Module1

July 22, 2022

1 Homework 1: PCA

1.1 Problem 1 - Principal Component Analysis

In this problem you'll be implementing Dimensionality reduction using Principal Component Analysis technique.

The gist of PCA Algorithm to compute principal components is follows: - Calculate the covariance matrix X of data points. - Calculate eigenvectors and corresponding eigenvalues. - Sort the eigenvectors according to their eigenvalues in decreasing order. - Choose first k eigenvectors which satisfies target explained variance. - Transform the original data of shape m observations times n features into m observations times k selected features.

The skeleton for the *PCA* class is below. Scroll down to find more information about your tasks.

[1]:
```
import math
import pickle
import gzip
import numpy as np
import pandas
import matplotlib.pylab as plt
%matplotlib inline
```

[2]:
```
from sklearn.preprocessing import StandardScaler
```

[3]:
```
arr = np.array([2,4,2,1,6])
np.sort(arr)[::-1]
print(arr / sum(arr))
arr = np.sort(arr)[::-1]
print(arr / sum(arr))
```

```
[0.13333333 0.26666667 0.13333333 0.06666667 0.4       ]
[0.4        0.26666667 0.13333333 0.13333333 0.06666667]
```

[4]:
```
class PCA:
    def __init__(self, target_explained_variance=None):
        """
        explained_variance: float, the target level of explained variance
```

```
        """
        self.target_explained_variance = target_explained_variance
        self.feature_size = -1

    def standardize(self, X):
        """
        standardize features using standard scaler
        :param X: input data with shape m (# of observations) X n (# of␣
 ↪features)
        :return: standardized features (Hint: use skleanr's StandardScaler.␣
 ↪Import any library as needed)
        """
        # your code here
        scaler = StandardScaler()
        return scaler.fit_transform(X)

        #Another way to standardize
#           (X-np.tile(X.mean(axis=0),(X.shape[0],1)))/np.tile(X.std(axis=0),(X.
 ↪shape[0],1))


    def compute_mean_vector(self, X_std):
        """
        compute mean vector
        :param X_std: transformed data
        :return n X 1 matrix: mean vector
        """
        # your code here

        return X_std.mean(axis=0)



    def compute_cov(self, X_std, mean_vec):
        """
        Covariance using mean, (don't use any numpy.cov)
        :param X_std:
        :param mean_vec:
        :return n X n matrix:: covariance matrix
        """
        # your code here

        n = X_std.shape[0]
        new_X =  X_std - mean_vec

        return np.dot(np.transpose(new_X), new_X) / (n-1)
```

```python
    def compute_eigen_vector(self, cov_mat):
        """
        Eigenvector and eigen values using numpy. Uses numpy's eigenvalue␣
↪function
        :param cov_mat:
        :return: (eigen_values, eigen_vector)
        """
        # your code here
        return np.linalg.eig(cov_mat)


    def compute_explained_variance(self, eigen_vals):
        """
        sort eigen values and compute explained variance.
        explained variance informs the amount of information (variance)
        can be attributed to each of  the principal components.
        :param eigen_vals:
        :return: explained variance.
        """
        # your code here

        eigen_vals = np.sort(eigen_vals)[::-1]



        return eigen_vals / np.sum(eigen_vals)




    def cumulative_sum(self, var_exp):
        """
        return cumulative sum of explained variance.
        :param var_exp: explained variance
        :return: cumulative explained variance
        """
        return np.cumsum(var_exp)

    def compute_weight_matrix(self, eig_pairs, cum_var_exp):
        """
        compute weight matrix of top principal components conditioned on target
        explained variance.
        (Hint : use cumilative explained variance and target_explained_variance␣
↪to find
        top components)
```

```
        :param eig_pairs: list of tuples containing eigenvalues and␣
↪eigenvectors,
        sorted by eigenvalues in descending order (the biggest eigenvalue and␣
↪corresponding eigenvectors first).
        :param cum_var_exp: cumulative expalined variance by features
        :return: weight matrix (the shape of the weight matrix is n X k)
        """
        # your code here
        weight_matrix = np.ones((self.feature_size, 1))
        for i in range(len(eig_pairs)):
            if cum_var_exp[i] < self.target_explained_variance:
                weight_matrix[i] = eig_pairs[i][1]

        return weight_matrix



    def transform_data(self, X_std, matrix_w):
        """
        transform data to subspace using weight matrix
        :param X_std: standardized data
        :param matrix_w: weight matrix
        :return: data in the subspace
        """
        return X_std.dot(matrix_w)

    def fit(self, X):
        """
        entry point to the transform data to k dimensions
        standardize and compute weight matrix to transform data.
        The fit functioin returns the transformed features. k is the number of␣
↪features which cumulative
        explained variance ratio meets the target_explained_variance.
        :param   m X n dimension: train samples
        :return  m X k dimension: subspace data.
        """

        self.feature_size = X.shape[1]

        # your code here
        x_std = self.standardize(X)

        mean_vect = self.compute_mean_vector(x_std)
        cov_matrix = self.compute_cov(x_std, mean_vect)
        eig_vals, eig_vects = self.compute_eigen_vector(cov_matrix)
```

```
        eig_pairs = []
        for i in range(len(eig_vals)):
            eig_pairs.append((np.abs(eig_vals[i]), eig_vects[:,i]))

        var = self.compute_explained_variance(eig_vals)
        var_sum = self.cumulative_sum(var)

        w_matrix = self.compute_weight_matrix(eig_pairs, var_sum)

        print(len(matrix_w),len(matrix_w[0]))
        return self.transform_data(X_std=x_std, matrix_w=matrix_w)
```

[**PART A**] Your task involves implementing helper functions to compute *mean, covariance, eigenvector and weights.*

complete `fit()` to using all helper functions to find reduced dimension data.

Run PCA on *fashion mnist dataset* to reduce the dimension of the data.

fashion mnist data consists of samples with *784 dimensions.*

Report the reduced dimension k for target explained variance of **0.99**

[5]:
```
X_train = pickle.load(open('./data/fashionmnist/train_images.pkl','rb'))
y_train = pickle.load(open('./data/fashionmnist/train_image_labels.pkl','rb'))

X_train = X_train[:1500]
y_train = y_train[:1500]
```

[6]:
```
# pca_.compute_mean_vector(X_train)
```

[7]:
```
pca_handler = PCA(target_explained_variance=0.99)
X_train_updated = pca_handler.fit(X_train)
```

```
      ␣
↪---------------------------------------------------------------------------

      ValueError                                Traceback (most recent call␣
↪last)

      <ipython-input-7-678395cd90d4> in <module>
        1 pca_handler = PCA(target_explained_variance=0.99)
  ----> 2 X_train_updated = pca_handler.fit(X_train)


      <ipython-input-4-eb1235992626> in fit(self, X)
      143         var_sum = self.cumulative_sum(var)
      144
```

```
--> 145         w_matrix = self.compute_weight_matrix(eig_pairs, var_sum)
    146 
    147         print(len(matrix_w),len(matrix_w[0]))


    <ipython-input-4-eb1235992626> in compute_weight_matrix(self, eig_pairs, cum_var_exp)
    102         for i in range(len(eig_pairs)):
    103             if cum_var_exp[i] < self.target_explained_variance:
--> 104                 weight_matrix[i] = eig_pairs[i][1]
    105 
    106         return weight_matrix


    ValueError: could not broadcast input array from shape (784) into shape (1)
```

[ ]:

```

```