order to find an optimal one. In this part, we will learn how to use Kernel PCA with SVM Classification model.

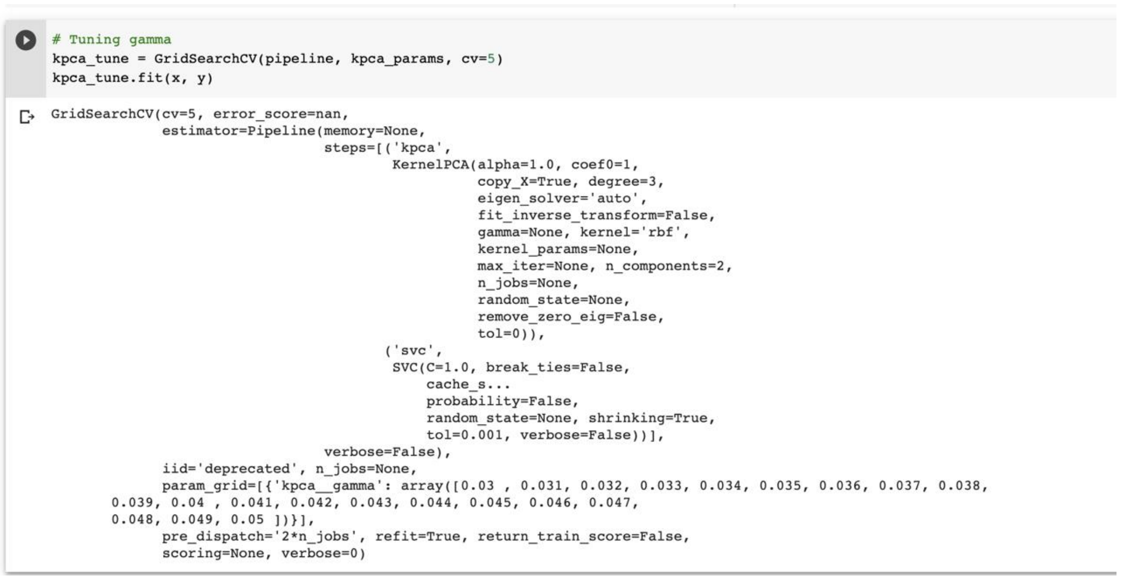
- Type the following code to PCA.ipynb.



Kernel PCA

```
# Create kpca instance
kpca = KernelPCA(n_components=2, kernel="rbf")

# Build a pipeline with PCA and SVC
pipeline = Pipeline([("kpca",kpca),
                     ("svc",SVC())])

# Select parameter range for kpca
kpca_params = [{"kpca__gamma": np.arange(0.03,0.05,0.001)}] #Select all values from 0.03 to 0.05 incremented by 0.001
```

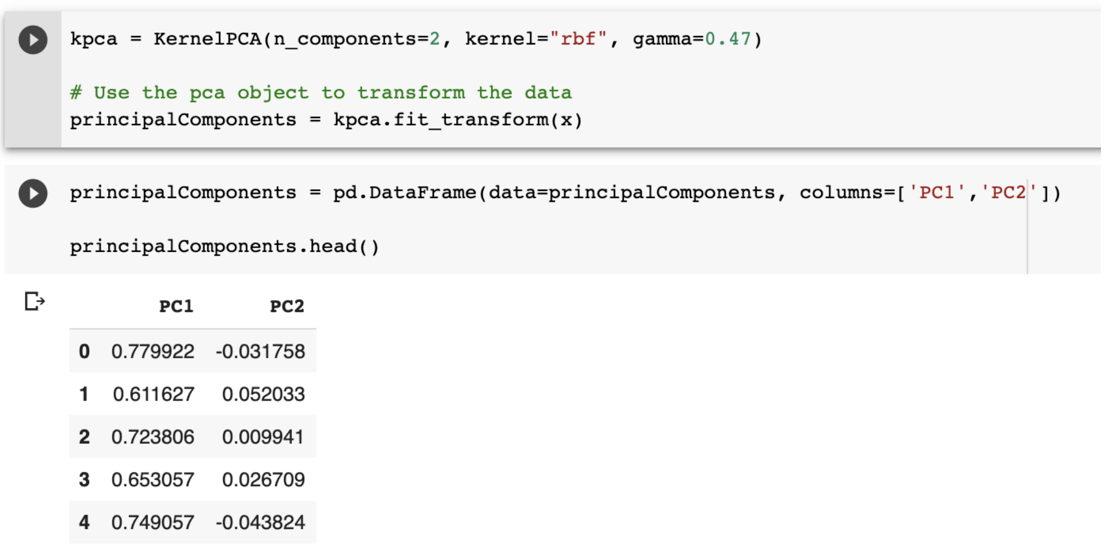
- Type the following code to tune the gamma hyperparameter of kpca.



```
# Tuning gamma
kpca_tune = GridSearchCV(pipeline, kpca_params, cv=5)
kpca_tune.fit(x, y)
```

```
GridSearchCV(cv=5, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('kpca',
                                        KernelPCA(alpha=1.0, coef0=1,
                                                  copy_X=True, degree=3,
                                                  eigen_solver='auto',
                                                  fit_inverse_transform=False,
                                                  gamma=None, kernel='rbf',
                                                  kernel_params=None,
                                                  max_iter=None, n_components=2,
                                                  n_jobs=None,
                                                  random_state=None,
                                                  remove_zero_eig=False,
                                                  tol=0)),
                                       ('svc',
                                        SVC(C=1.0, break_ties=False,
                                            cache_s...
                                            probability=False,
                                            random_state=None, shrinking=True,
                                            tol=0.001, verbose=False))],
                                verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid=[{'kpca__gamma': array([0.03 , 0.031, 0.032, 0.033, 0.034, 0.035, 0.036, 0.037, 0.038,
       0.039, 0.04 , 0.041, 0.042, 0.043, 0.044, 0.045, 0.046, 0.047,
       0.048, 0.049, 0.05 ])}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

- Type the following code to initialize the kpca model with the gamma hyperparameter.



```
kpca = KernelPCA(n_components=2, kernel="rbf", gamma=0.47)

# Use the pca object to transform the data
principalComponents = kpca.fit_transform(x)
```

```
principalComponents = pd.DataFrame(data=principalComponents, columns=['PC1','PC2'])

principalComponents.head()
```

	PC1	PC2
0	0.779922	-0.031758
1	0.611627	0.052033
2	0.723806	0.009941
3	0.653057	0.026709
4	0.749057	-0.043824

Push Your Work to GitHub

Download the notebook from Colab:

File -> PCA.ipynb

Move the downloaded file into your **Module5** working folder.

Open terminal and Navigate to the GitHub folder of this week HOS.

Make sure the assignment files on the subfolder Module5 of hos05a_YouGithubUserName folder, enter the following command to upload your work:

```
>>>> git add .
>>>> git commit -m “Submission for HOS05”
>>>> git push origin master
```