index

April 20, 2022

1 Image Recognition with PCA - Lab

1.1 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.

1.2 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

1.3 Load the data

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

[1]:
```
# Load the dataset
from sklearn.datasets import load_digits
data = load_digits()
print(data.data.shape, data.target.shape)
```

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

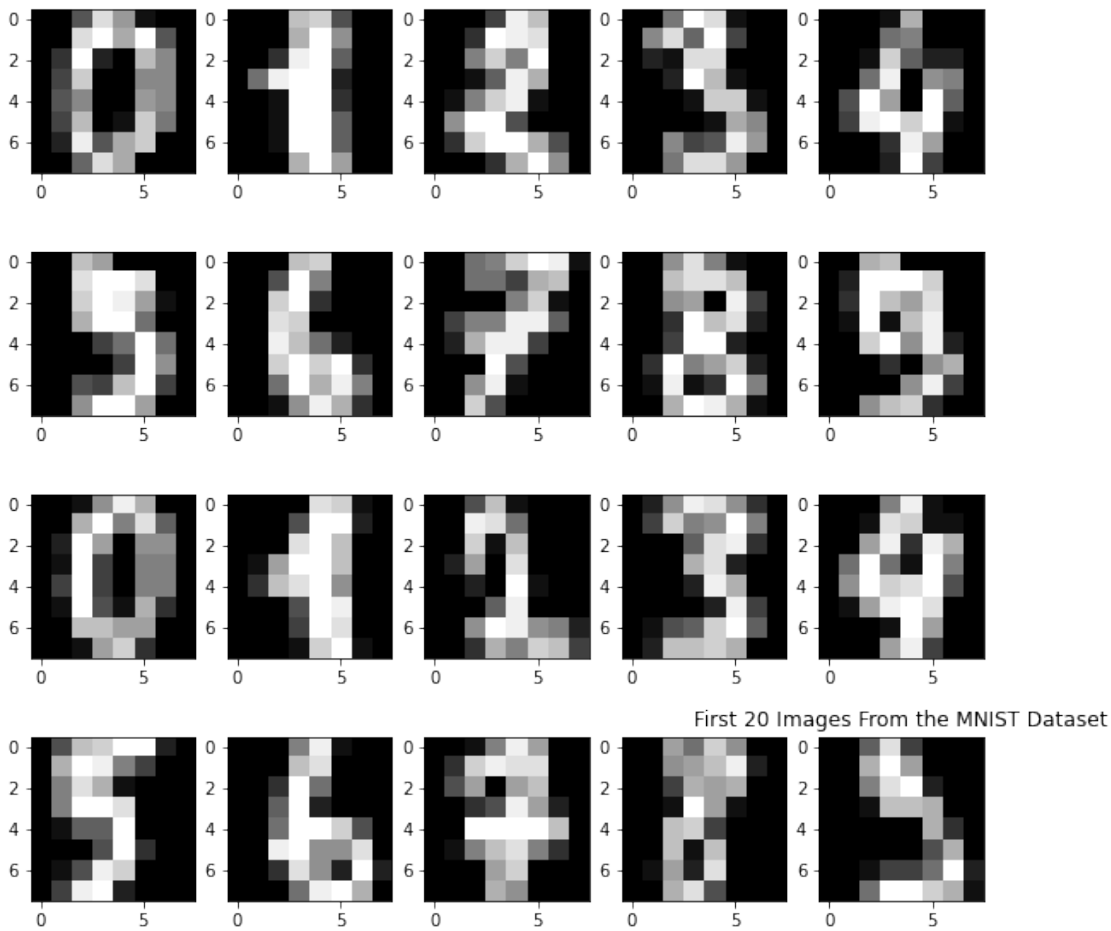
1.4 Preview the dataset

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

[11]:
```
# Display the first 20 images
import matplotlib.pyplot as plt

fig, axes = plt.subplots(nrows=4, ncols=5, figsize=(10,10))

for n in range(0,20):
    ax = axes[n//5][n%5]
    ax.imshow(data.images[n], cmap = "gray")
```

```
plt.title("First 20 Images From the MNIST Dataset")
```

[11]:
```
Text(0.5, 1.0, 'First 20 Images From the MNIST Dataset')
```



1.5 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

[12]:
```
# Split the data
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,)
```

[14]:
```
# Fit a naive model
from sklearn import svm
clf = svm.SVC(gamma='auto')
%timeit  clf.fit(X_train, y_train)
```

```
259 ms ± 2.55 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

[15]:
```
# Training and test accuracy
train_acc = clf.score(X_train, y_train)
test_acc = clf.score(X_test, y_test)
print('Training Accuracy: {}\nTesting Accuracy: {}'.format(train_acc, test_acc))
```

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

1.5.1 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.

[16]:
```
# Your code here
#   Your code may take several minutes to run
from sklearn.model_selection import GridSearchCV

clf_II = svm.SVC()

params = {"C":[0.1, 2, 5, 10],
          "gamma" : [10**-3, 10**-1, 5]}

grid_clf = GridSearchCV(estimator = clf_II,param_grid = params, cv = 5)

%timeit grid_clf.fit(X_train, y_train)
```

```
11 s ± 41.2 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

[17]:
```
# Print the best parameters
grid_clf.best_params_
```

[17]:
```
{'C': 2, 'gamma': 0.001}
```

[19]:
```
# Print the training and test accuracy
train_acc = grid_clf.best_estimator_.score(X_train, y_train)
test_acc = grid_clf.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
```

[21]:
```
import pandas as pd

pd.DataFrame(grid_clf.cv_results_)
```

[21]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C
0	0.085570	0.000236	0.057578	0.003867	0.1
1	0.171951	0.000297	0.073765	0.002019	0.1
2	0.106200	0.000685	0.063726	0.005890	0.1
3	0.048678	0.000365	0.035844	0.002544	2
4	0.203174	0.001251	0.062820	0.004415	2
5	0.125507	0.000598	0.063172	0.006663	2
6	0.048609	0.000604	0.032762	0.000557	5
7	0.202286	0.000645	0.070390	0.004514	5
8	0.125291	0.000631	0.067550	0.002296	5
9	0.048551	0.000400	0.038358	0.000618	10
10	0.202233	0.001086	0.063226	0.002653	10
11	0.125704	0.000866	0.067391	0.001419	10

	param_gamma	params	split0_test_score
0	0.001	{'C': 0.1, 'gamma': 0.001}	0.940741
1	0.1	{'C': 0.1, 'gamma': 0.1}	0.107407
2	5	{'C': 0.1, 'gamma': 5}	0.107407
3	0.001	{'C': 2, 'gamma': 0.001}	0.988889
4	0.1	{'C': 2, 'gamma': 0.1}	0.107407
5	5	{'C': 2, 'gamma': 5}	0.107407
6	0.001	{'C': 5, 'gamma': 0.001}	0.988889
7	0.1	{'C': 5, 'gamma': 0.1}	0.107407
8	5	{'C': 5, 'gamma': 5}	0.107407
9	0.001	{'C': 10, 'gamma': 0.001}	0.988889
10	0.1	{'C': 10, 'gamma': 0.1}	0.107407
11	5	{'C': 10, 'gamma': 5}	0.107407

	split1_test_score	split2_test_score	split3_test_score
0	0.962963	0.973978	0.955390
1	0.103704	0.104089	0.107807
2	0.103704	0.104089	0.107807
3	0.992593	0.992565	0.996283
4	0.103704	0.104089	0.107807
5	0.103704	0.104089	0.107807
6	0.992593	0.992565	0.996283

```
7            0.103704           0.104089           0.107807
8            0.103704           0.104089           0.107807
9            0.992593           0.992565           0.996283
10           0.103704           0.104089           0.107807
11           0.103704           0.104089           0.107807

    split4_test_score  mean_test_score  std_test_score  rank_test_score
0            0.955390         0.957692        0.010871                4
1            0.107807         0.106163        0.001860                5
2            0.107807         0.106163        0.001860                5
3            0.985130         0.991092        0.003788                1
4            0.107807         0.106163        0.001860                5
5            0.107807         0.106163        0.001860                5
6            0.985130         0.991092        0.003788                1
7            0.107807         0.106163        0.001860                5
8            0.107807         0.106163        0.001860                5
9            0.985130         0.991092        0.003788                1
10           0.107807         0.106163        0.001860                5
11           0.107807         0.106163        0.001860                5
```

1.6 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.

[31]:
```
# Your code here
from sklearn.decomposition import PCA
pca = PCA(n_components = 30)
pca.fit_transform(X_train)

sum(pca.explained_variance_ratio_)
```

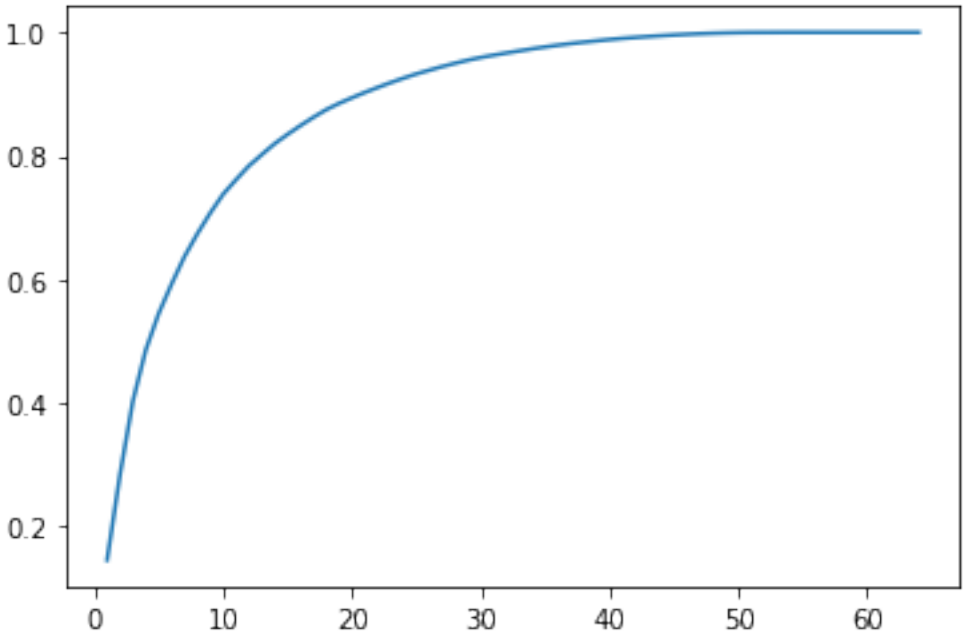
[31]:
```
0.9593119979453406
```

1.7 Plot the explained variance versus the number of features

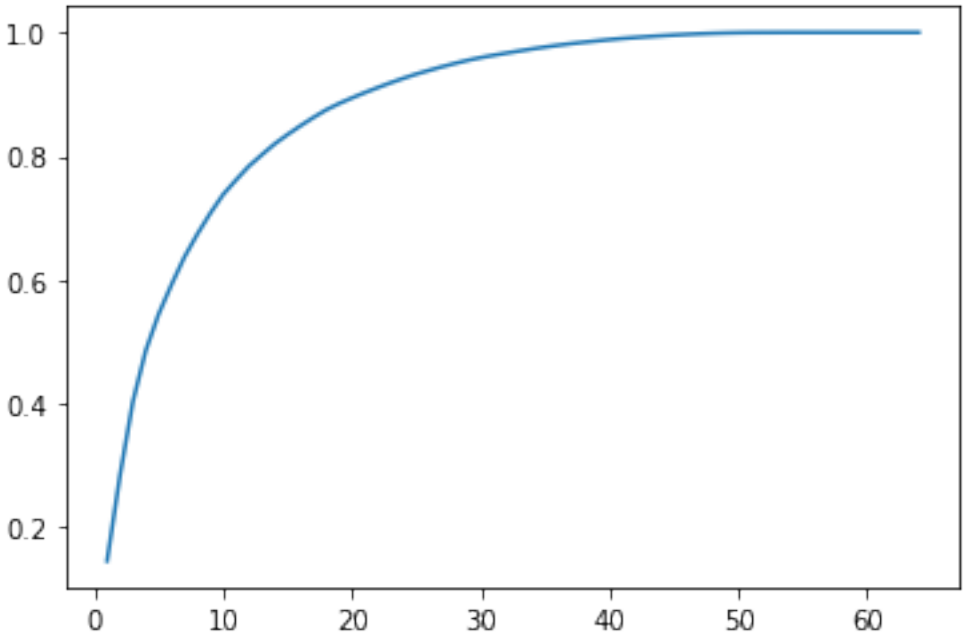
In order to determine the number of features you wish to reduce the dataset to, it is sensible to plot the overall variance accounted for by the first n principal components. Create a graph of the variance explained versus the number of principal components.

[40]:
```
# Your code here
l = []
for i in range(1, 65):
    pca = PCA(n_components = i)
```

```
    pca.fit_transform(X_train)
    l.append(sum(pca.explained_variance_ratio_))

plt.plot(range(1, 65), l);
```



[41]:
```
### G

pca = PCA()
pca.fit_transform(X_train)

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



1.8 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.

[44]:
```
# Your code here
for i in range(1, 65):
    pca = PCA(n_components = i)
    pca.fit_transform(X_train)
    if sum(pca.explained_variance_ratio_) >= 0.95:
        print(i)
        print(sum(pca.explained_variance_ratio_))
        break
```

```
29
0.9549599978745522
```

[47]:
```
## G
pca = PCA()
pca.fit_transform(X_train)

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,
 total_explained_variance[n_to_reach_95-1]))
```

```
Number features: 29     Total Variance Explained: 0.9549611953216072
```

1.9 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.

[55]:
```
# Your code here
pca_95 = PCA(n_components = 29)
X_train_pca = pca_95.fit_transform(X_train)

pca_95.explained_variance_ratio_.cumsum()[-1]
```

[55]:
```
0.9549478401811489
```

1.10 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.

[56]:
```
# Your code here
X_test_pca = pca_95.transform(X_test)
clf = svm.SVC(gamma='auto')
%timeit clf.fit(X_train_pca, y_train)

train_acc = clf.score(X_train_pca, y_train)
test_acc = clf.score(X_test_pca, y_test)
print('Training Accuracy: {}\nTesting Accuracy: {}'.format(train_acc, test_acc))
```

```
359 ms ± 4.07 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
Training Accuracy: 1.0
Testing Accuracy: 0.14888888888888888
```

1.10.1 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.

[ ]:
```
# Your code here
#   Your code may take several minutes to run

from sklearn.model_selection import GridSearchCV

clf_pca = svm.SVC()
```

```
params = {"C":[0.1, 2, 5, 10],
          "gamma" : [10**-3, 10**-1, 5]}

grid_pca = GridSearchCV(estimator = clf_pca,param_grid = params, cv = 5)

%timeit grid_pca.fit(X_train_pca, y_train)
```

[ ]:
```
# Print the best parameters
grid_pca.best_params_
```

[ ]:
```
# Print the training and test accuracy
train_acc = grid_pca.best_estimator_.score(X_train_pca, y_train)
test_acc = grid_pca.best_estimator_.score(X_test_pca, y_test)
print('Training Accuracy: {}\tTesting Accuracy: {}'.format(train_acc, test_acc))
```

[ ]:
```
pd.DataFrame(grid_pca.cv_results_)
```

1.11 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.