



[Unit 2 Nonlinear Classification,](#)
[Linear regression, Collaborative](#)

[Course](#) > [Filtering \(2 weeks\)](#)

> [Project 2: Digit recognition \(Part 1\)](#) >

8. Dimensionality Reduction Using
PCA

Audit Access Expires May 11, 2020

You lose all access to this course, including your progress, on May 11, 2020.

Upgrade by Apr 1, 2020 to get unlimited access to the course as long as it exists on the site. **[Upgrade now](#)**

8. Dimensionality Reduction Using PCA

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

Correction to `features.py`: The version of the project archive earlier than March 22 contains an error in `part1/features.py`. You could either download the corrected [mnist.tar.gz](#) version now, or correct Lines 130, 136, and lines 147, 154-157 in `features.py` as follows:

```

128 - def plot_PC(X, pcs, labels):
128 + ###Correction note: Differing from the release, this function takes an extra input feature_means.
129 +
130 + def plot_PC(X, pcs, labels, feature_means):
129 131     """
130 132     Given the principal component vectors as the columns of matrix pcs,
131 133     this function projects each sample in X onto the first two principal components
132 134     and produces a scatterplot where points are marked with the digit depicted in
133 135     the corresponding image.
134 136     labels = a numpy array containing the digits corresponding to each image in X.
135 137     """
136 - pc_data = project_onto_PC(X, pcs, n_components=2)
138 + pc_data = project_onto_PC(X, pcs, n_components=2, feature_means=feature_means)
137 139     text_labels = [str(z) for z in labels.tolist()]
138 140     fig, ax = plt.subplots()
139 141     ax.scatter(pc_data[:, 0], pc_data[:, 1], alpha=0, marker=".")
144 146     plt.show()
145 147

```

```

147 - def reconstruct_PC(x_pca, pcs, n_components, X):
149 + ###Correction note: Differing from the release, this function takes an extra input feature_means.
150 +
151 + def reconstruct_PC(x_pca, pcs, n_components, X, feature_means):
148 152     """
149 153     Given the principal component vectors as the columns of matrix pcs,
150 154     this function reconstructs a single image from its principal component
151 155     representation, x_pca.
152 156     X = the original data to which PCA was applied to get pcs.
153 157     """
154 - feature_means = X - center_data(X)
155 - feature_means = feature_means[0, :]
156 158     x_reconstructed = np.dot(x_pca, pcs[:, range(n_components)].T) + feature_means
157 - return x_reconstructed
159 + return x_reconstructed + feature_means

```

Correction to main.py: The version of the project archive earlier than March 22 contains an error in part1/test.py. You could either download the corrected [mnist.tar.gz](#) version now, or correct Lines 174, 181, and 185 in main.py as follows:

```

171 171 # TODO: Use the plot_PC function in features.py to produce scatterplot
172 172 #       of the first 100 MNIST images, as represented in the space spanned by the
173 173 #       first 2 principal components found above.
174 174 - plot_PC(train_x[range(100), :], pcs, train_y[range(100)])
175 175 + plot_PC(train_x[range(000, 100), :], pcs, train_y[range(000, 100)], feature_means)#feature_means added since release
176 176
177 177 # TODO: Use the reconstruct_PC function in features.py to show
178 178 #       the first and second MNIST images as reconstructed solely from
179 179 #       their 18-dimensional principal component representation.
180 180 #       Compare the reconstructed images with the originals.
181 181 - firstimage_reconstructed = reconstruct_PC(train_pca[0, :], pcs, n_components, train_x)
182 182 + firstimage_reconstructed = reconstruct_PC(train_pca[0, :], pcs, n_components, train_x, feature_means)#feature_means ad
183 183 plot_images(firstimage_reconstructed)
184 184 plot_images(train_x[0, :])
185 185 - secondimage_reconstructed = reconstruct_PC(train_pca[1, :], pcs, n_components, train_x)
186 186 + secondimage_reconstructed = reconstruct_PC(train_pca[1, :], pcs, n_components, train_x, feature_means)#feature_means at
187 187 plot_images(secondimage_reconstructed)
188 188 plot_images(train_x[1, :])

```

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 using the mean learned from training data 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.

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

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

Correction on features.py and main.py: Please download a corrected version of

the project archive [mnist.tar.gz](#) or correct these files as described above before this problem.

```
1 def project_onto_PC(X, pcs, n_components, feature_means):
2     """
3     Given principal component vectors pcs = principal_components(X)
4     this function returns a new data array in which each sample in X
5     has been projected onto the first n_components principal components
6     """
7     # TODO: first center data using the feature_means
8     # TODO: Return the projection of the centered dataset
9     #         on the first n_components principal components.
10    #         This should be an array with dimensions: n x n_components.
11    # Hint: these principal components = first n_components columns
12    #         of the eigenvectors returned by principal_components().
13    #         Note that each eigenvector is already be a unit-vector,
14    #         so the projection may be done using matrix multiplication.
15    X_centered = X - feature_means
```

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

Correct

```
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 principal components.
    """
    centered_data = X - feature_means
    return np.dot(centered_data, pcs[:, range(n_components)])
```

Test results

CORRECT

[See full output](#)

[See full output](#)

Submit

You have used 2 of 25 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 = 0.14739999999999998



Answer: 0.1474

Submit

You have used 1 of 5 attempts

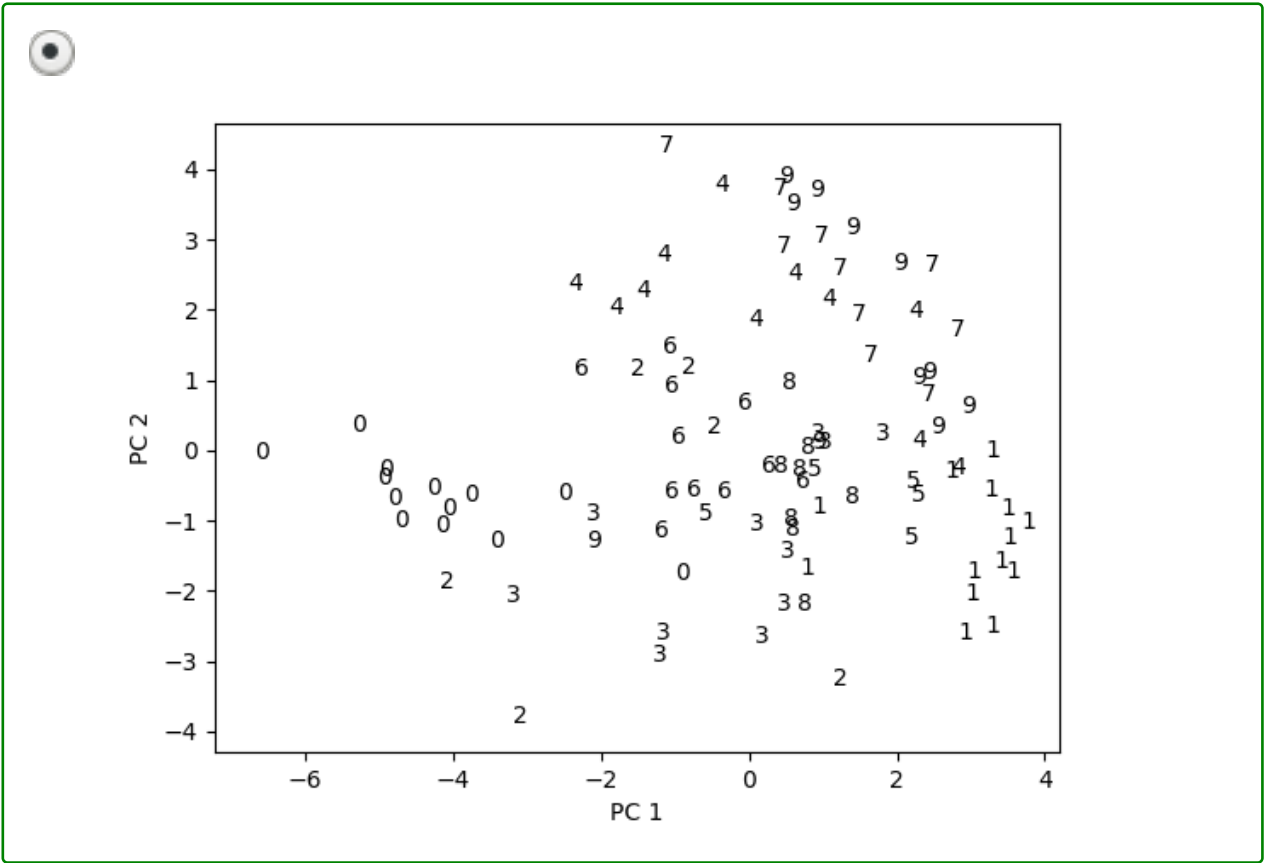
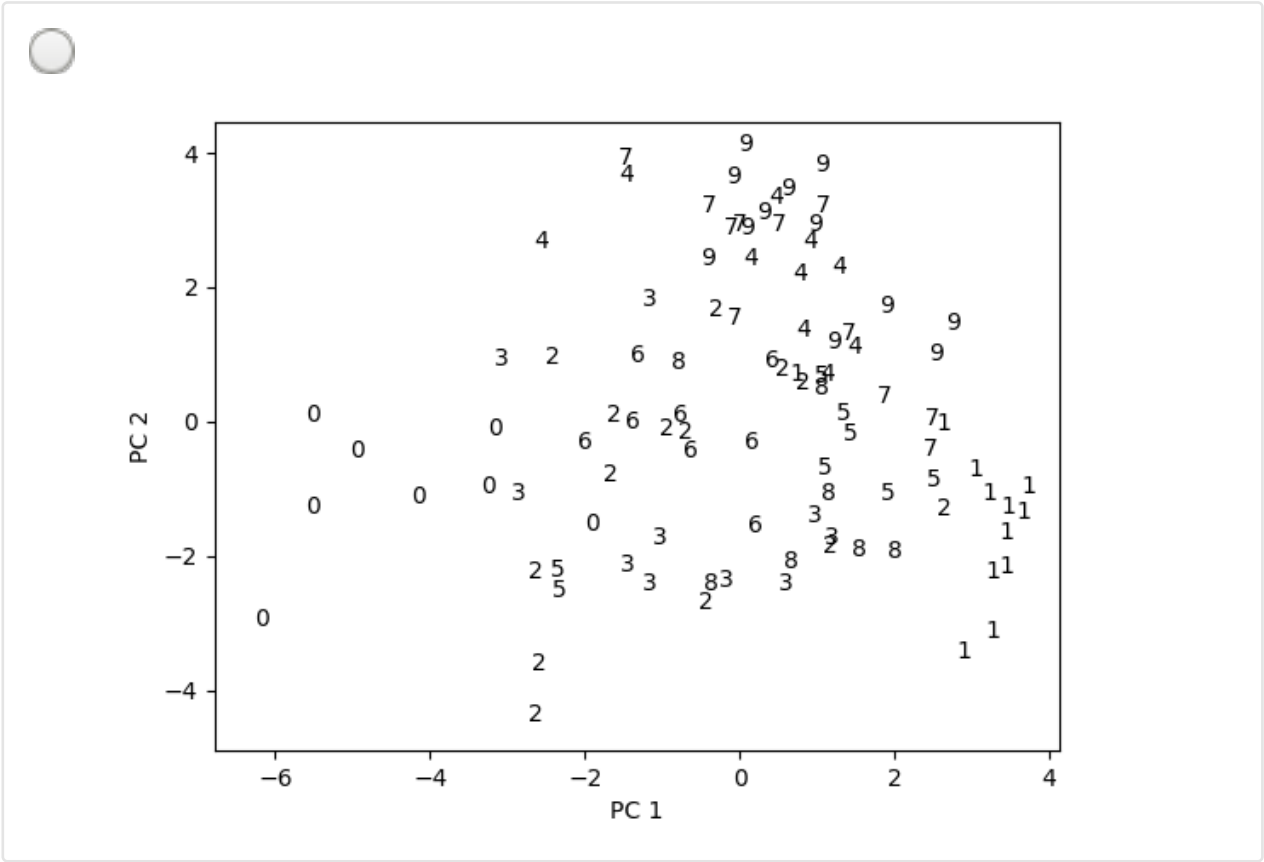
 Answers are displayed within the problem

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?





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

i 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 insufficient for fully separating the different classes of MNIST digits.

Discussion

[Hide Discussion](#)

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

[Add a Post](#)[Show all posts](#)[by recent activity](#)

-  [Error correction for Project onto Principal Components](#)
[We have fixed the grader for Project onto Principal Components. Now you should be able to s...](#) 15
 [Pinned](#)  [Staff](#)
-  [Corrected plot_PC\(\) function in features.py.](#)
[When you get to the last question \(*Testing PCA \(continued*\)\), you're asked to create a plot. Un...](#) 6
 [Pinned](#)
-  [Sign of completion not coming](#)
[All the codes in the section have been completed and have been graded but the signature of c...](#) 2

?	<u>How to tackle PCA problem</u>	2
	<u>L&G, Can anybody give a hint on how to tackle this problem? I have no clue of what is what. Ty...</u>	
✓	<u>2 days before the due date: what's the correct project_onto_PC()?</u>	6
	<u>There are so many threads discussing what should be changed for the PCA and the Cubic ques...</u>	
✓	<u>[STAFF] Can you check my project_onto_PC function?</u>	2
	<u>I am passing the online grader and getting full marks on it. But, my error rate for the softmax r...</u>	
💬	<u>Projection onto PC - test.py WRONG</u>	5
	<u>It took me one day to try to understand what I was doing so wrong that I could not pass the te...</u>	
?	<u>[staff] Answer is Complex</u>	5
	<u>I am getting partial credit because in one of the test my matrix has real numbers but the grade...</u>	
💬	<u>[staff]Is there some error on the original code of "reconstruct_PC(x_pca, pcs, n_components, X)"?</u>	4
	<u>Hello, I got some error on def function "reconstruct_PC(x_pca, pcs, n_components, X)", it is pro...</u>	
?	<u>Why no increase in computational efficiency?</u>	1
	<u>Hj, I got everything correct but was wondering why it takes almost as much time to train the m...</u>	
?	<u>[STAFF] Testing PCA question</u>	2
	<u>While running main.py, had this weird problem: line 33, in <module> train_x_centered, feature...</u>	
🔍	<u>Using plot_images() to plot training data next to test data side by side</u>	

Learn About Verified Certificates

© All Rights Reserved