learn-co-curriculum / dsc-pca-and-digital-image-processing-lab Public

View license

2 stars 156 forks

Star Watch

Code Issues Pull requests Actions Projects Security Insights

solution ...

This branch is 4 commits ahead, 5 commits behind master.

h-parker fix grammar/spelling/typos ... on Feb 1, 2020 5

View code

README.md

Image Recognition with PCA - Lab

Introduction

In this lab, you'll explore the classic MNIST dataset of handwritten digits. While not as large as the previous dataset on facial image recognition, it still provides a 64-dimensional dataset that is ripe for feature reduction.

Objectives

In this lab you will:

- Use PCA to discover the principal components with images
- Use the principal components of a dataset as features in a machine learning model
- Calculate the time savings and performance gains of layering in PCA as a preprocessing step in machine learning pipelines

Load the data

Load the `load_digits` dataset from the `datasets` module of scikit-learn.

```
from sklearn.datasets import load_digits
data = load_digits()
print(data.data.shape, data.target.shape)
```

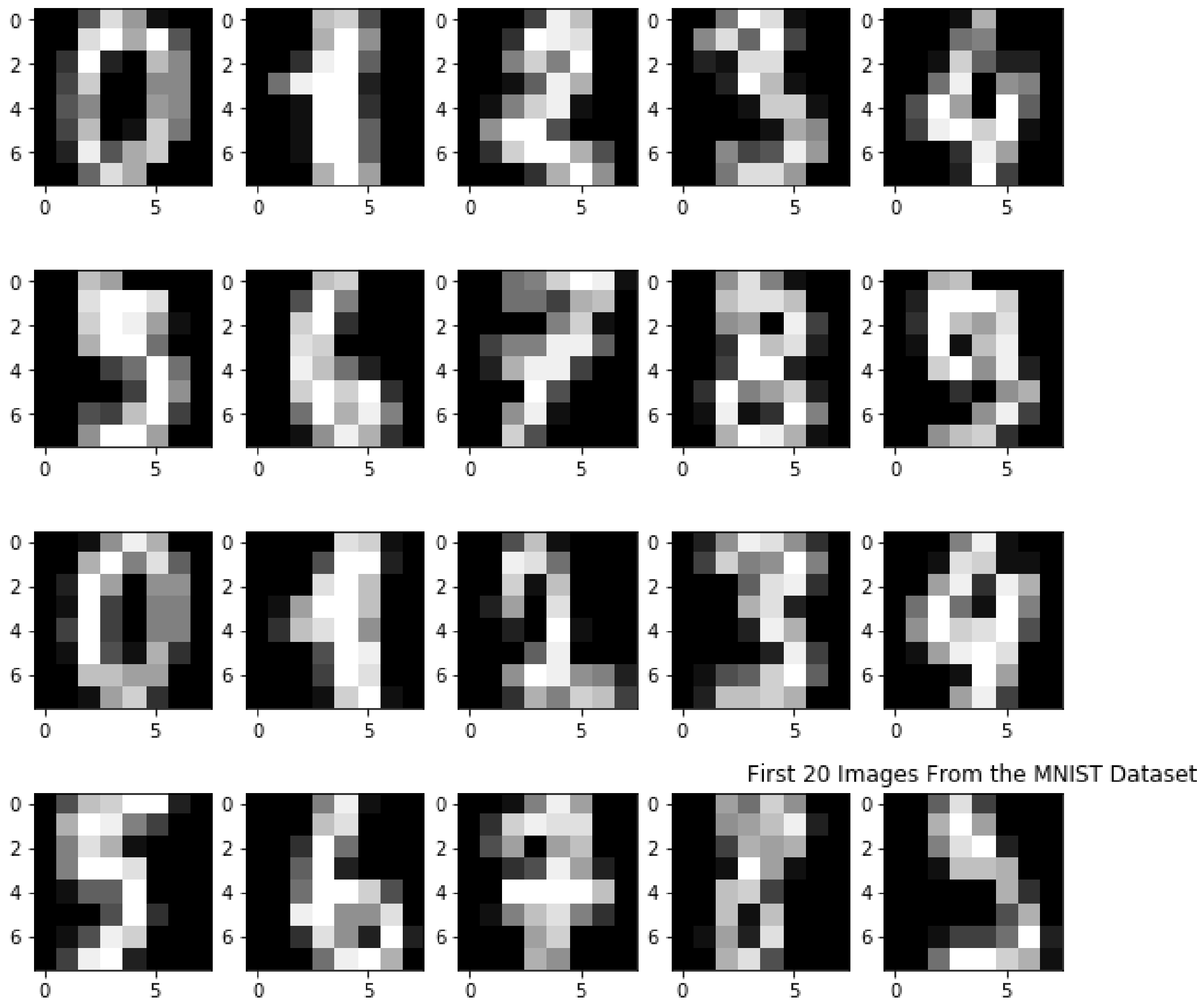
```
(1797, 64) (1797,)
```

Preview the dataset

Now that the dataset is loaded, display the first 20 images.

```
import matplotlib.pyplot as plt
%matplotlib inline

fig, axes = plt.subplots(nrows=4, ncols=5, figsize=(10,10))
for n in range(20):
    i = n //5
    j = n%5
    ax = axes[i][j]
    ax.imshow(data.images[n], cmap=plt.cm.gray)
plt.title('First 20 Images From the MNIST Dataset');
```



Baseline model

Now it's time to fit an initial baseline model.

- Split the data into training and test sets. Set `random_state=22`
- Fit a support vector machine to the dataset. Set `gamma='auto'`
- Record the training time
- Print the training and test accucary of the model

```
from sklearn import svm
from sklearn.model_selection import train_test_split
X = data.data
y = data.target
X_train, X_test, y_train, y_test = train_test_split(X,y, random_state=22)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(1347, 64) (450, 64) (1347,) (450,)
```

```python
clf = svm.SVC(gamma='auto')
%timeit clf.fit(X_train, y_train)
```

```
252 ms ± 14.8 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```python
# Training and test accuracy
train_acc = clf.score(X_train, y_train)
test_acc = clf.score(X_test, y_test)
print('Training Accuracy: {}\tTesting Accuracy: {}'.format(train_acc, test_acc))
```

```
Training Accuracy: 1.0  Testing Accuracy: 0.58
```

Grid search baseline

Refine the initial model by performing a grid search to tune the hyperparameters. The two most important parameters to adjust are `'C'` and `'gamma'`. Once again, be sure to record the training time as well as the training and test accuracy.

```python
import numpy as np
from sklearn.model_selection import GridSearchCV

clf = svm.SVC()

param_grid = {'C' : np.linspace(.1, 10, num=11),
             'gamma' : np.linspace(10**-3, 5, num=11)}

grid_search = GridSearchCV(clf, param_grid, cv=5)
%timeit grid_search.fit(X_train, y_train)
```

```
1min 22s ± 127 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```python
grid_search.best_params_
```

```
{'C': 2.08, 'gamma': 0.001}
```

```
train_acc = grid_search.best_estimator_.score(X_train, y_train)
test_acc = grid_search.best_estimator_.score(X_test, y_test)
print('Training Accuracy: {}\tTesting Accuracy: {}'.format(train_acc, test_acc))
```

```
Training Accuracy: 1.0	Testing Accuracy: 0.9911111111111112
```

Compressing with PCA

Now that you've fit a baseline classifier, it's time to explore the impacts of using PCA as a preprocessing technique. To start, perform PCA on `X_train`. (Be sure to only fit PCA to `X_train`; you don't want to leak any information from the test set.) Also, don't reduce the number of features quite yet. You'll determine the number of features needed to account for 95% of the overall variance momentarily.

```
from sklearn.decomposition import PCA
import seaborn as sns
sns.set_style('darkgrid')
```

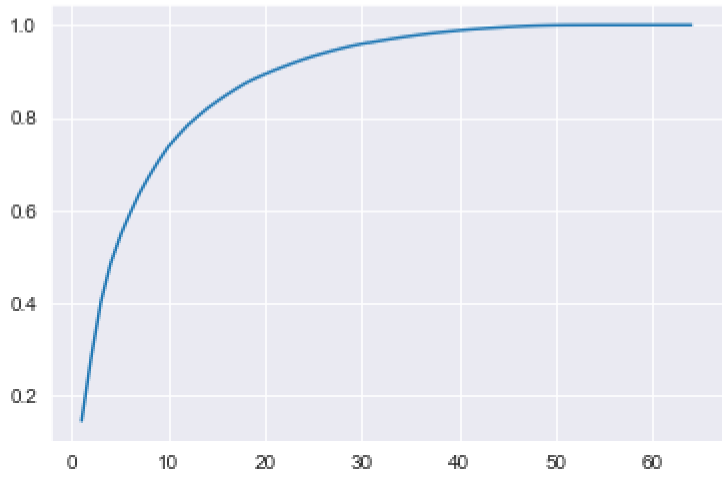
```
pca = PCA()
X_pca = pca.fit_transform(X_train)
```

Plot the explained variance versus the number of features

In order to determine the number of features you wish to reduce the dataset to, it is sensible to plot the overall variance accounted for by the first n principal components. Create a graph of the variance explained versus the number of principal components.

```
plt.plot(range(1,65), pca.explained_variance_ratio_.cumsum())
```

```
[<matplotlib.lines.Line2D at 0x1a19ddb7f0>]
```



Determine the number of features to capture 95% of the variance

Great! Now determine the number of features needed to capture 95% of the dataset's overall variance.

```
total_explained_variance = pca.explained_variance_ratio_.cumsum()
n_over_95 = len(total_explained_variance[total_explained_variance >= .95])
n_to_reach_95 = X.shape[1] - n_over_95 + 1
print("Number features: {}\tTotal Variance Explained: {}".format(n_to_reach_95, tota
```

```
Number features: 29     Total Variance Explained: 0.9549611953216074
```

Subset the dataset to these principal components which capture 95% of the overall variance

Use your knowledge to reproject the dataset into a lower-dimensional space using PCA.

```
pca = PCA(n_components=n_to_reach_95)
X_pca_train = pca.fit_transform(X_train)
pca.explained_variance_ratio_.cumsum()[-1]
```

```
0.954954231338425
```

Refit a model on the compressed dataset

Now, refit a classification model to the compressed dataset. Be sure to time the required training time, as well as the test and training accuracy.

```
X_pca_test = pca.transform(X_test)
clf = svm.SVC(gamma='auto')
%timeit clf.fit(X_pca_train, y_train)
```

```
137 ms ± 958 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

```
train_pca_acc = clf.score(X_pca_train, y_train)
test_pca_acc = clf.score(X_pca_test, y_test)
print('Training Accuracy: {}\tTesting Accuracy: {}'.format(train_pca_acc, test_pca_a
```

```
Training Accuracy: 1.0  Testing Accuracy: 0.14888888888888888
```

Grid search

Finally, use grid search to find optimal hyperparameters for the classifier on the reduced dataset. Be sure to record the time required to fit the model, the optimal hyperparameters and the test and train accuracy of the resulting model.

```
clf = svm.SVC()

param_grid = {'C' : np.linspace(.1, 10, num=11),
             'gamma' : np.linspace(10**-3, 5, num=11)}

grid_search = GridSearchCV(clf, param_grid, cv=5)
%timeit grid_search.fit(X_pca_train, y_train)
```

```
51.8 s ± 917 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
grid_search.best_params_
```

```
{'C': 2.08, 'gamma': 0.001}
```

```python
train_acc = grid_search.best_estimator_.score(X_pca_train, y_train)
test_acc = grid_search.best_estimator_.score(X_pca_test, y_test)
print('Training Accuracy: {}\tTesting Accuracy: {}'.format(train_acc, test_acc))
```

```
Training Accuracy: 0.9992576095025983   Testing Accuracy: 0.9933333333333333
```

Summary

Well done! In this lab, you employed PCA to reduce a high dimensional dataset. With this, you observed the potential cost benefits required to train a model and performance gains of the model itself.

Releases

No releases published

Packages

No packages published

Contributors 4

mathymitchell

sumedh10 Sumedh Panchadhar

lmcm18

h-parker Hannah

Languages

- **Jupyter Notebook** 100.0%