Machine Learning
MSE FTP MachLe
[Christoph Würsch (mailto:christoph.wuersch@ost.ch)](mailto:christoph.wuersch@ost.ch)

OST
Ostschweizer
Fachhochschule

Lab 11, A6: Feature Engineering using PCA

First, let's make sure this notebook works well in both python 2 and 3, import a few common modules, ensure MatplotLib plots figures inline and prepare a function to save the figures:

Setup

In [1]:
```
# To support both python 2 and python 3
from __future__ import division, print_function, unicode_literals

# Common imports
import numpy as np
import os

# to make this notebook's output stable across runs
np.random.seed(42)

# To plot pretty figures
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
plt.rcParams['axes.labelsize'] = 14
plt.rcParams['xtick.labelsize'] = 12
plt.rcParams['ytick.labelsize'] = 12

# Where to save the figures
PROJECT_ROOT_DIR = "."
CHAPTER_ID = "dim_reduction"

def save_fig(fig_id, tight_layout=True):
    path = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID, fig_id + ".png")
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format='png', dpi=300)
```

(a) Loading the MINST Dataset

Exercise: Load the MNIST dataset (introduced in chapter 3) and split it into a training set and a test set (take the first 60,000 instances for training, and the remaining 10,000 for testing).

In [2]:
```
from sklearn.decomposition import PCA
```

In [3]:
```
#from sklearn.datasets import fetch_mldata

from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784')
```

In [19]:
```
X_train = mnist['data'][:60000].values
y_train = mnist['target'][:60000].astype(int).values

X_test = mnist['data'][60000:].values
y_test = mnist['target'][60000:].astype(int).values
```

In [20]:
```
X_train
y_train
```

Out[20]:
```
array([5, 0, 4, ..., 5, 6, 8], dtype=int64)
```

In [10]:
```
#we save the data to disk so that we can fetch it fast if we need it using the next cell:

import pandas as pd
#mnist.data.to_pickle('mnist_data_784.pkl')
#mnist.target.to_pickle('mnist_label_784.pkl');

#read data from pickle file if there is no internet connection
X=pd.read_pickle('mnist_data_784.pkl').values
y=pd.read_pickle('mnist_label_784.pkl').values

X_train = X[:60000:,:]
y_train = y[:60000].astype(int)

X_test = X[60000:,:]
y_test = y[60000:].astype(int)
```

(b) Training a Random Forest classifier on the dataset

Exercise: Train a Random Forest classifier on the dataset and time how long it takes, then evaluate the resulting model on the test set.

In [11]:
```
from sklearn.ensemble import RandomForestClassifier

rnd_clf = RandomForestClassifier(random_state=42)
```

In [12]:
```
import time

t0 = time.time()
rnd_clf.fit(X_train, y_train)
t1 = time.time()
```

In [13]:
```
print("Training took {:.2f}s".format(t1 - t0))
```

```
Training took 45.57s
```

In [14]:
```
from sklearn.metrics import accuracy_score

y_pred = rnd_clf.predict(X_test)
accuracy_score(y_test, y_pred)
```

Out[14]:
```
0.9705
```

(c) Use PCA to reduce the dataset's dimensionality, with an explained variance ratio of 95%

Exercise: Next, use PCA to reduce the dataset's dimensionality, with an explained variance ratio of 95%.

In [21]:
```
from sklearn.decomposition import PCA

pca = PCA(n_components=0.95)
X_train_reduced = pca.fit_transform(X_train)
```

Exercise: Train a new Random Forest classifier on the reduced dataset and see how long it takes. Was training much faster?

In [22]:
```
rnd_clf2 = RandomForestClassifier(random_state=42)
t0 = time.time()
rnd_clf2.fit(X_train_reduced, y_train)
t1 = time.time()
```

In [23]:
```
print("Training took {:.2f}s".format(t1 - t0))
```

```
Training took 92.38s
```

Oh no! Training is actually more than twice slower now! How can that be? Well, as we saw in this chapter, dimensionality reduction does not always lead to faster training time: it depends on the dataset, the model and the training algorithm. See figure 8-6 (the `manifold_decision_boundary_plot*` plots above). If you try a softmax classifier instead of a random forest classifier, you will find that training time is reduced by a factor of 3 when using PCA. Actually, we will do this in a second, but first let's check the precision of the new random forest classifier.

(d) Evaluate the classifier on the test set: how does it compare to the previous classifier?

In [24]:
```
X_test_reduced = pca.transform(X_test)

y_pred = rnd_clf2.predict(X_test_reduced)
accuracy_score(y_test, y_pred)
```

Out[24]:
```
0.9481
```

It is common for performance to drop slightly when reducing dimensionality, because we do lose some useful signal in the process. However, the performance drop is rather severe in this case. So PCA really did not help: it slowed down training and reduced performance. :(

Let's see if it helps when using softmax regression:

In [25]:
```
from sklearn.linear_model import LogisticRegression

log_clf = LogisticRegression(multi_class="multinomial", solver="lbfgs", random_state=42,max_it
t0 = time.time()
log_clf.fit(X_train, y_train)
t1 = time.time()
```

```
C:\Users\wurc\.conda\envs\ML\lib\site-packages\sklearn\linear_model\_logistic.py:765: Converg
enceWarning: 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 (https://scikit-learn.org/stab
le/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://sc
ikit-learn.org/stable/modules/linear_model.html#logistic-regression)
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

In [26]:
```
print("Training took {:.2f}s".format(t1 - t0))
```

```
Training took 165.93s
```

In [27]:
```
y_pred = log_clf.predict(X_test)
accuracy_score(y_test, y_pred)
```

Out[27]:
```
0.9202
```

Okay, so softmax regression takes much longer to train on this dataset than the random forest classifier, plus it performs worse on the test set. But that's not what we are interested in right now, we want to see how much PCA can help softmax regression. Let's train the softmax regression model using the reduced dataset:

In [28]:
```
log_clf2 = LogisticRegression(multi_class="multinomial", solver="lbfgs", random_state=42,max_i
t0 = time.time()
log_clf2.fit(X_train_reduced, y_train)
t1 = time.time()
```

```
C:\Users\wurc\.conda\envs\ML\lib\site-packages\sklearn\linear_model\_logistic.py:765: Converg
enceWarning: 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 (https://scikit-learn.org/stab
le/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://sc
ikit-learn.org/stable/modules/linear_model.html#logistic-regression)
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

In [29]:
```
print("Training took {:.2f}s".format(t1 - t0))
```

```
Training took 86.67s
```

Nice! Reducing dimensionality led to a 4× speedup. :) Let's the model's accuracy:

In [30]:
```
y_pred = log_clf2.predict(X_test_reduced)
accuracy_score(y_test, y_pred)
```

Out[30]:
```
0.9236
```

A very slight drop in performance, which might be a reasonable price to pay for a 4× speedup, depending on the application.

So there you have it: PCA can give you a formidable speedup... but not always!