PCA using SVD

April 16, 2019

1 Statistical and Machine Learning (01.113) - HW4 Question 3

By: Adam Ilyas 1002010

In this problem, we will perform principal component analysis (PCA) on sklearn's diabetes dataset. Start by importing the required packages and load the dataset.

```
In [1]: import numpy as np
        from sklearn import decomposition, datasets
        import matplotlib.pyplot as plt
        import seaborn as sns
        import pandas as pd
        %matplotlib inline

In [2]: data = datasets.load_diabetes()
```

You can find out more on how to use sklearn's PCA module from:
https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

```
In [3]: X = data["data"]
        y = data["target"]
        feature_names = data["feature_names"]
        df = pd.DataFrame(X, columns=feature_names)
        df.head()
```

Out[3]:

	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.002592	0.019908	-0.017646
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493	-0.068330	-0.092204
2	0.085299	0.050680	0.044451	-0.005671	-0.045599	-0.034194	-0.032356	-0.002592	0.002864	-0.025930
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.034309	0.022692	-0.009362
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.002592	-0.031991	-0.046641

1.1 Write code to print the matrix V that will be used to transform the dataset, and print all the singular values.

Given a design matrix X, perform the following steps:

1. Normalize each column of X:

For each column of X, compute the mean and subtract it from each entry in the column. At the same time, compute the standard deviation and divide each entry in the column by it (this ensures that each column is normalized to have mean 0 and standard deviation 1). 2. Do SVD on X to yield $X = U\Sigma V^\top$ 3. Multiply X by V to get $XV = U\Sigma$ 4. Retain the first L columns of XV and remove the remaining columns.

```
In [4]: def StandardScaler(X):
            """
            X: 2d array
            Returns
                X with each columns scaled by the column mean and column std
            """
            N, d = X.shape
            scaled_data = np.zeros((N, d))
            for col_index in range(10):
                current_column = X[:, col_index]
                col_mean = np.mean(current_column)
                col_std = np.std(current_column)
                scaled_data[:, col_index] = [(el - col_mean)/col_std for el in current_column]
            return scaled_data

In [5]: X_scaled = StandardScaler(X)
        u, s, vh = np.linalg.svd(X_scaled)

In [6]: print("Matrix V: ")
        V = vh.T
        print(V)
```

```
Matrix V:
[[-0.21643101  0.04437151  0.49466811 -0.4140095  -0.68686389  0.2258505
  -0.10953821  0.01493468 -0.00810057 -0.00326309]
 [-0.18696711 -0.38654811 -0.10685833 -0.67986052  0.37345612 -0.04173103
  -0.06760551  0.44293966  0.00210552 -0.00366069]
 [-0.3031625  -0.15628061  0.1675317   0.49982533  0.12935936  0.4031419
  -0.51985787  0.39294187 -0.04237751 -0.00824809]
 [-0.2717397  -0.13825564  0.51356804 -0.01966734  0.48689014  0.27276274
   0.32064908 -0.47736435 -0.0271941   0.00322111]
 [-0.34325493  0.57302669 -0.0685867  -0.06839533  0.12917415 -0.00540864
   0.07364908  0.12941351  0.04203984 -0.70977447]
 [-0.35186062  0.45593985 -0.26969438 -0.16777384  0.11673143  0.1332572
  -0.23054011 -0.19131121  0.35931549  0.56319605]
 [ 0.28243639  0.50624287  0.38602787 -0.07602005  0.24499115 -0.1063716
```

```
 -0.00753445  0.32463641 -0.48124771  0.31744413]
[-0.42883325 -0.06818423 -0.38068121  0.0079212  -0.14364377  0.0339454
  0.07123619 -0.18058834 -0.77381656  0.09059464]
[-0.37861731 -0.0261893   0.0636315   0.26442742 -0.1516611  -0.17873005
  0.64731345  0.44966002  0.18945947  0.26446735]
[-0.32218282 -0.0849466   0.27684271  0.08708624  0.03138792 -0.80506447
 -0.35727279 -0.1666087   0.01527381 -0.0026109 ]]
```

```
In [7]: print('Singular Values:')
        print(s)
```

```
Singular Values:
[42.17466853 25.68276971 23.08755816 20.55043949 17.10806903 16.32182255
 15.39999097 13.84514267  5.88365535  1.94518745]
```

1.2 Now perform PCA on the dataset and print out the 3 most important components for the first 10 data-points.

```
In [8]: def pca_transform(X, n_components=2):
            X_scaled = StandardScaler(X)
            u, s, vh = np.linalg.svd(X_scaled)
            V = vh.T
            XV = X_scaled@V
            return XV[:, :n_components]
```

```
In [9]: X_pca = pca_transform(X, n_components=3)
        X_pca.shape
```

```
Out[9]: (442, 3)
```

```
In [10]: # first 10 data points
         X_pca[:10]
```

```
Out[10]: array([[-0.58720767, -1.94682793,  0.58923299],
                [ 2.83161209,  1.37208454,  0.02791506],
                [-0.27214757, -1.63489803,  0.73927034],
                [-0.04931005,  0.38225333, -2.01303697],
                [ 0.75645071,  0.81196754, -0.05725853],
                [ 3.96635524, -0.38105927, -0.33738317],
                [ 1.99378667, -0.80553831, -0.71219915],
                [-2.07586704,  1.82792114,  0.52492352],
                [-0.60303259, -0.88125266, -0.07671973],
                [ 0.21215262, -0.49290431, -0.81436321]])
```