8. Dimensionality Reduction Using PCA

Project due Mar 15, 2023 08:59 -03 Completed

PCA finds (orthogonal) directions of maximal variation in the data. In this problem we're 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 prob

Project onto Principal Components

3.0/3.0 points (graded)

Fill in function `project_onto_PC` in **features.py** that implements PCA dimensionality r

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

$$\widetilde{X} V$$

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

Function input:: You are given the full principal component matrix V' as `pcs` and the computed from the training data set as `feature_means` in this function. Note that `pc` are learned from the training data set, which should not be computed in this function u

Available Functions: You have access to the NumPy python library as `np`.

```
def project_onto_PC(X, pcs, n_components, feature_means):
    """
    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 feature_means
    # 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 = (X - feature_means)
```

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

Correct

Testing PCA

1.0/1.0 point (graded)

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

Retrain your softmax regression model (using the original labels) on the MNIST training error on the test data, this time using these 18-dimensional PCA-representations rathe values.

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

Error rate for 18-dimensional PCA features = 0.14739999999999998

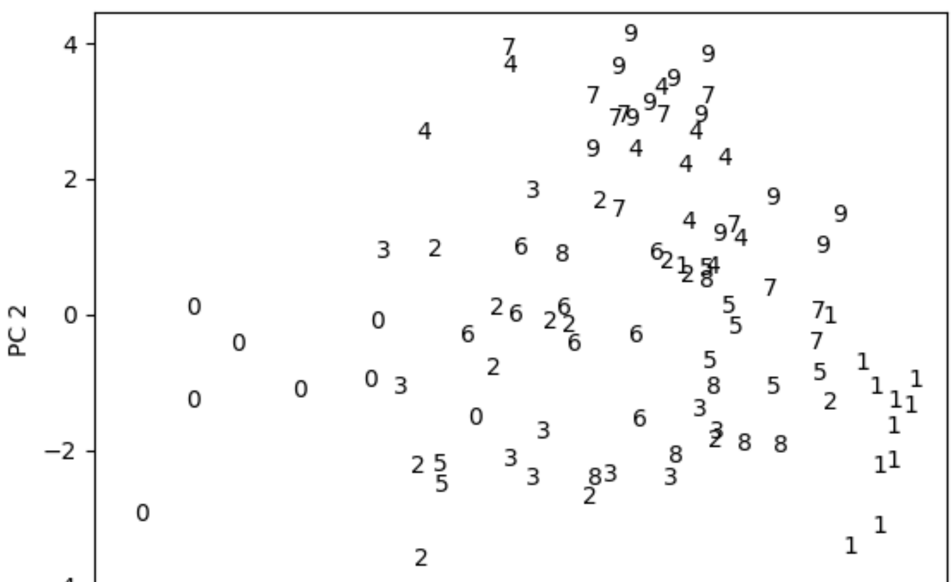
Submit You have used 1 of 5 attempts

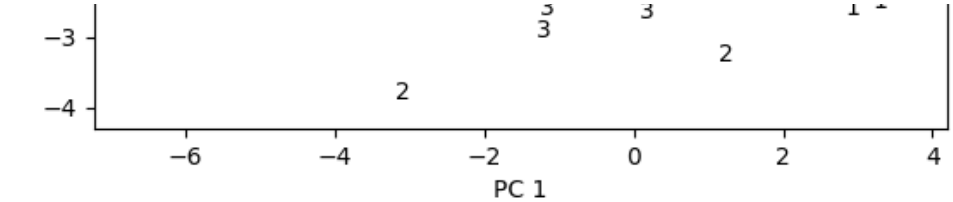
Testing PCA (continued)

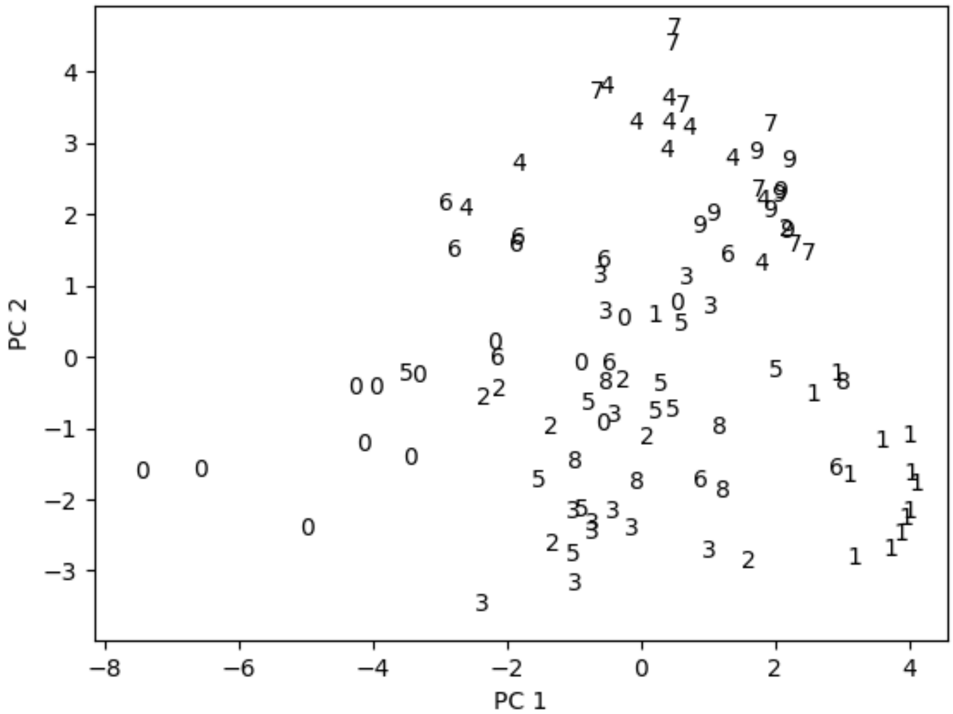
1.0/1.0 point (graded)

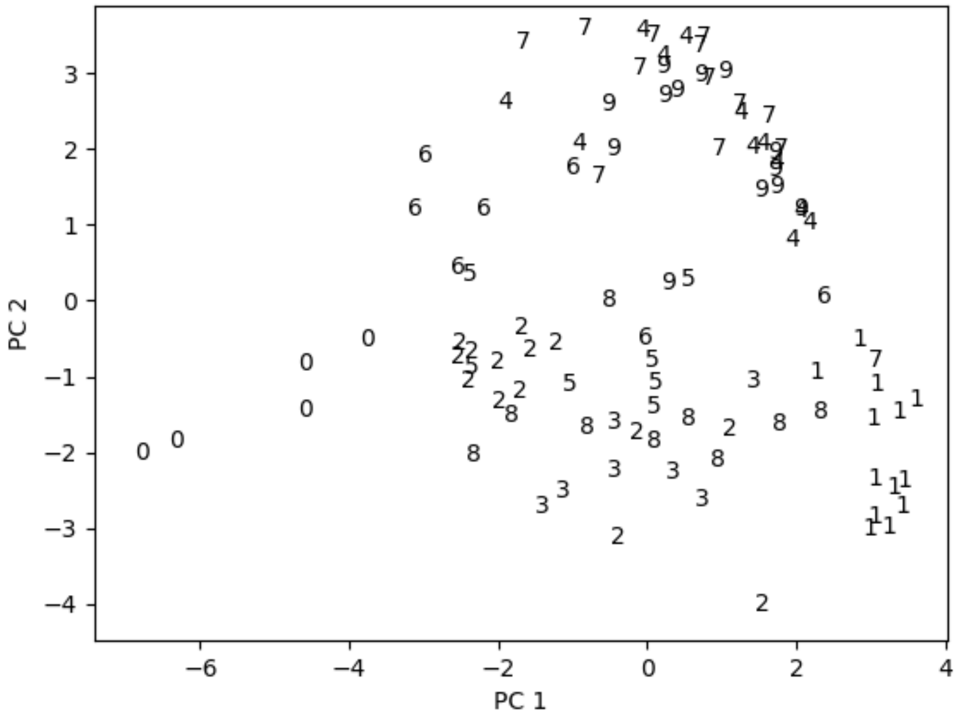
Use `plot_PC` in **main.py** to visualize the first 100 MNIST images, as represented in th first 2 principal components of the training data.

What does your PCA look like?




PC 1


PC 2
PC 1


PC 2
PC 1

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

Submit You have used 1 of 2 attempts

Remark: Two dimensional PCA plots offer a nice way to visualize some global structure
data, although approaches based on nonlinear dimension reduction may be more insigh
Notice that for our data, the first 2 principal components are insufficent for fully separa
classes of MN

Previous Next