8. Dimensionality Reduction Using PCA

Extension Note: Project 2 due date has been extended by 2 days to **July 18 23:59UTC** (Note the UTC time zone).

PCA finds (orthogonal) directions of maximal variation in the data. In this problem we're going to project our data onto the principal components and explore the effects on performance.

You will be working in the files `part1/main.py` and `part1/features.py` in this problem

Project onto Principal Components

3.0/3.0 points (graded)

Fill in function `project_onto_PC` in **features.py** that implements PCA dimensionality reduction of dataset X.

Note that to project a given $n \times d$ dataset X into its k-dimensional PCA representation, one can use matrix multiplication, after first centering X:

$$\widetilde{X}V$$

where $\widetilde{X}$ is the centered version of the original data X and V is the $d \times k$ matrix whose columns are the top k eigenvectors of $\widetilde{X}^T\widetilde{X}$. This is because the eigenvectors are of unit-norm, so there is no need to divide by their length.

You are given the full principal component matrix V' as `pcs` in this function.

Available Functions: You have access to the NumPy python library as `np` and the function `center_data` which returns a centered version of the data, where each feature now has mean = 0

Correction on test.py:: We have updated the test for this function in `test.py` to remove some ambiguity. Please download a corrected verion of the project archive mnist.tar.gz or correct the function `check_project_onto_PC` in `test.py` to the following:

```
def check_project_onto_PC():
 ex_name = "Project onto PC"
 X = np.array([
     [1, 2, 3],
     [2, 4, 6],
     [3, 6, 9],
     [4, 8, 12],
 ]);
 pcs = features.principal_components(X)
 exp_res = np.array([
     [5.61248608, 0],
     [1.87082869, 0],
     [-1.87082869, 0],
     [-5.61248608, 0],
 ])
 n_components = 2
 if check_array(
         ex_name, features.project_onto_PC,
         exp_res, X, pcs, n_components):
     return
 log(green("PASS"), ex_name, "")
```

```
    """
    Given principal component vectors pcs = principal_components(X)
    this function returns a new data array in which each sample in X
    has been projected onto the first n_components principcal components.
    """
    # TODO: first center data using the centerData() function.

    # TODO: Return the projection of the centered dataset
```

```
#       on the first n_components principal components.
#       This should be an array with dimensions: n x n_components.
# Hint: these principal components = first n_components columns
#       of the eigenvectors returned by principal_components().
#       Note that each eigenvector is already be a unit-vector,
#       so the projection may be done using matrix multiplication.
centered_X = center_data(X)
return centered_X @ pcs[:,:n_components]
```

Press ESC then TAB or click outside of the code editor to exit

Correct

```
def project_onto_PC(X, pcs, n_components):
    """
    Given principal component vectors pcs = principal_components(X)
    this function returns a new data array in which each sample in X
    has been projected onto the first n_components principcal components.
    """
    centered_data = center_data(X)  # first center data.
    return np.dot(centered_data, pcs[:, range(n_components)])
```

Test results

See full output

```
CORRECT
```

See full output

Submit You have used 1 of 20 attempts

Answers are displayed within the problem

Note: we only use the training dataset to determine the principal components. It is **improper** to use the test dataset for anything except evaluating the accuracy of our predictive model. If the test data is used for other purposes such as selecting good features, it is possible to overfit the test set and obtain overconfident estimates of a model's performance.

Testing PCA

1.0/1.0 point (graded)

Use `project_onto_PC` to compute a 18-dimensional PCA representation of the MNIST training and test datasets, as illustrated in `main.py`.

Retrain your softmax regression model (using the original labels) on the MNIST training dataset and report its error on the test data, this time using these 18-dimensional PCA-representations rather than the raw pixel values.

If your PCA implementation is correct, the model should perform nearly as well when only given 18 numbers encoding each image as compared to the 784 in the original data (error on the test set using PCA features should be around 0.15). This is because PCA ensures these 18 feature values capture the maximal amount of variation from the original 784-dimensional data.

Error rate for 18-dimensional PCA features = 0.1483 ✓ **Answer:** 0.1483

Submit You have used 1 of 5 attempts

Answers are displayed within the problem

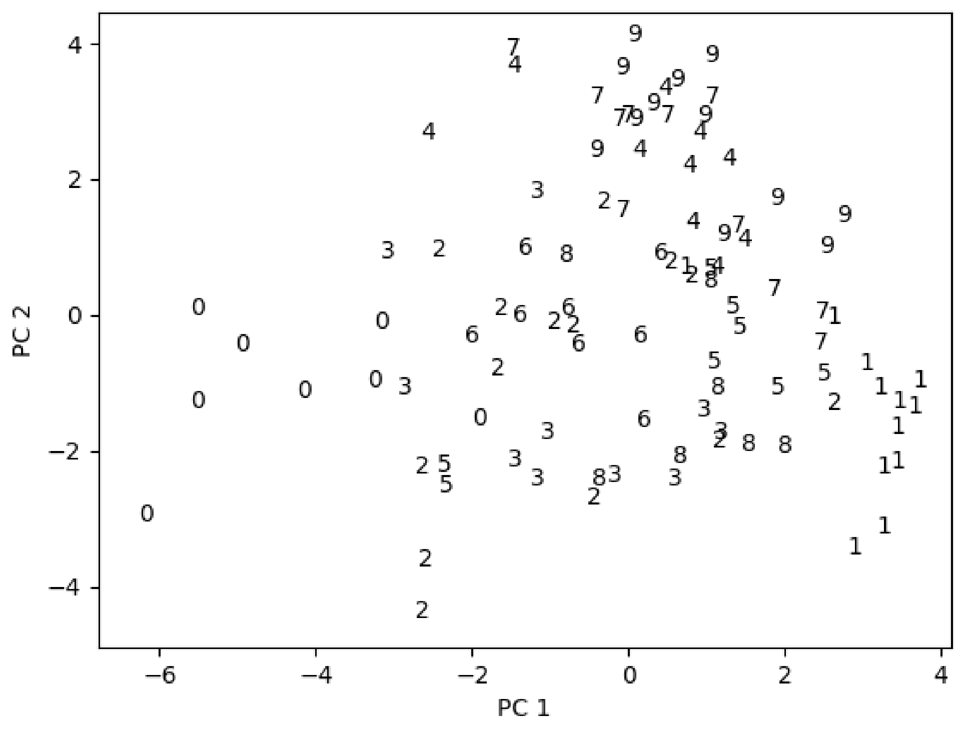
Testing PCA (continued)

1.0/1.0 point (graded)

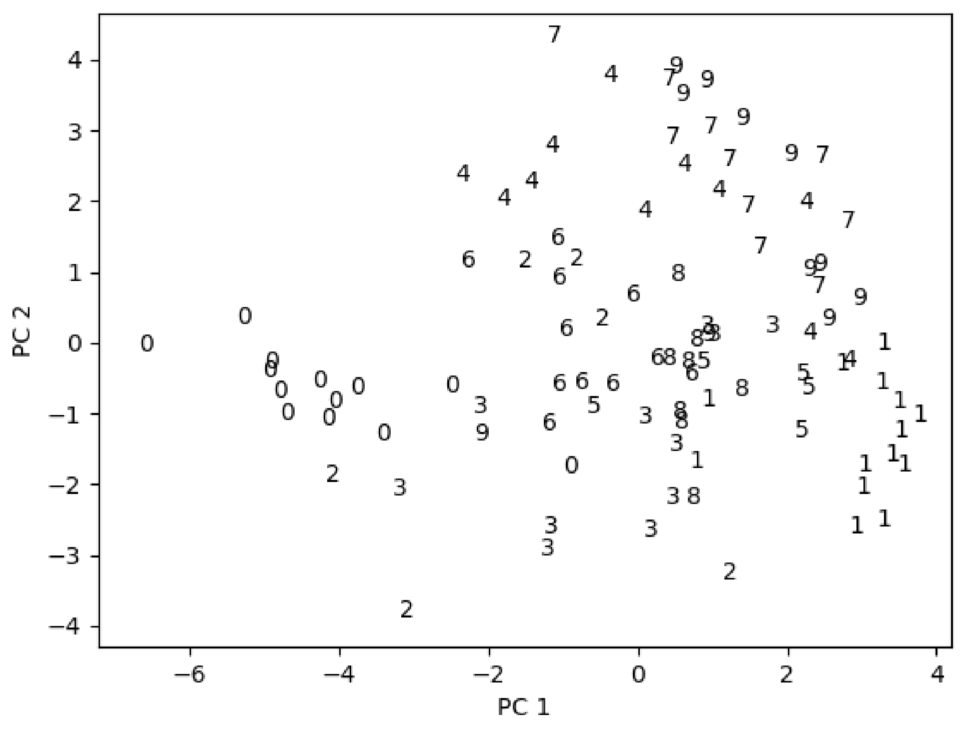
Use `plot_PC` in **main.py** to visualize the first 100 MNIST images, as represented in the space spanned by the first 2 principal components of the training data.

What does your PCA look like?

○

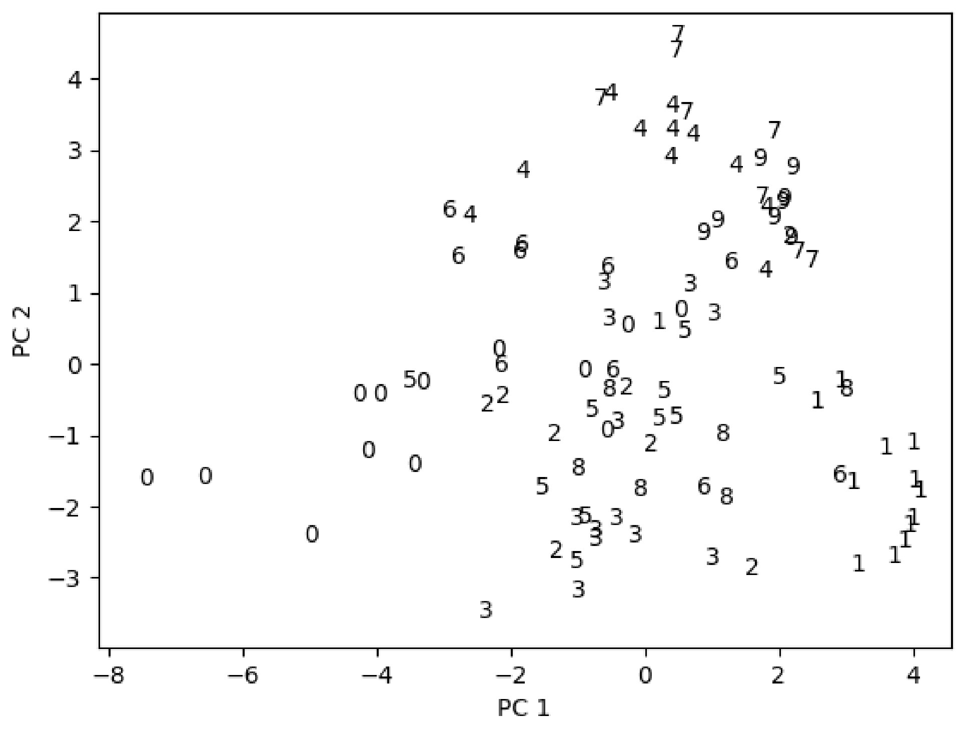

PC 1
PC 2

◉

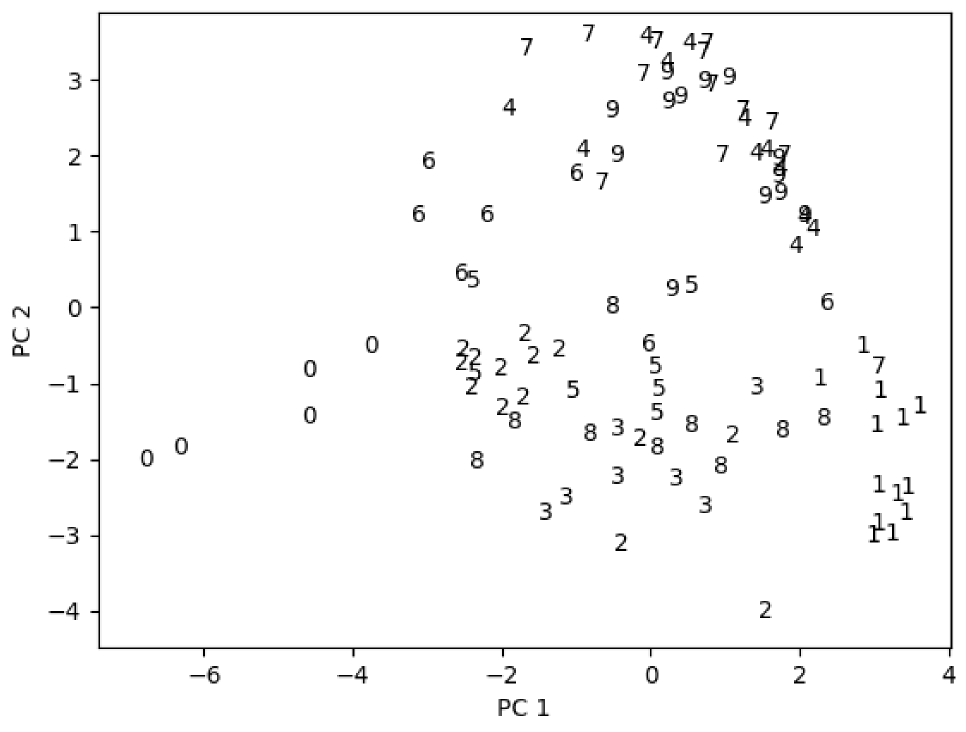

PC 1
PC 2

✓

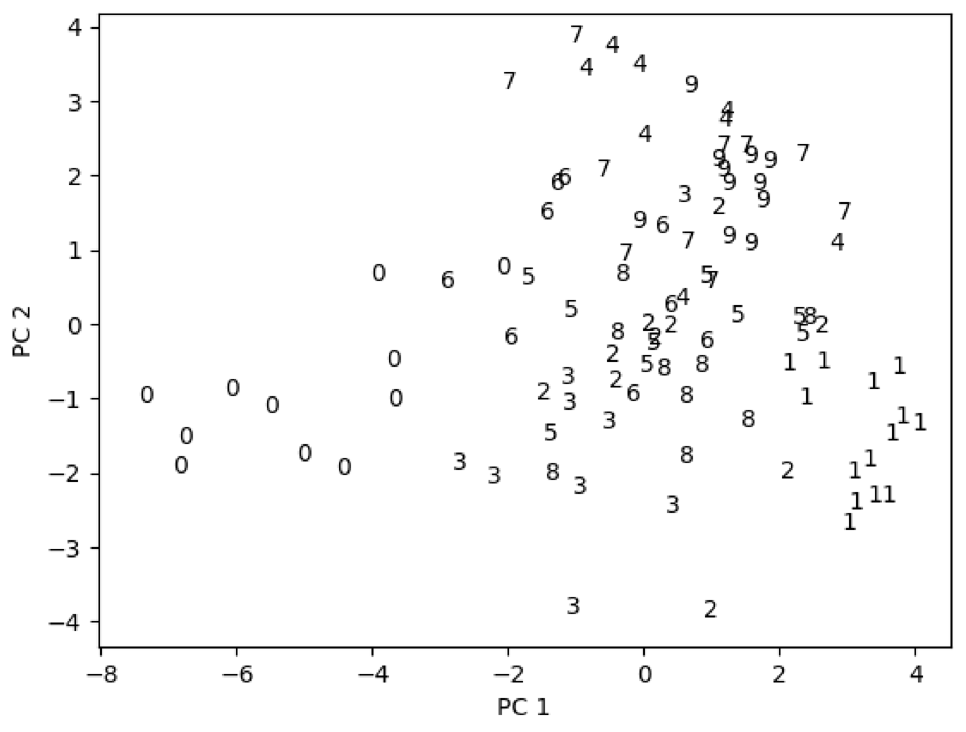
○


PC 1
PC 2

○



○



Use the calls to `plot_images()` and `reconstruct_PC` in **main.py** to plot the reconstructions of the first two MNIST images (from their 18-dimensional PCA-representations) alongside the originals.

Submit You have used 1 of 2 attempts

Answers are displayed within the problem

Remark: Two dimensional PCA plots offer a nice way to visualize some global structure in high-dimensional data, although approaches based on nonlinear dimension reduction may be more insightful in certain cases. Notice that for our data, the first 2 principal components are insufficent for fully separating the different classes of MNIST digits.

Discussion

Show Discussion

Topic: Unit 2 Nonlinear Classification, Linear regression, Collaborative Filtering (2 weeks):Project 2: Digit recognition (Part 1) / 8. Dimensionality Reduction Using PCA