Dimensionality Reduction and Principal Component Analysis

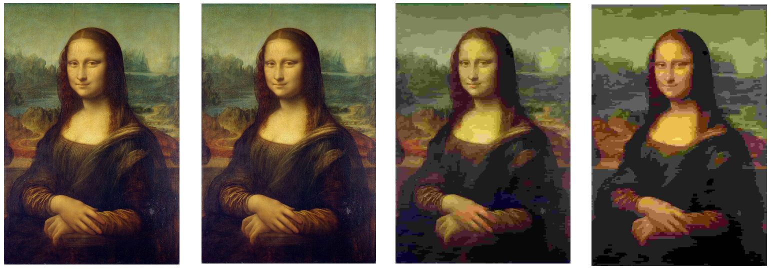
Dimensionality Reduction

Before running any ML algorithm on our data, we may want to reduce the number of features

- To save computer memory/disk space if the data are large.
- To reduce execution time.

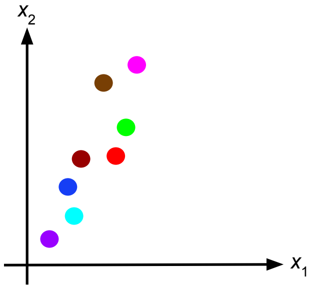
- To visualize our data, e.g., if the dimensions can be reduced to 2 or 3.

Dimensionality Reduction results in an approximation of the original data

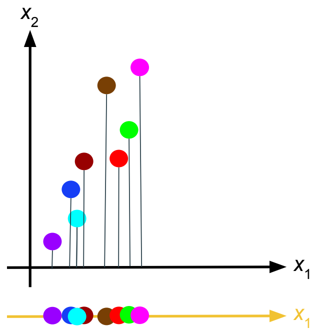


Original 50% 5% 1%

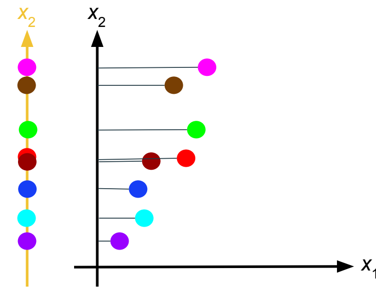
Projecting Data - 2D to 1D



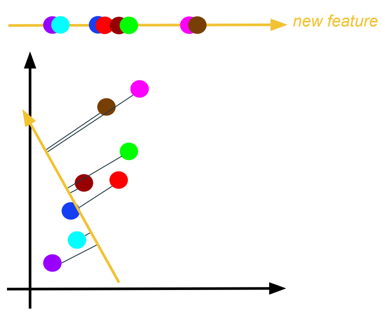
Projecting Data - 2D to 1D



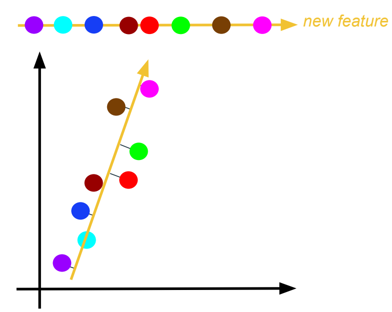
Projecting Data - 2D to 1D



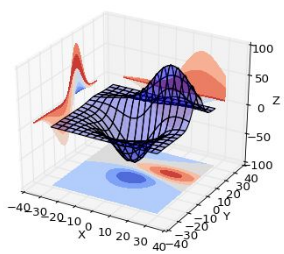
Projecting Data - 2D to 1D

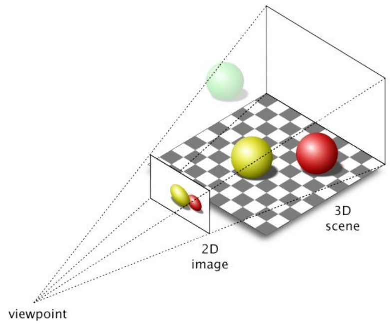


Projecting Data - 2D to 1D

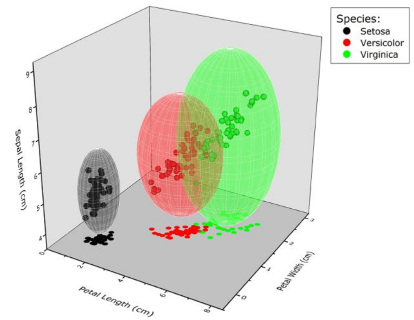


Projecting Data - 3D to 2D

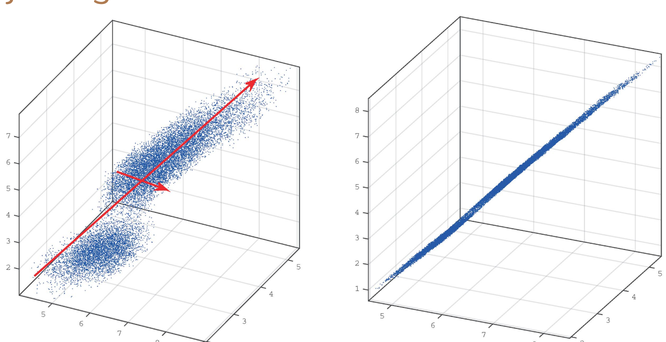




Projecting Data - 3D to 2D



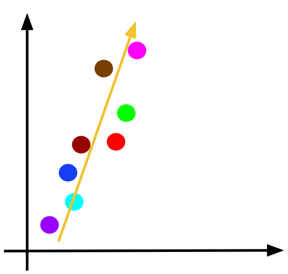
Projecting Data - 3D to 2D



(a) (b)

Why such projections are effective

- In many datasets, some of the features are correlated.
- Correlated features can often be well approximated by a smaller number of new features.
- For example, consider the problem of predicting housing prices. Some of the features may be the square footage of the house, number of bedrooms, number of bathrooms, and lot size. These features are likely correlated.

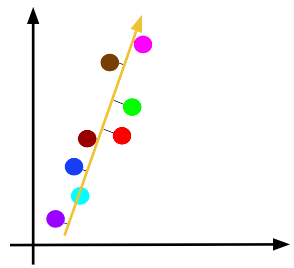


Principal Component Analysis (PCA)

Suppose we want to reduce data from *d* dimensions to *k* dimensions, where *d* > *k*.

PCA finds *k* vectors onto which to project the data so that the projection errors are minimized.

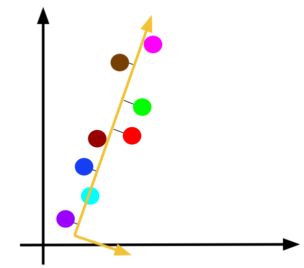
In other words, PCA finds the *principal components*, which offer the best approximation.



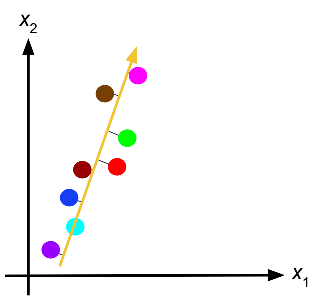
Principal Component Analysis (PCA)

Determine new basis.

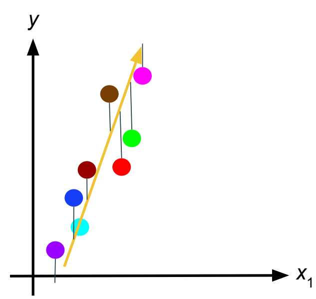
Project data onto *k* axes of new basis with largest variance.



PCA ≠ Linear Regression



PCA



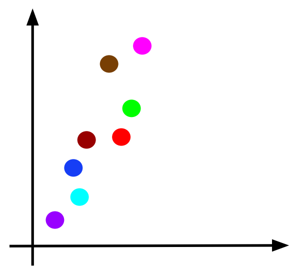
Linear Regression

PCA Algorithm

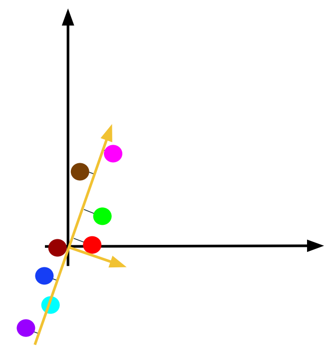
Given *n* data points, each with *d* features, i.e., an *n*×*d* matrix **X**:

- Preprocessing: perform feature scaling
- Compute covariance matrix $\Sigma = \frac{1}{n}\sum_{i=1}^{n}(x^{(i)})(x^{(i)})^T = \frac{1}{n}\mathbf{X}^T*\mathbf{X}$
- Calculate eigenvalues and corresponding eigenvectors of covariance matrix $\mathbf{\Sigma}$ via singular value decomposition
- This yields a new basis of *d* vectors as well as the variance along each of the *d* axes
- Retain the *k* vectors with the largest corresponding variance and project the data onto this *k*-dimensional subspace

PCA Example



PCA Example



PCA with `sklearn`

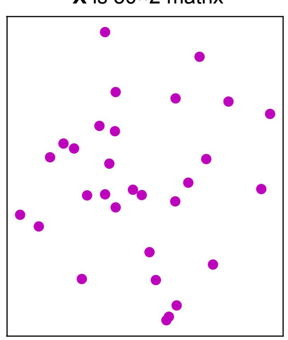
```python
from sklearn.decomposition import PCA
pca = PCA(n_components=1)
Z = pca.fit_transform(X)
print(Z.shape)
print(pca.explained_variance_ratio_)
```

(30, 1)
[0.5815958]

Shape of compressed data Z

58% of variance explained by first principal component

X is 30×2 matrix



PCA with `sklearn`

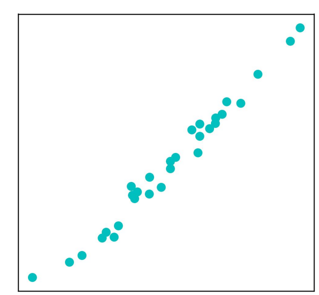
```python
from sklearn.decomposition import PCA
pca = PCA(n_components=1)
Z = pca.fit_transform(X)
print(Z.shape)
print(pca.explained_variance_ratio_)
```

(30, 1)
[0.9945036]

Shape of compressed data Z

99% of variance explained by first principal component

X is 30×2 matrix



PCA with `sklearn`

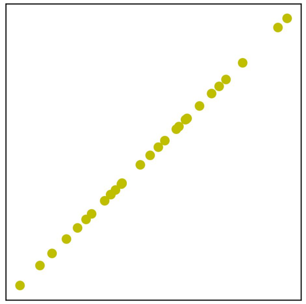
```
from sklearn.decomposition import PCA
pca = PCA(n_components=1)
Z = pca.fit_transform(X)
print(Z.shape)
print(pca.explained_variance_ratio_)
```

(30, 1)
[1.0000000]

Shape of compressed data Z

100% of variance explained by first principal component

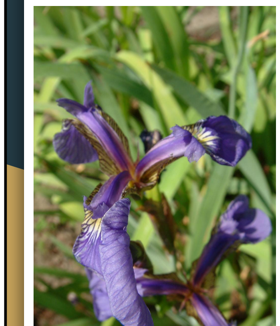
X is 30×2 matrix



Iris Data

For 50 instances of each type of iris, we have four features:

sepal length, sepal width, petal length, petal width



Iris setosa

Iris virginica

Iris versicolor

Feature Scaling

```
from sklearn import preprocessing
X_scaled = preprocessing.scale(X)
```

sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
5.1	3.5	3.4	0.2
4.9	3.0	1.1	1.8
4.7	3.2	2.3	2.1
4.6	2.6	4.5	0.1
5.0	2.8	1.4	0.3
5.4	3.3	5.7	1.4
4.6	2.2	6.1	0.2
5.0	3.4	2.5	2.0
4.4	2.9	1.4	1.7
…	…	…	…
5.9	3.0	5.1	1.8

X is 150×4 matrix

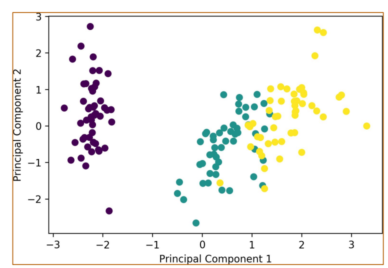
sepal length	sepal width	petal length	petal width
-0.9	0.3	-0.1	0.2
-1.1	-0.7	-0.7	-0.5
0.6	0.0	-0.3	0.0
0.1	-0.1	-1.2	0.1
1.4	-1.0	0.5	-1.1
-0.8	0.0	0.6	-0.4
-0.5	0.4	-0.3	0.0
0.0	0.9	1.3	0.6
-0.4	-0.8	0.9	0.7
…	…	…	…
-0.2	-0.6	0.3	-0.8

X_scaled is 150×4 matrix

PCA: 4D to 2D

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
Z = pca.fit_transform(X_scaled)
print(Z.shape)
print(pca.explained_variance_ratio_)
```

Z is 150×2 matrix



(150, 2)
[0.728 0.230]

Shape of compressed data Z

95% of variance explained by first *two* principal components

Eigenfaces

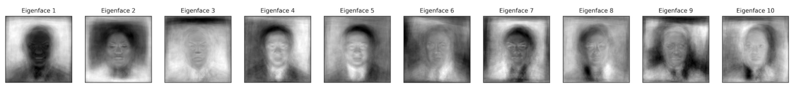
We reduce features of 250x250 pixel images from 62,500 to 200

```
from sklearn.decomposition import PCA
pca = PCA(n_components=200)
eigenfaces = pca.components_
print(pca.explained_variance_ratio_)
print(pca.explained_variance_ratio_.sum())
```

[0.22 0.09 0.05 0.04 0.04 0.03 0.03 0.03 0.02 0.02 ...]
0.952283956204

57% of variance explained by first 10 principal components

95% of variance explained by first 200 principal components



Eigenfaces

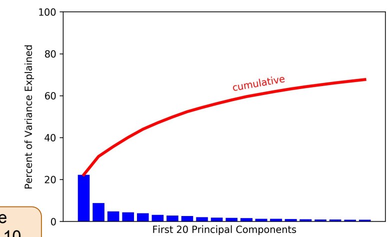
We reduce features of 250x250 pixel images from 62,500 to 200

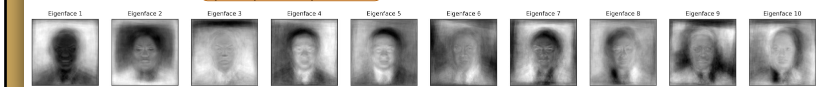
```
from sklearn.decomposition import PCA
pca = PCA(n_components=200)
eigenfaces = pca.components_
print(pca.explained_variance_ratio_)
print(pca.explained_variance_ratio_.sum())
```

[0.22 0.09 0.05 0.04 0.04 0.03 0.03 0.03 0.02 0.02 ...]
0.952283956204

57% of variance explained by first 10 principal components

95% of variance explained by first 200 principal components





Eigenfaces

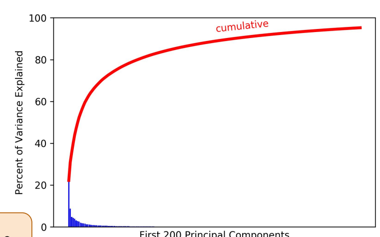
We reduce features of 250x250 pixel images from 62,500 to 200

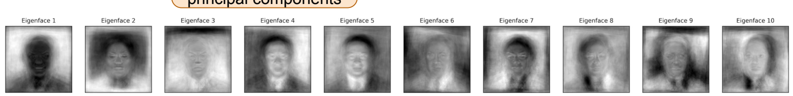
```
from sklearn.decomposition import PCA
pca = PCA(n_components=200)
eigenfaces = pca.components_
print(pca.explained_variance_ratio_)
print(pca.explained_variance_ratio_.sum())
```

[0.22 0.09 0.05 0.04 0.04 0.03 0.03 0.03 0.02 0.02 ...]
0.952283956204

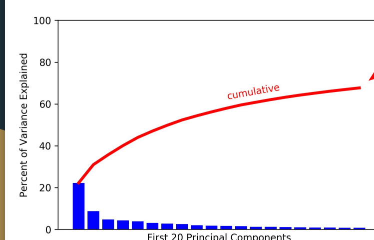
57% of variance explained by first 10 principal components

95% of variance explained by first 200 principal components





How to choose *k*



- We want to lose as little information as possible, i.e., we want the *proportion of variance that is retained* to be as large as possible
- Typically, *k* is chosen so that 95% or 99% of the variance in the data is retained
- If there are many correlated features in the data, often a high percentage of the variance can be retained while using a small number of features, i.e., *k* much less than *d*

PCA Summary

- Prior to running a ML algorithm, PCA can be used to reduce the number of dimensions in the data. This is helpful, e.g., to speed up execution of the ML algorithm.
- Since datasets often have many correlated features, PCA is effective in reducing the number of features while retaining most of the variance in the data.
- Before performing PCA, feature scaling is critical.
- Principal components aren't easily interpreted features. We're not using a subset of the original features.

Overview