PCA

August 19, 2022

[1]:
```python
from sklearn.datasets import load_digits
```

[2]:
```python
import pandas as pd
import numpy as np
```

[3]:
```python
data = load_digits()
```

[4]:
```python
print(data.DESCR)
```

```
.. _digits_dataset:

Optical recognition of handwritten digits dataset
--------------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 1797
    :Number of Attributes: 64
    :Attribute Information: 8x8 image of integer pixels in the range 0..16.
    :Missing Attribute Values: None
    :Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)
    :Date: July; 1998

This is a copy of the test set of the UCI ML hand-written digits datasets
https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digit
s

The data set contains images of hand-written digits: 10 classes where
each class refers to a digit.

Preprocessing programs made available by NIST were used to extract
normalized bitmaps of handwritten digits from a preprinted form. From a
total of 43 people, 30 contributed to the training set and different 13
to the test set. 32x32 bitmaps are divided into nonoverlapping blocks of
4x4 and the number of on pixels are counted in each block. This generates
an input matrix of 8x8 where each element is an integer in the range
0..16. This reduces dimensionality and gives invariance to small
distortions.
```

```
For info on NIST preprocessing routines, see M. D. Garris, J. L. Blue, G.
T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C.
L. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5469,
1994.

.. topic:: References

  - C. Kaynak (1995) Methods of Combining Multiple Classifiers and Their
    Applications to Handwritten Digit Recognition, MSc Thesis, Institute of
    Graduate Studies in Science and Engineering, Bogazici University.
  - E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.
  - Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Qin.
    Linear dimensionalityreduction using relevance weighted LDA. School of
    Electrical and Electronic Engineering Nanyang Technological University.
    2005.
  - Claudio Gentile. A New Approximate Maximal Margin Classification
    Algorithm. NIPS. 2000.
```

[5]:

```python
features = data.data
target = data.target

features.shape,target.shape
```

[5]:

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

[6]:

```python
from sklearn.model_selection import train_test_split
```

[7]:

```python
X_train, X_test, y_train, y_test = train_test_split(features, target,␣
 ↪train_size = 0.6, random_state = 3)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(1078, 64)
(719, 64)
(1078,)
(719,)
```

[8]:

```python
from sklearn.linear_model import LogisticRegression
```

[9]:

```python
my_model = LogisticRegression()
```

[10]:

```python
my_model.fit(X_train,y_train)
```

```
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
```

```
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```
[10]: LogisticRegression()
```

```
[11]: preds = my_model.predict(X_test)
```

```
[12]: from sklearn.metrics import accuracy_score,confusion_matrix
```

```
[13]: print(accuracy_score(y_test, preds))
```

```
0.9582753824756607
```

```
[14]: print(confusion_matrix(y_test, preds))
```

```
[[74  0  0  0  1  0  0  0  0  0]
 [ 0 72  0  0  0  0  0  0  2  1]
 [ 0  1 77  0  0  0  0  0  0  0]
 [ 0  0  2 66  0  0  0  0  1  0]
 [ 0  4  0  0 79  0  1  1  1  0]
 [ 0  1  0  0  1 60  0  0  0  2]
 [ 0  1  0  0  0  1 55  0  1  0]
 [ 0  0  0  1  0  0  0 73  2  0]
 [ 0  1  0  0  0  1  0  0 66  0]
 [ 0  0  0  0  1  1  0  0  1 67]]
```

```
[15]: from sklearn.decomposition import PCA
```

Initially I have 64 dimensions. Now I want to bring it down to 4 dimensions

```
[61]: pca = PCA(n_components=.99)
```

```
[62]: pca.fit(X_train)
```

```
[62]: PCA(n_components=0.99)
```

```
[63]: pca.explained_variance_ratio_
```

```
[63]: array([0.14924092, 0.13473079, 0.1156578 , 0.08302537, 0.06011606,
              0.05032773, 0.04284684, 0.03556855, 0.03294894, 0.03039433,
              0.02454552, 0.02311806, 0.01912235, 0.01765833, 0.01483066,
              0.01426726, 0.01361367, 0.01250578, 0.01025873, 0.00956485,
              0.00899525, 0.00826308, 0.00757479, 0.00733326, 0.00666861,
              0.00603645, 0.00573677, 0.00514727, 0.00493476, 0.00431694,
              0.00372314, 0.00357927, 0.00333257, 0.00320508, 0.00296291,
              0.00287043, 0.0026174 , 0.0023106 , 0.00222457, 0.00213944,
              0.00178448])
```

```
[64]: X_train_transformed = pca.transform(X_train)
      X_test_transformed = pca.transform(X_test)
```

```
[65]: X_train_transformed.shape, X_test_transformed.shape
```

```
[65]: ((1078, 41), (719, 41))
```

```
[66]: X_train_transformed
```

```
[66]: array([[  8.31885532, -11.63536164,   1.93999083, …,   0.70688983,
               -0.28709199,  -0.10932069],
             [  3.69591051,  22.68576943,  -5.66316747, …,   1.32126734,
               -2.25843322,   0.22766461],
             [  8.7400552 ,  -2.00460958,  12.21941747, …,   0.42945575,
               -0.28278685,  -1.00717996],
             …,
             [ -0.20762926,  20.34392093,  -2.0983895 , …,  -1.18724655,
               -0.36066818,  -0.82807739],
             [ 19.78689345,  -6.56299576,  18.79910461, …,  -1.92655226,
               -3.21873055,  -2.71094689],
             [-12.92312023, -11.63513571,  14.44423877, …,  -0.49248432,
                0.07430153,   0.55638722]])
```

```
[67]: pca.inverse_transform(X_train_transformed).shape
```

```
[67]: (1078, 64)
```

```
[68]: my_model.fit(X_train_transformed,y_train)
```

```
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

[68]:

```
LogisticRegression()
```

[69]:

```
preds = my_model.predict(X_test_transformed)
```

[70]:

```
print(accuracy_score(y_test, preds))
```

```
0.9485396383866481
```

[ ]:

```

```