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

OST Ostschweizer Fachhochschule

Lab 10, A7 Principal Component Analysis for Noise Filtering

PCA as Noise Filtering

PCA can also be used as a filtering approach for noisy data. The idea is this: any components with variance much larger than the effect of the noise should be relatively unaffected by the noise. So if you reconstruct the data using just the largest subset of principal components, you should be preferentially keeping the signal and throwing out the noise.

Let's see how this looks with the digits data. First we will plot several of the input noise-free data:

In [2]:
```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA

#digits = load_digits()
#digits.data.shape

from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784',parser='auto')

#digits.data = mnist['data'][:2000]
#digits.target = mnist['target'][:2000].astype(int)
```

In [3]:
```
mnist.data
```

Out[3]:

	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	...	pixel775	pixel776	pixel777	p
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
69995	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
69996	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
69997	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
69998	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
69999	0	0	0	0	0	0	0	0	0	0	...	0	0	0	

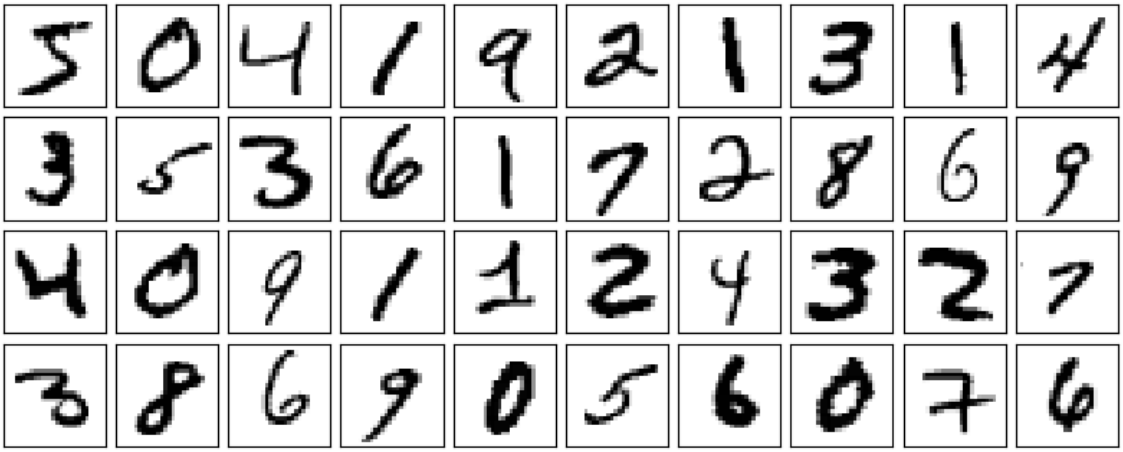
70000 rows × 784 columns

In [4]:
```
digits=mnist
```

In [5]:
```
digits.data = mnist['data'][:2000]
digits.target = mnist['target'][:2000].astype(int)
```

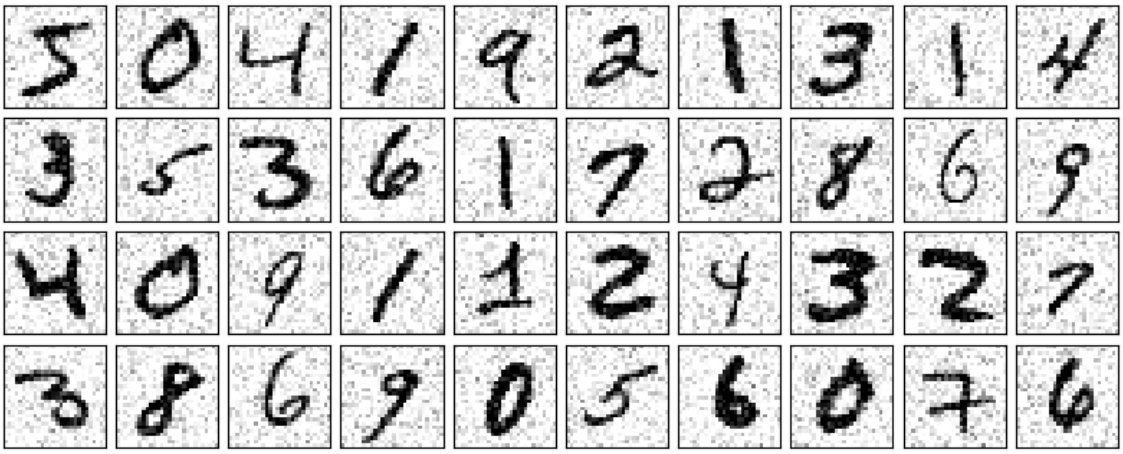
In [6]:
```
data=digits.data.iloc[:,:2000].values
target=digits.target.iloc[:2000].values
```

In [7]:
```
def plot_digits(data):
    fig, axes = plt.subplots(4, 10, figsize=(10, 4),
                             subplot_kw={'xticks':[], 'yticks':[]},
                             gridspec_kw=dict(hspace=0.1, wspace=0.1))
    for i, ax in enumerate(axes.flat):
        ax.imshow(data[i].reshape(28, 28),
                  cmap='binary', interpolation='nearest',
                  clim=(0, 256))
plot_digits(data)
```



Now lets add some random noise to create a noisy dataset, and re-plot it:

In [8]:
```
np.random.seed(42)
noisy = np.random.normal(data, 40)
plot_digits(noisy)
```



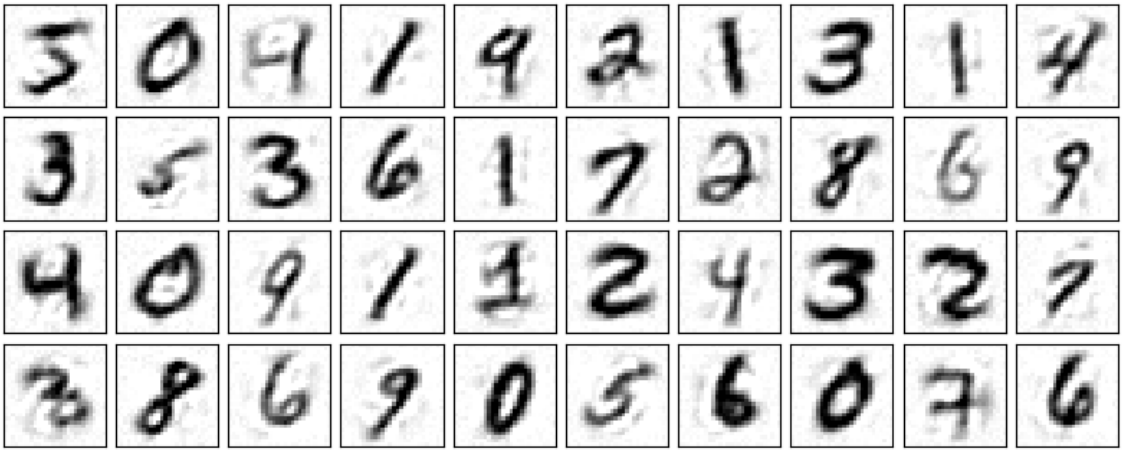
It's clear by eye that the images are noisy, and contain spurious pixels. Let's train a PCA on the noisy data, requesting that the projection preserve 50% of the variance:

In [9]:
```
pca = PCA(0.60).fit(noisy)
pca.n_components_
```

Out[9]: 40

Here 50% of the variance amounts to 12 principal components. Now we compute these components, and then use the

In [10]:
```
components = pca.transform(noisy)
filtered = pca.inverse_transform(components)
plot_digits(filtered)
```



This signal preserving/noise filtering property makes PCA a very useful feature selection routine—for example, rather than training a classifier on very high-dimensional data, you might instead train the classifier on the lower-dimensional representation, which will automatically serve to filter out random noise in the inputs.

Principal Component Analysis Summary

In this section we have discussed the use of principal component analysis for dimensionality reduction, for visualization of high-dimensional data, for noise filtering, and for feature selection within high-dimensional data. Because of the versatility and interpretability of PCA, it has been shown to be effective in a wide variety of contexts and disciplines. Given any high-dimensional dataset, I tend to start with PCA in order to visualize the relationship between points (as we did with the digits), to understand the main variance in the data (as we did with the eigenfaces), and to understand the intrinsic dimensionality (by plotting the explained variance ratio). Certainly PCA is not useful for every high-dimensional dataset, but it offers a straightforward and efficient path to gaining insight into high-dimensional data.

PCA's main weakness is that it tends to be highly affected by outliers in the data. For this reason, many robust variants of PCA have been developed, many of which act to iteratively discard data points that are poorly described by the initial components. Scikit-Learn contains a couple interesting variants on PCA, including `RandomizedPCA` and `SparsePCA`, both also in the `sklearn.decomposition` submodule. `RandomizedPCA`, which we saw earlier, uses a non-deterministic method to quickly approximate the first few principal components in very high-dimensional data, while `SparsePCA` introduces a regularization term that serves to enforce sparsity of the components.

In the following sections, we will look at other unsupervised learning methods that build on some of the ideas of PCA.

In [ ]:
```
```