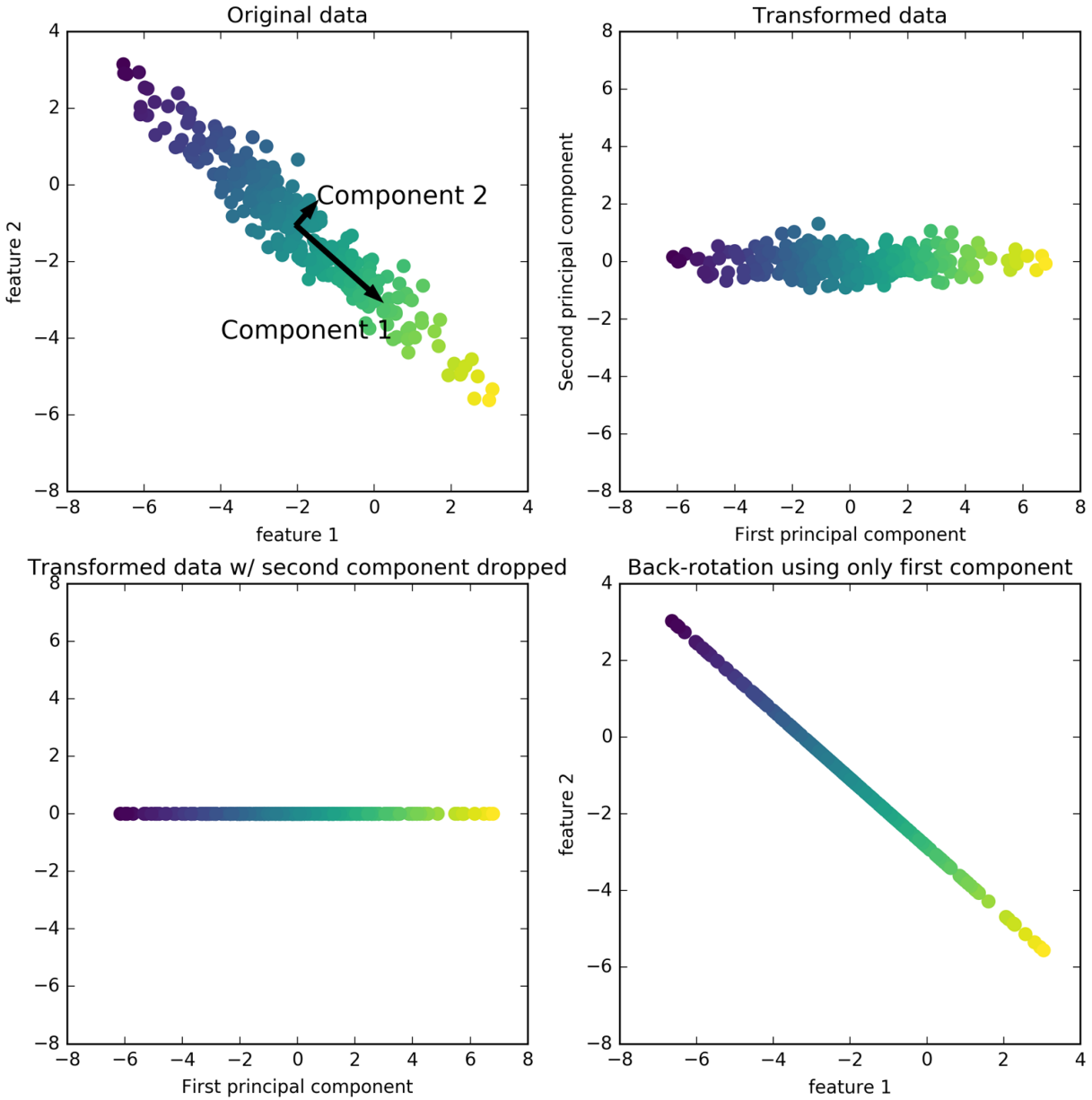
W4995 Applied Machine Learning

PCA, Discriminants, Manifold Learning

03/06/17

Andreas Müller

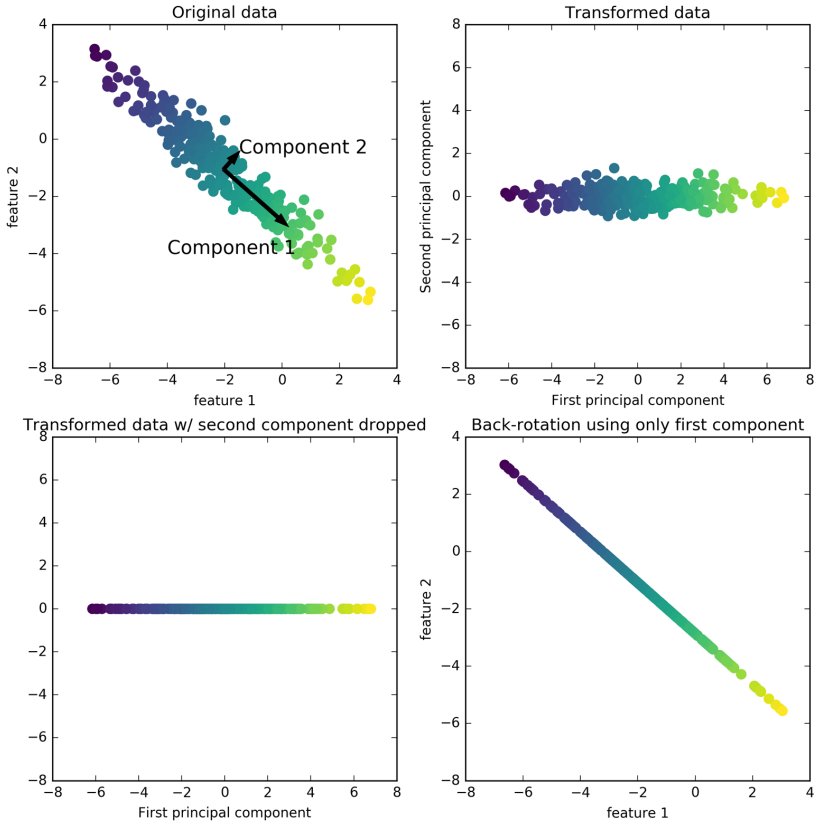
Principal Component Analysis


Original data
feature 2
feature 1
Component 2
Component 1
Transformed data
Second principal component
First principal component
Transformed data w/ second component dropped
First principal component
Back-rotation using only first component
feature 2
feature 1

PCA Objective(s)

Restricted rank reconstruction

$$\min_{X', \text{rank}(X')=r} ||X - X'||$$



PCA Objective(s)

Find directions of maximum variance:

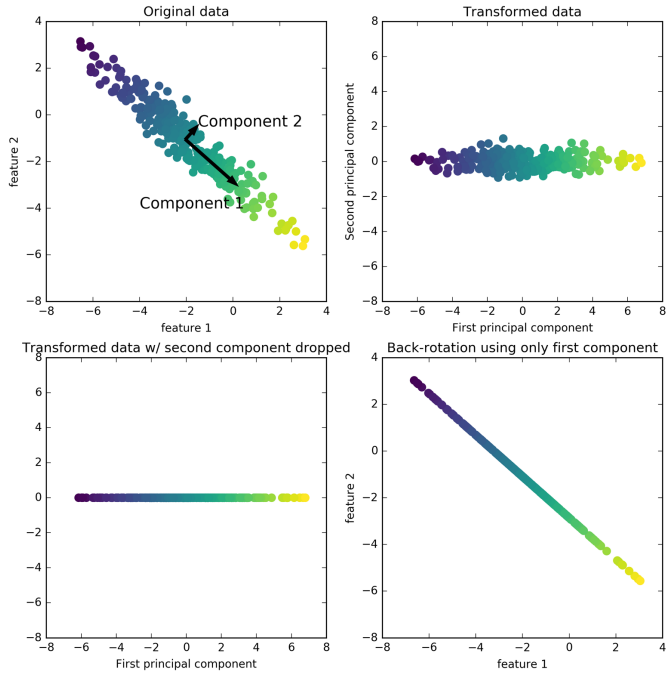
$$\max_{u_1 \in \mathbf{R}^p, ||u_1||=1} \text{var}(Xu_1)$$

Find projection (onto one vector) that maximizes the variance observed in the data.

$$\max_{u_1 \in \mathbf{R}^p, ||u_1||=1} u_1^T \text{cov}(X) u_1$$

Subtract projection onto u1, iterate to find more components.

Only well-defined up to sign / direction of arrow!

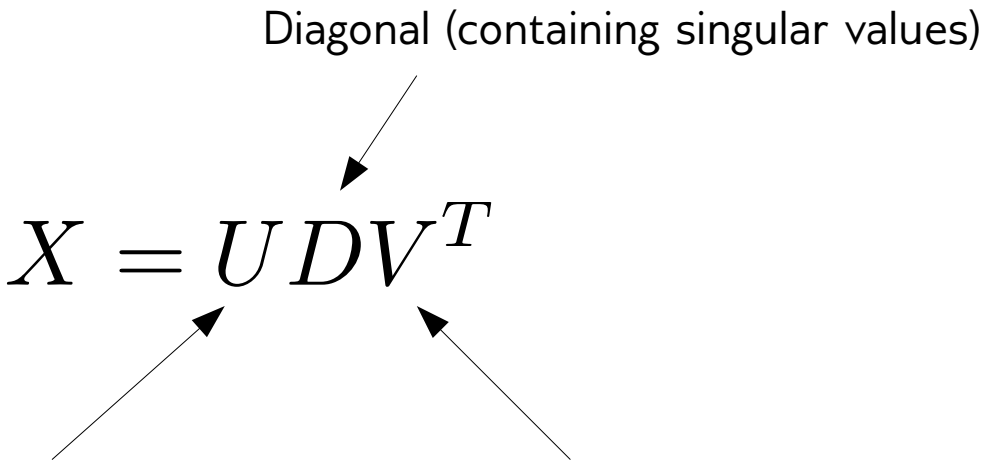


PCA Computation

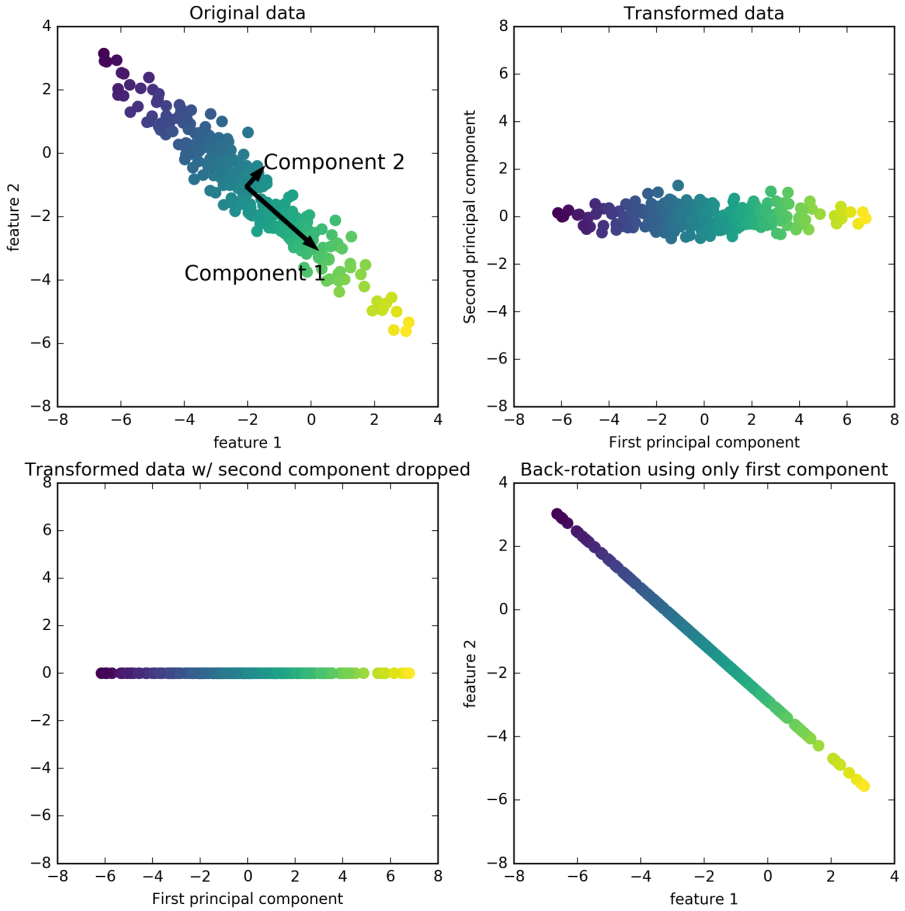
Center X (subtract mean)

In practice: Also scale to unit variance.

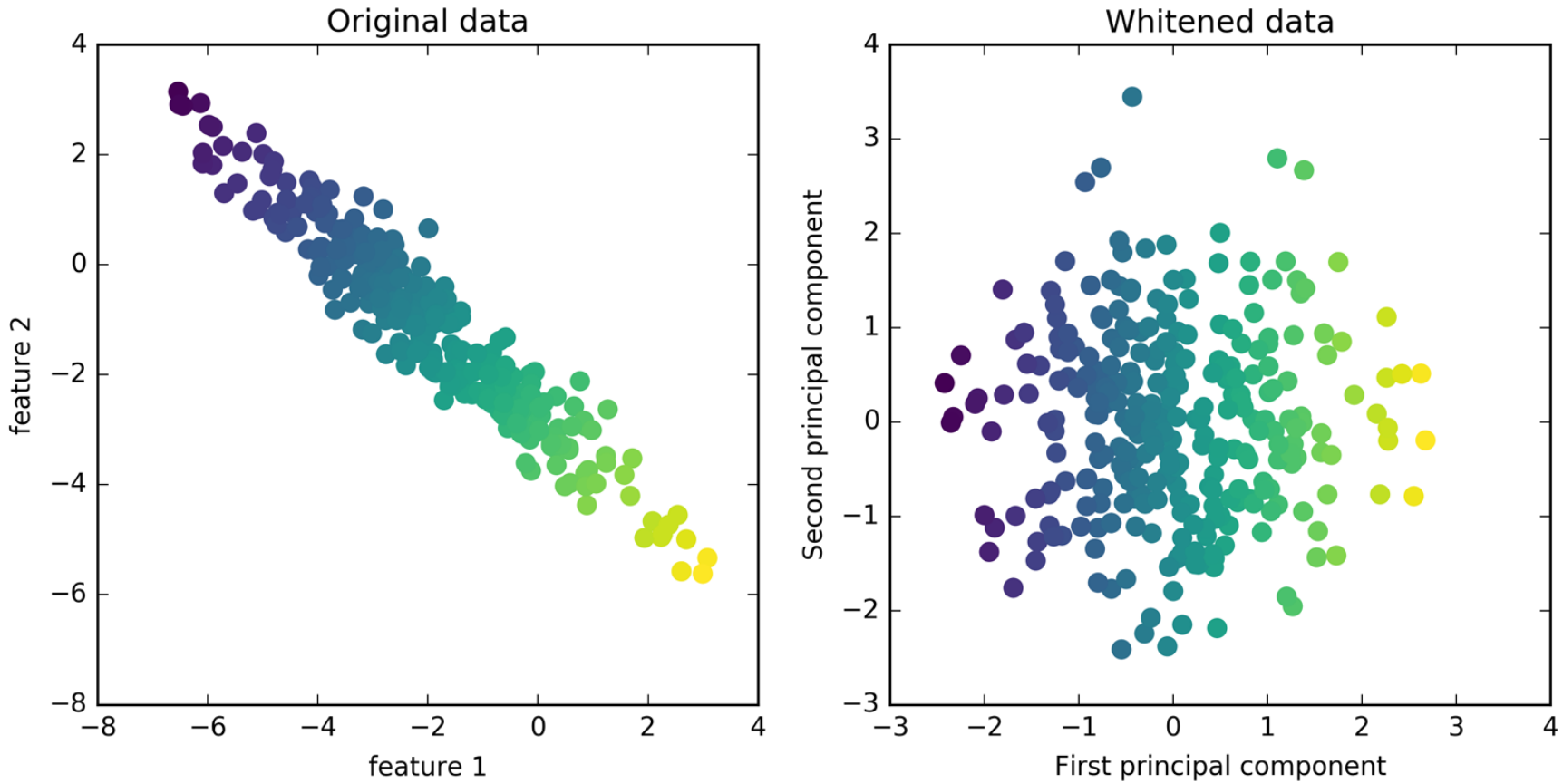
Compute singular value decomposition:



Back to intuition



Whitening



Same as using PCA without whitening, then doing StandardScaler

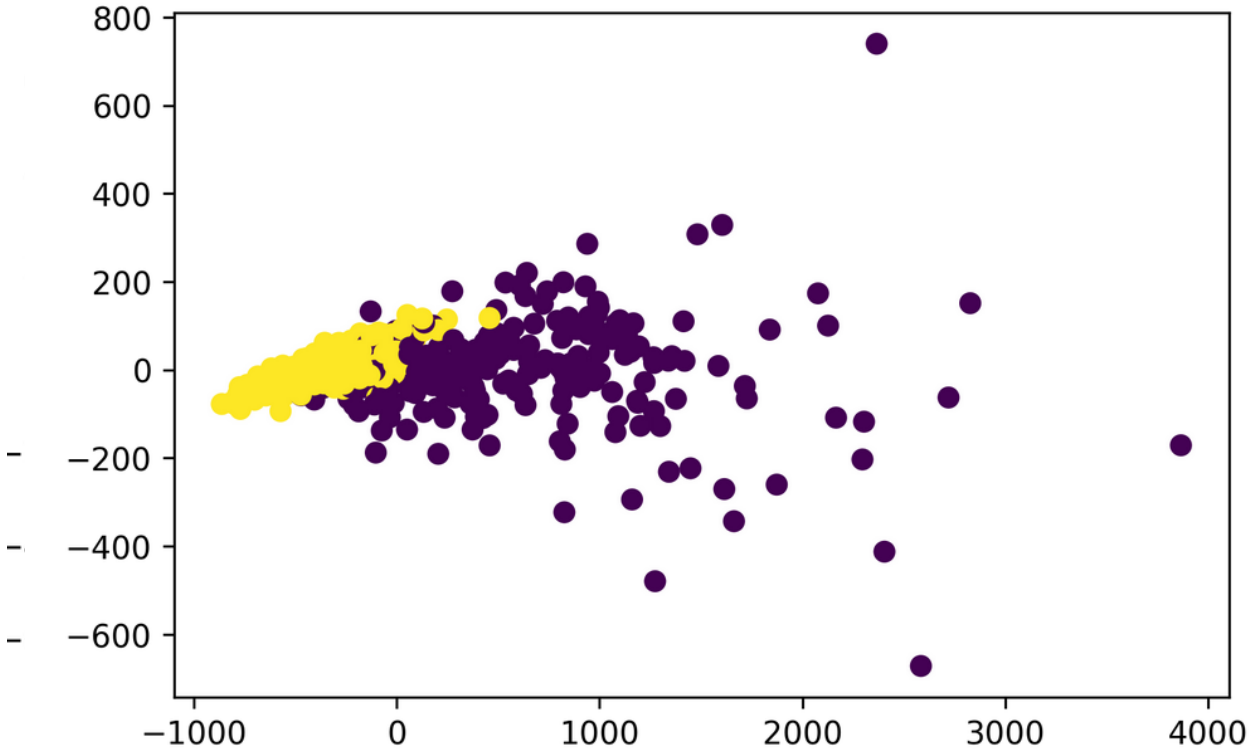
PCA for visualization

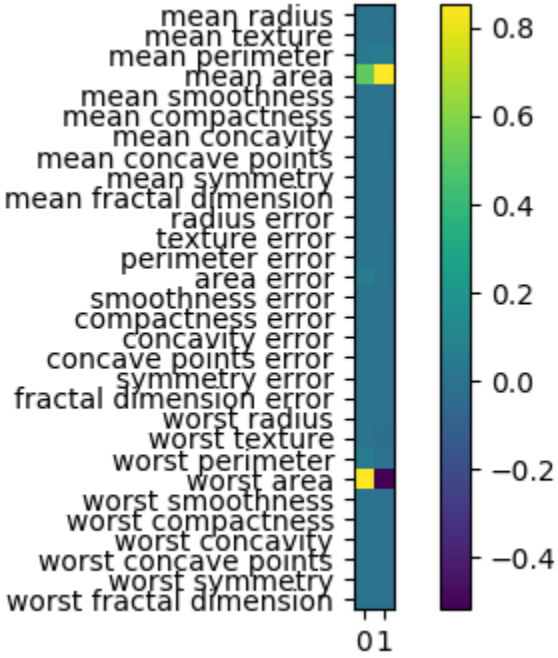
```
from sklearn.decomposition import PCA
print(cancer.data.shape)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(cancer.data)
plt.scatter(X_pca[:, 0], X_pca[:, 1])
print(X_pca.shape)
```

```
(569, 30)
(569, 2)
```

```
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=cancer.target)
```

```
<matplotlib.collections.PathCollection at 0x7fd884260e80>
```

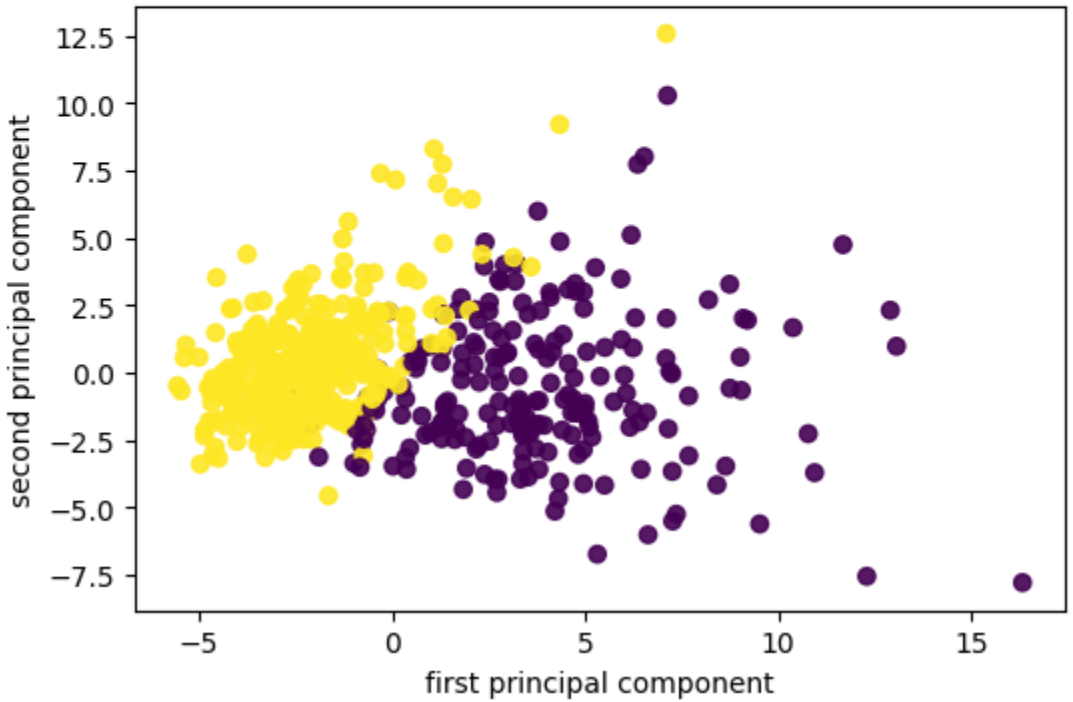




Scaling!

```
pca_scaled = make_pipeline(StandardScaler(), PCA(n_components=2))
X_pca_scaled = pca_scaled.fit_transform(cancer.data)
plt.scatter(X_pca_scaled[:, 0], X_pca_scaled[:, 1], c=cancer.target, alpha=.9)
plt.xlabel("first principal component")
plt.ylabel("second principal component")
```

```
<matplotlib.text.Text at 0x7fd87ebe1a90>
```

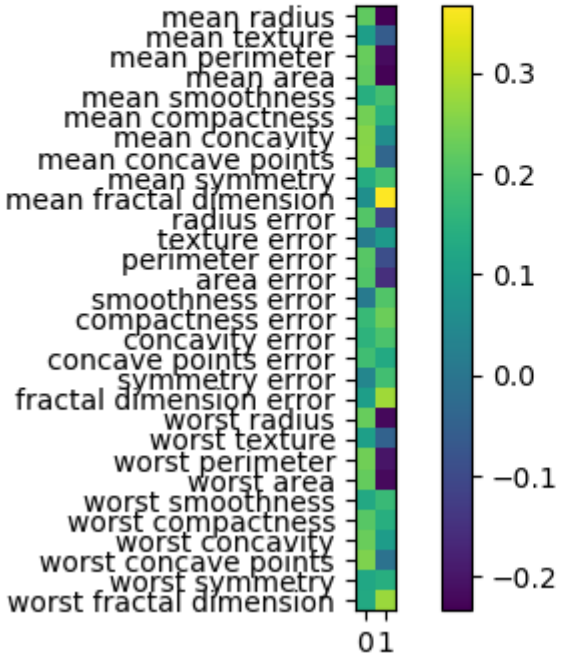


Imagine one feature with very large scale. Without scaling, it's guaranteed to be the first principal component!

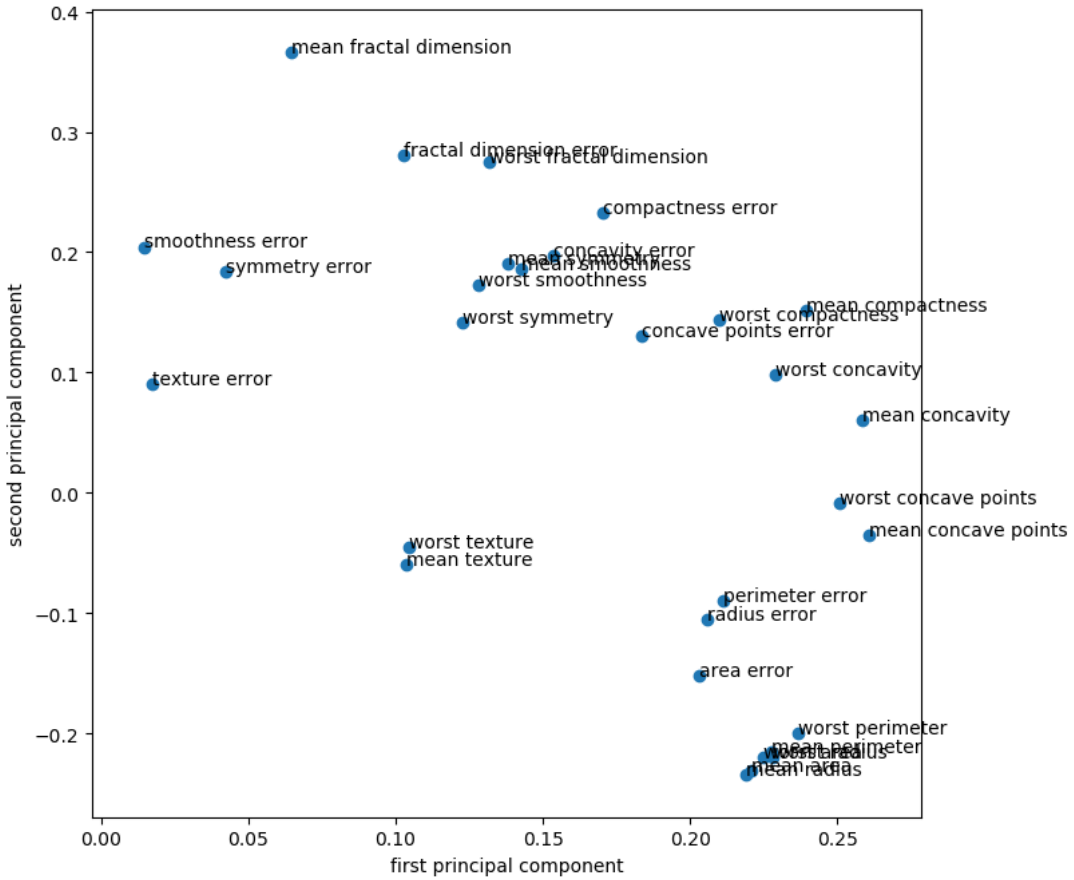
Inspecting Components

```
components = pca_scaled.named_steps['pca'].components_
plt.imshow(components.T)
plt.yticks(range(len(cancer.feature_names)), cancer.feature_names)
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x7fd87e032c88>
```



Direction (sign) of component is meaningless!



PCA for regularization

```
:
from sklearn.linear_model import LogisticRegression
```

```
:
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target, stratify=cancer.target, random_state=0)
```

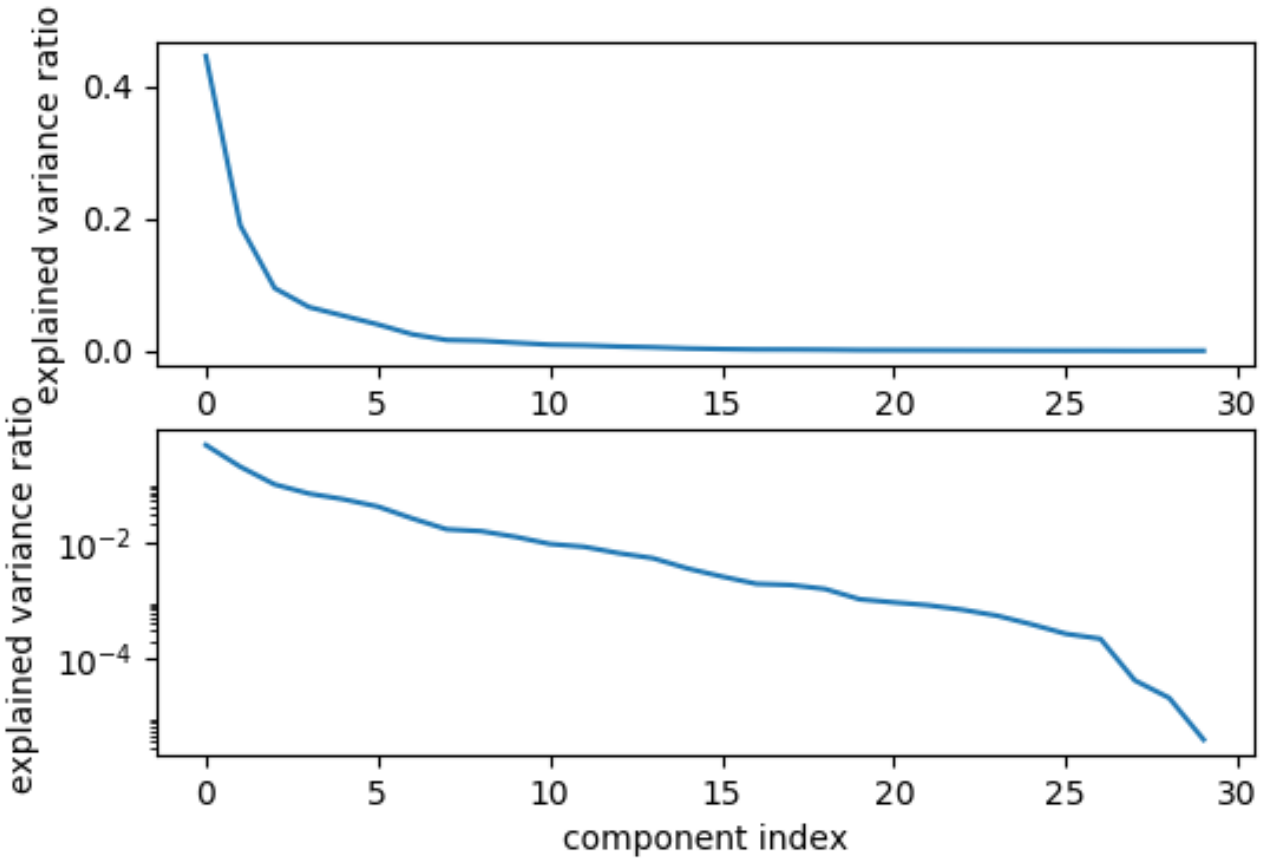
```
:
lr = LogisticRegression(C=10000).fit(X_train, y_train)
print(lr.score(X_train, y_train))
print(lr.score(X_test, y_test))
```

```
0.992957746479
0.944055944056
```

```
:
pca_lr = make_pipeline(StandardScaler(), PCA(n_components=2), LogisticRegression())
pca_lr.fit(X_train, y_train)
print(pca_lr.score(X_train, y_train))
print(pca_lr.score(X_test, y_test))
```

```
0.962441314554
0.923076923077
```

Variance covered




```
pca_lr = make_pipeline(StandardScaler(), PCA(n_components=6), LogisticRegression(C=10000))
pca_lr.fit(X_train, y_train)
print(pca_lr.score(X_train, y_train))
print(pca_lr.score(X_test, y_test))
```

```
0.981220657277
0.958041958042
```

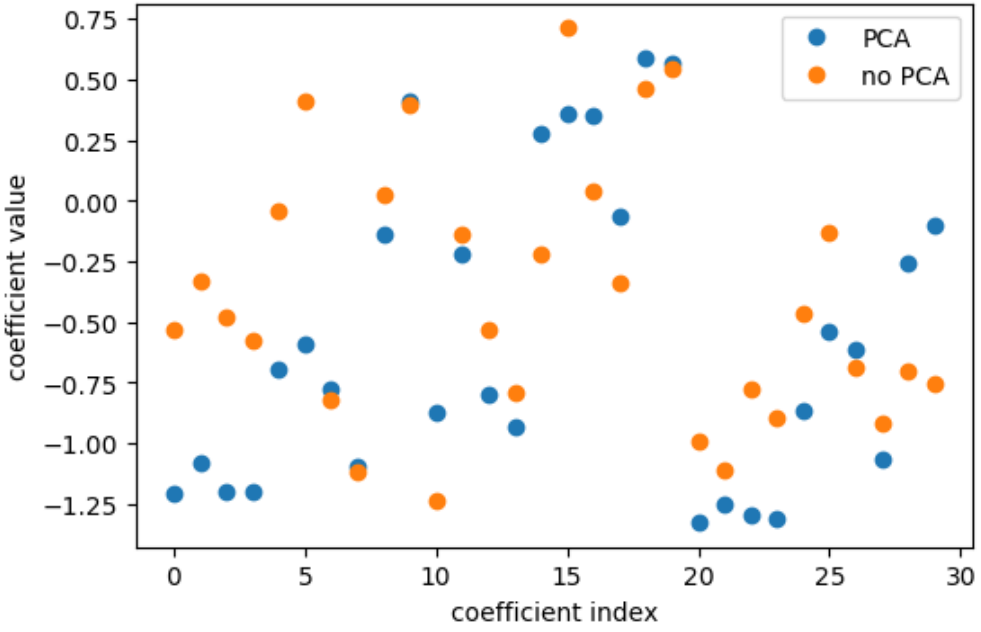
Interpreting coefficients

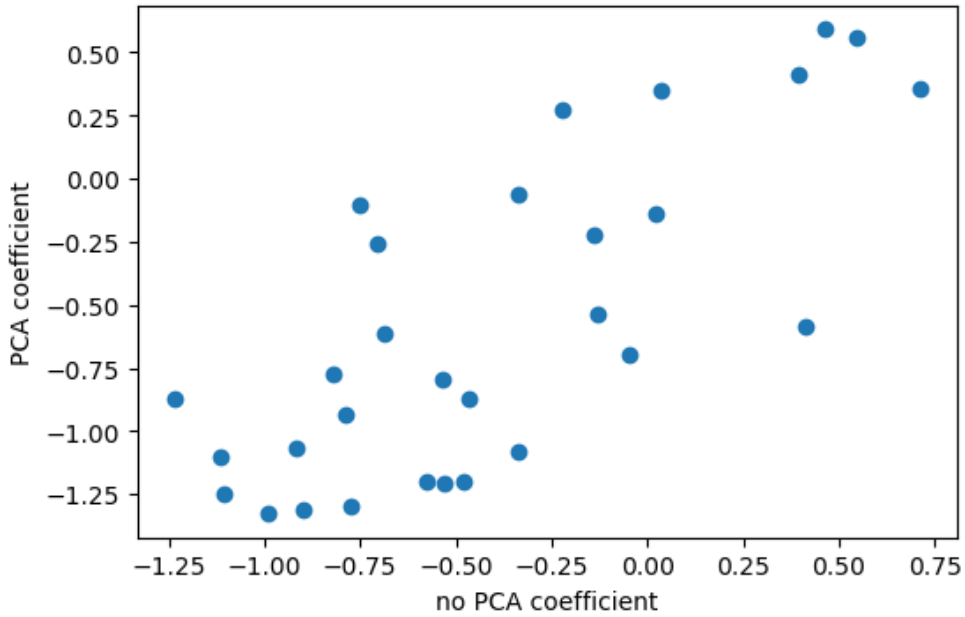
```
pca = pca_lr.named_steps['pca']
lr = pca_lr.named_steps['logisticregression']
```

```
coef_pca = pca.inverse_transform(lr.coef_)
```

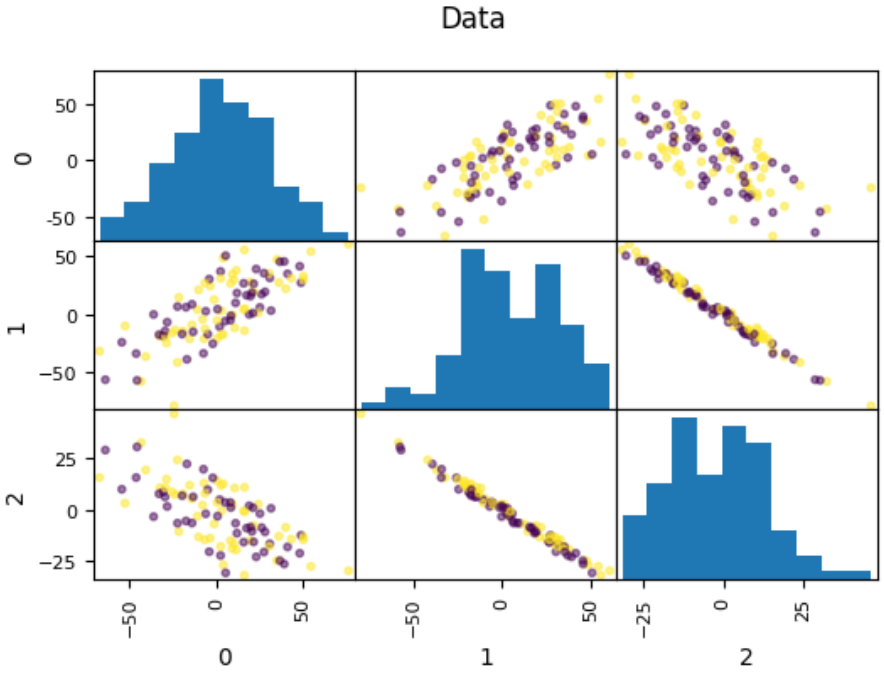
Rotating coefficients back into input space.
Makes sense because model is linear!
Otherwise more tricky.

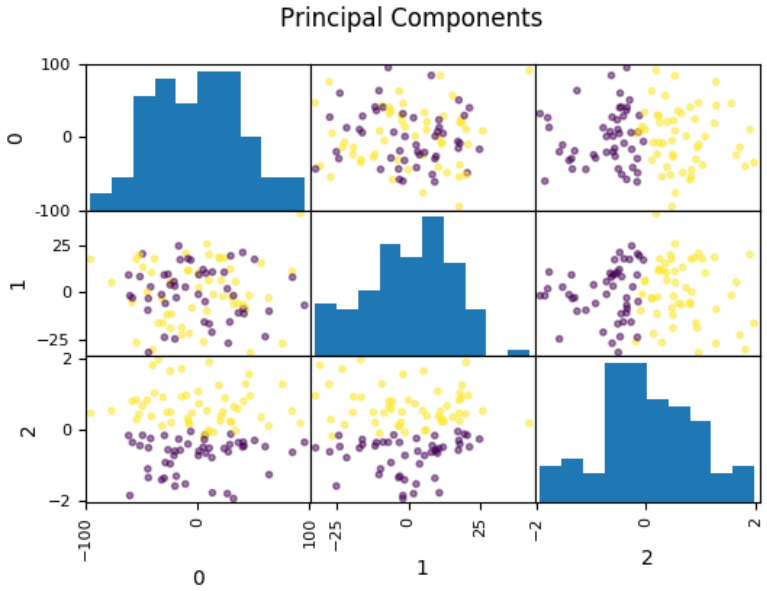
Comparing PCA + Logreg vs plain Logreg:





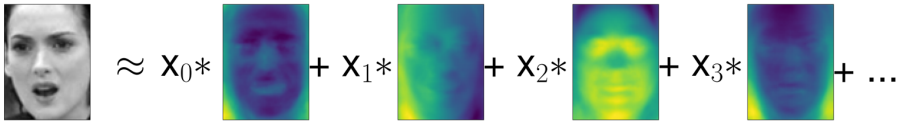
PCA is Unsupervised!

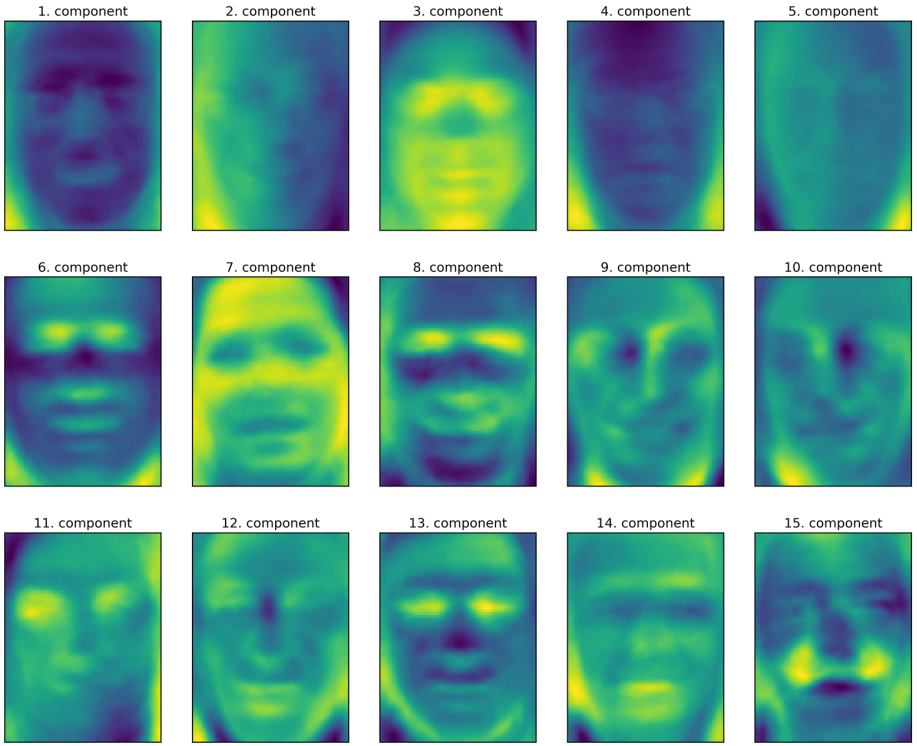




Dropping the first two principal components will result in random model!
All information is in the smallest principal component!

PCA for feature extraction





Remember:
Signs don't mean anything!

1-NN and Eigenfaces

```
from sklearn.neighbors import KNeighborsClassifier
# split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X_people, y_people, stratify=y_people, random_state=0)
# build a KNeighborsClassifier using one neighbor
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
print("Test set score of 1-nn: {:.2f}".format(knn.score(X_test, y_test)))
```

```
Test set score of 1-nn: 0.23
```

```
pca = PCA(n_components=100, whiten=True, random_state=0).fit(X_train)
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
print("X_train_pca.shape: {}".format(X_train_pca.shape))
```

```
X train pca.shape: (1547, 100)
```

```
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train_pca, y_train)
print("Test set accuracy: {:.2f}".format(knn.score(X_test_pca, y_test)))
```

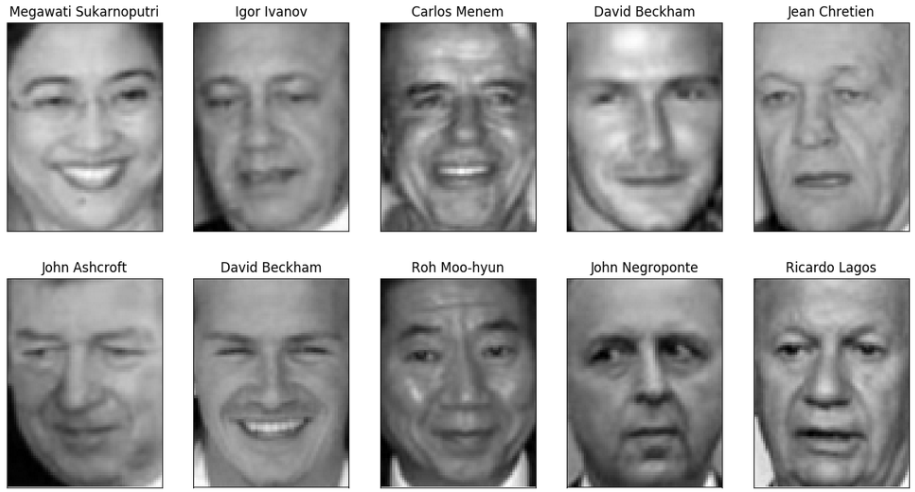
```
Test set accuracy: 0.31
```

Reconstruction

PCA for outlier detection

```
pca = PCA(n_components=100).fit(X_train)
```

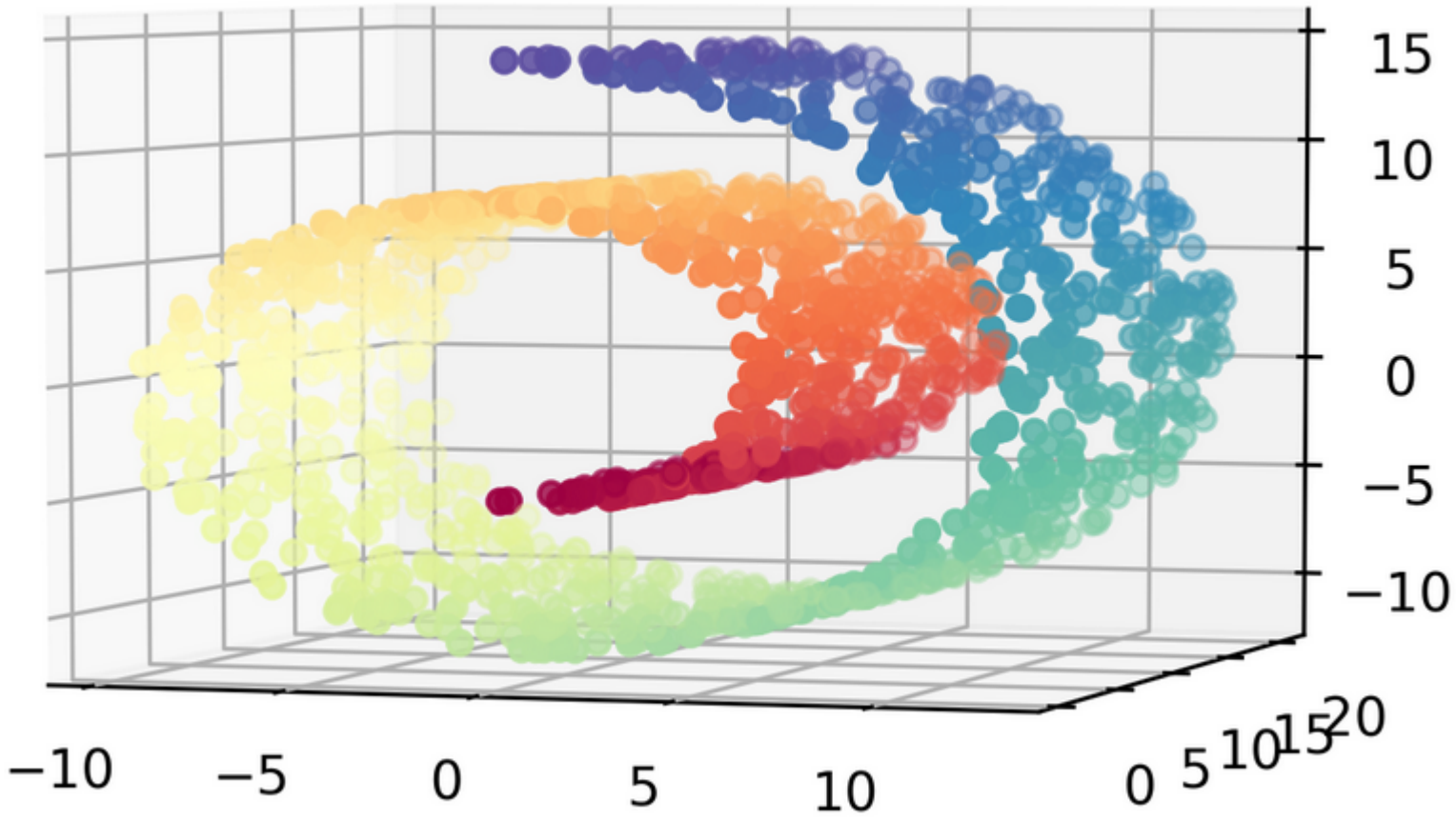
```
reconstruction_errors = np.sum((X_test - pca.inverse_transform(pca.transform(X_test))) ** 2, axis=1)
```



Best reconstructions

Worst reconstructions

Manifold Learning



Learn underlying “manifold” structure, use for dimensionality reduction

Pros and Cons

- For visualization only
- Axes don't correspond to anything in the input space.
- Often can't transform new data.
- Pretty pictures!

Algorithms in sklearn

- KernelPCA – does PCA, but with kernels!

  Eigenvalues of kernel-matrix
- Spectral embedding (Laplacian Eigenmaps)

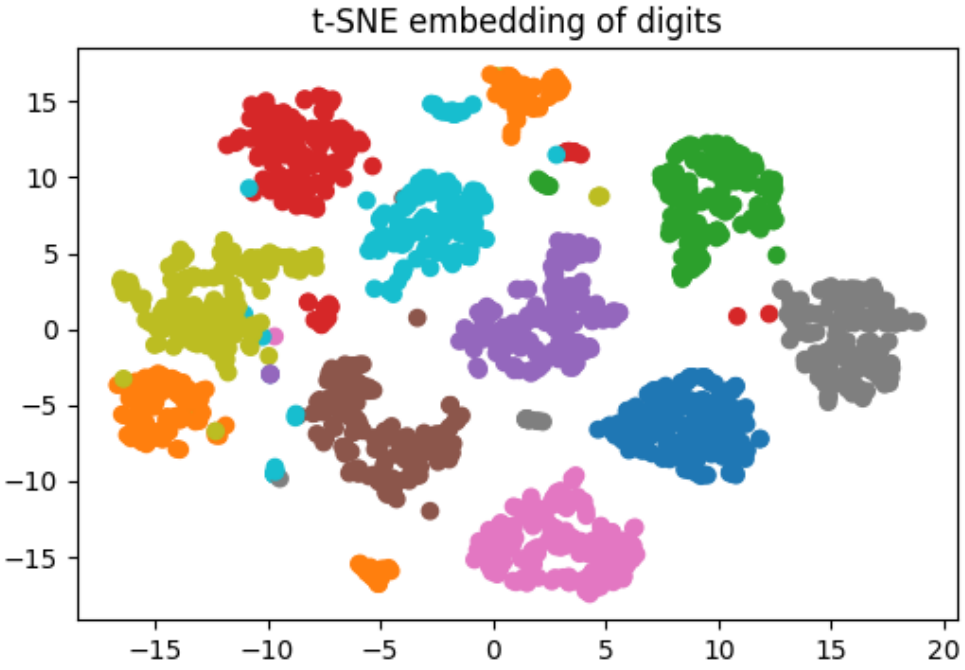
  Uses eigenvalues of graph laplacian
- Locally Linear Embedding
- Isomap “kernel PCA on manifold”
- t-SNE (t-distributed stochastic neighbor embedding)

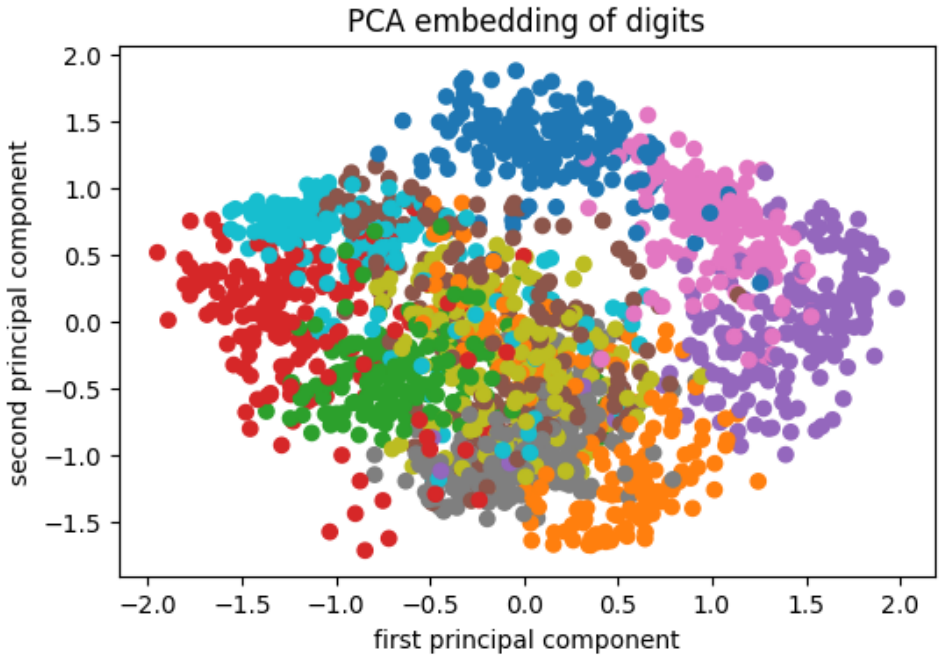
t-SNE

- Starts with a random embedding
- Iteratively updates points to make “close” points close.
- Global distances are less important, neighborhood counts.
- Good for getting coarse view of topology.
- Can be good for finding interesting data points.

```
:
from sklearn.manifold import TSNE
from sklearn.datasets import load_digits
digits = load_digits()
X = digits.data / 16.
```

```
:
X_tsne = TSNE().fit_transform(X)
X_pca = PCA(n_components=2).fit_transform(X)
```



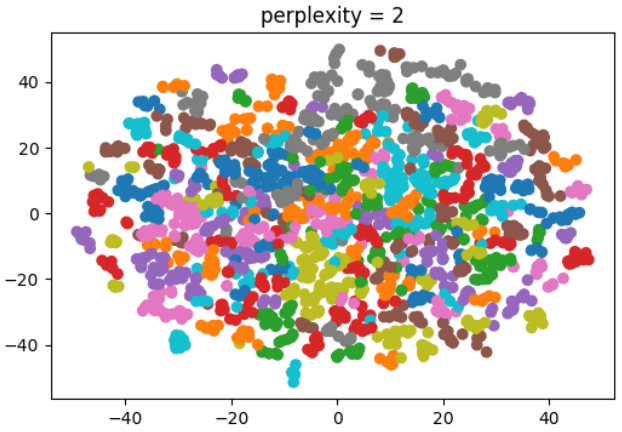


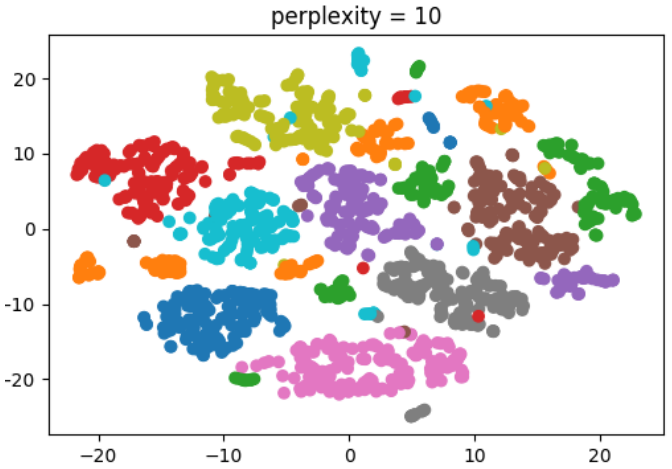
Tuning t-SNE

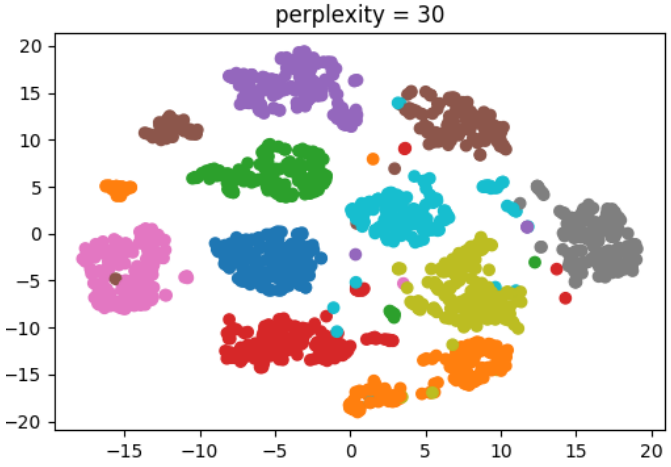
http://distill.pub/2016/misread-tsne/

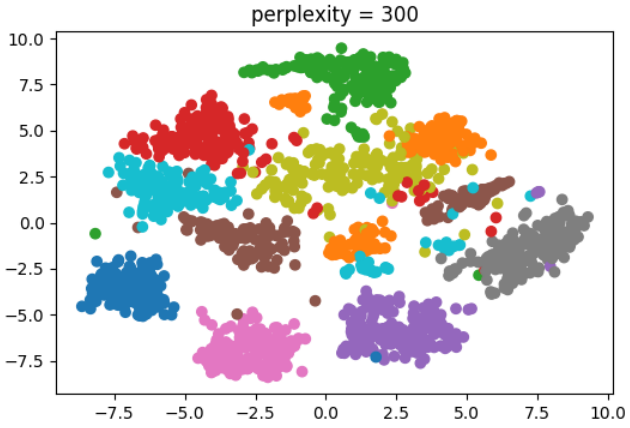
- Important parameter: perplexity
- Intuitively: bandwidth of neighbors to consider

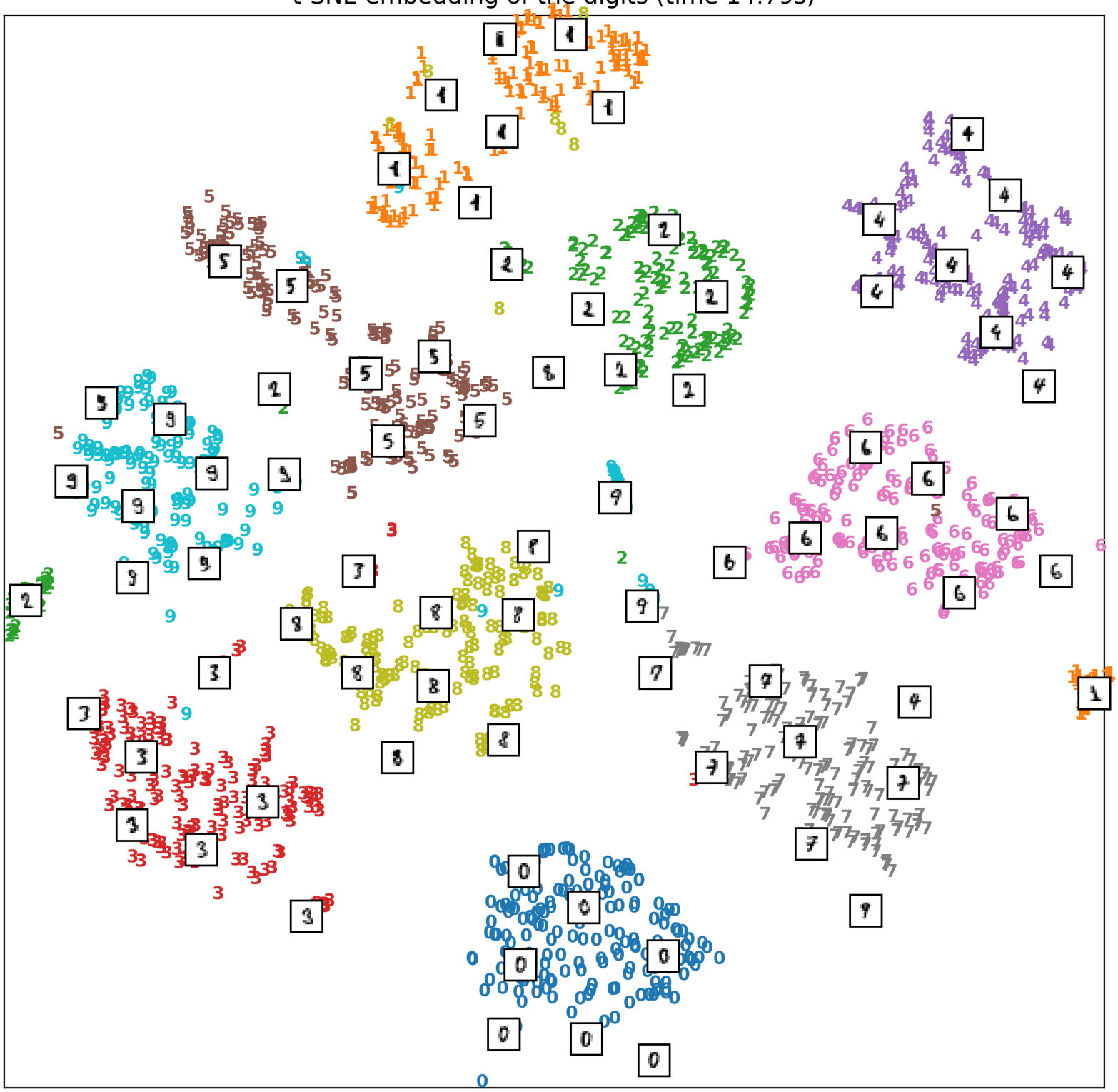
  (low perplexity: only close neighbors)

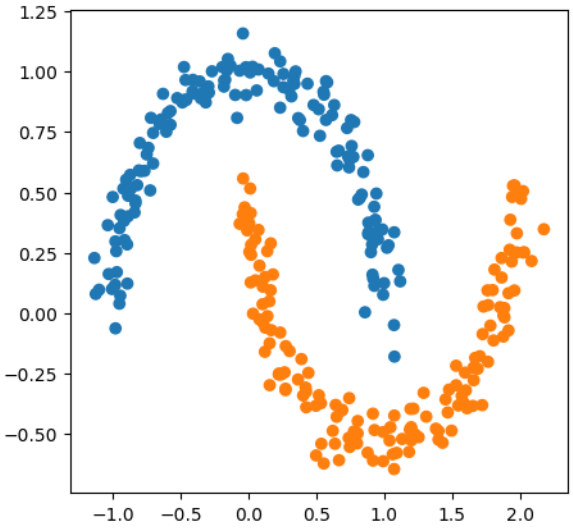


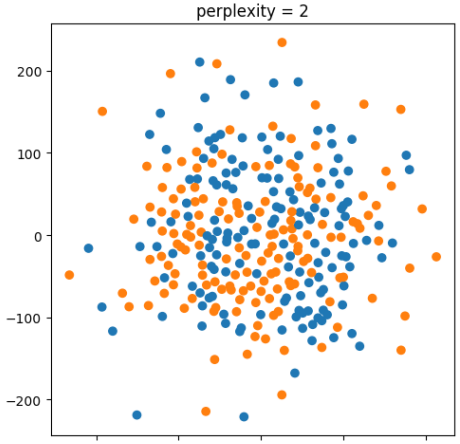


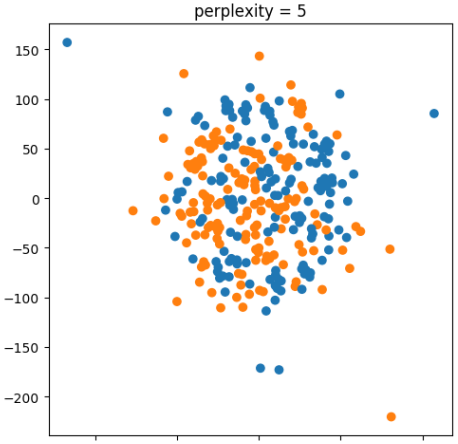


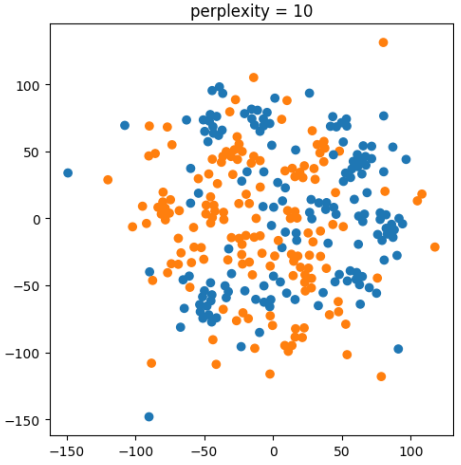


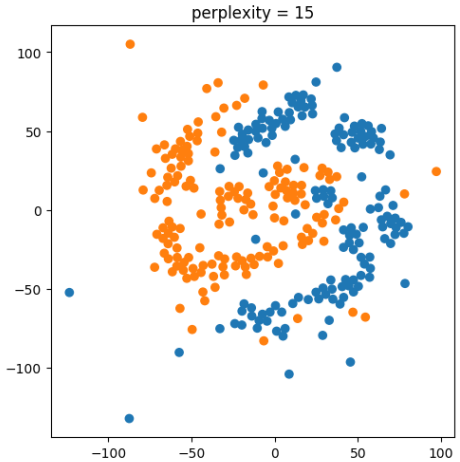

t-SNE embedding of the digits (time 14.79s)

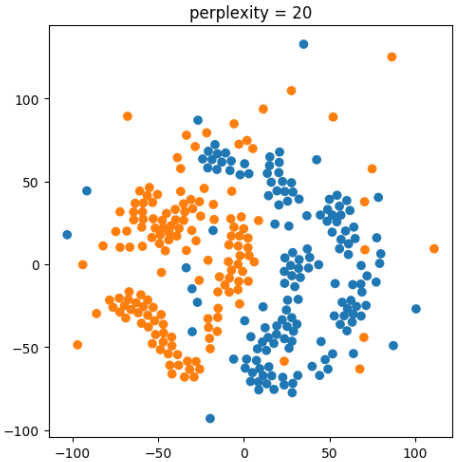


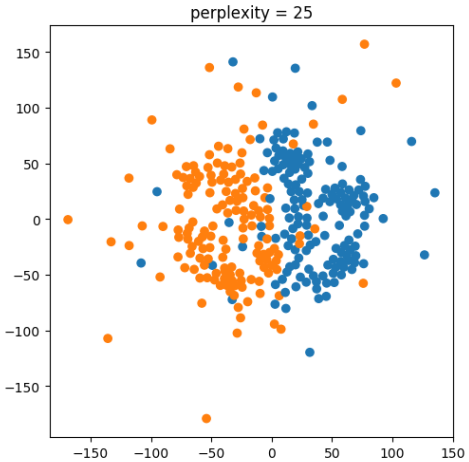

perplexity = 2

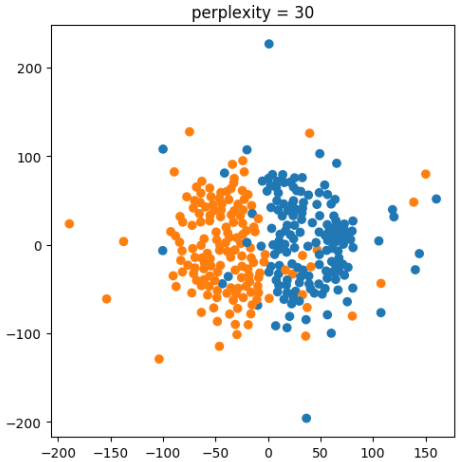

perplexity = 5

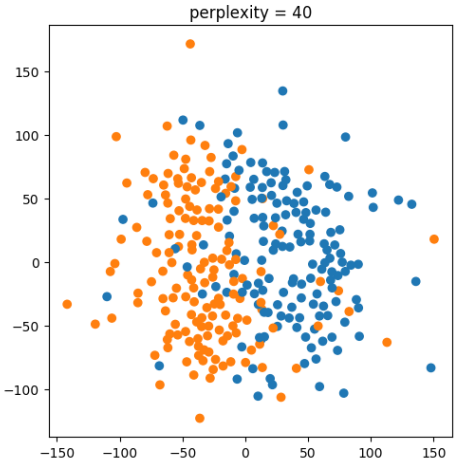

perplexity = 10

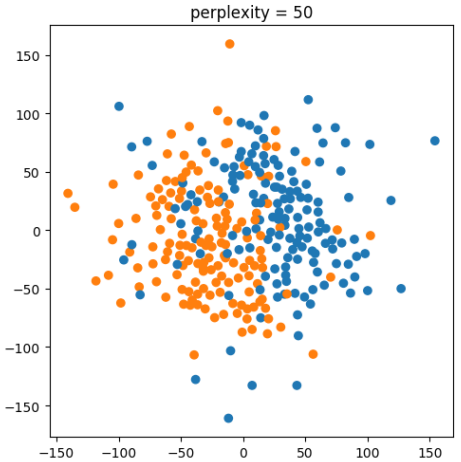

perplexity = 15


perplexity = 20


perplexity = 25


perplexity = 30


perplexity = 40


perplexity = 50

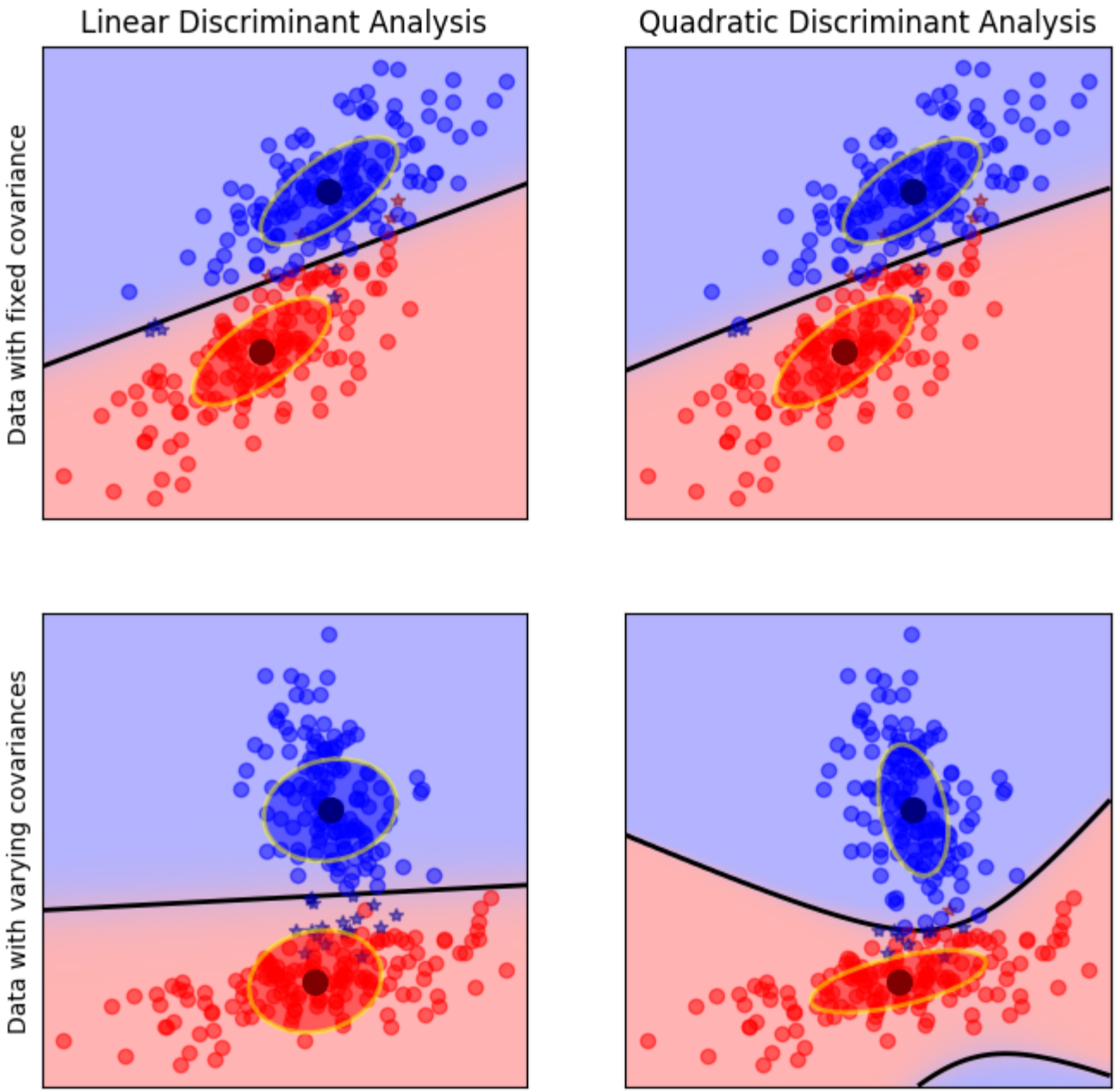
Discriminant Analysis

Linear Discriminant Analysis aka Fisher Discriminant

- Generative model: assumes each class has Gaussian distribution
- Covariances are the same for all classes.
- Very fast: only compute means and invert covariance matrix (works well if n_features << n_samples)
- Leads to linear decision boundary.
- Imagine: transform space by covariance matrix, then nearest centroid.
- No parameters to tune!
- Don't confuse with Latent Dirichlet Allocation (LDA)

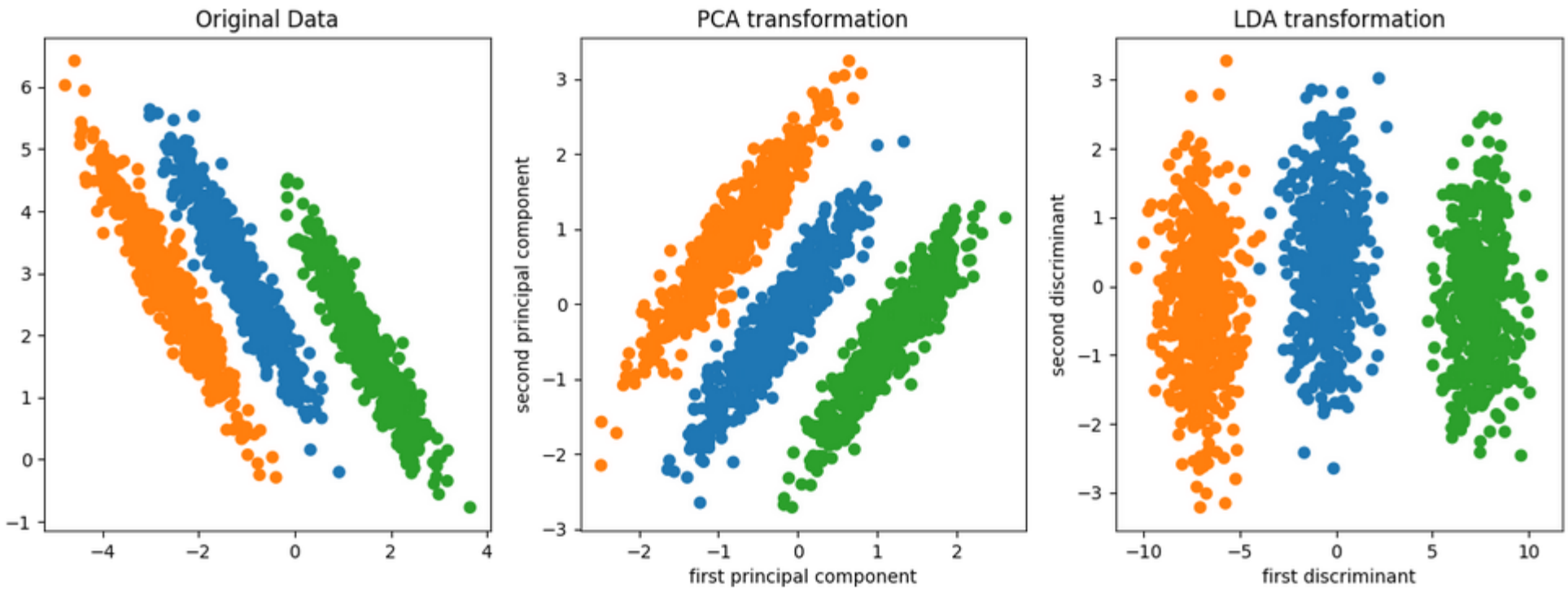
Quadratic Discriminant Analysis

- Each class is Gaussian, but separate covariance matrices!
- More flexible (quadratic decision boundary), but less robust: have less points per covariance matrix.
- Can't think of it as transformation of the space.

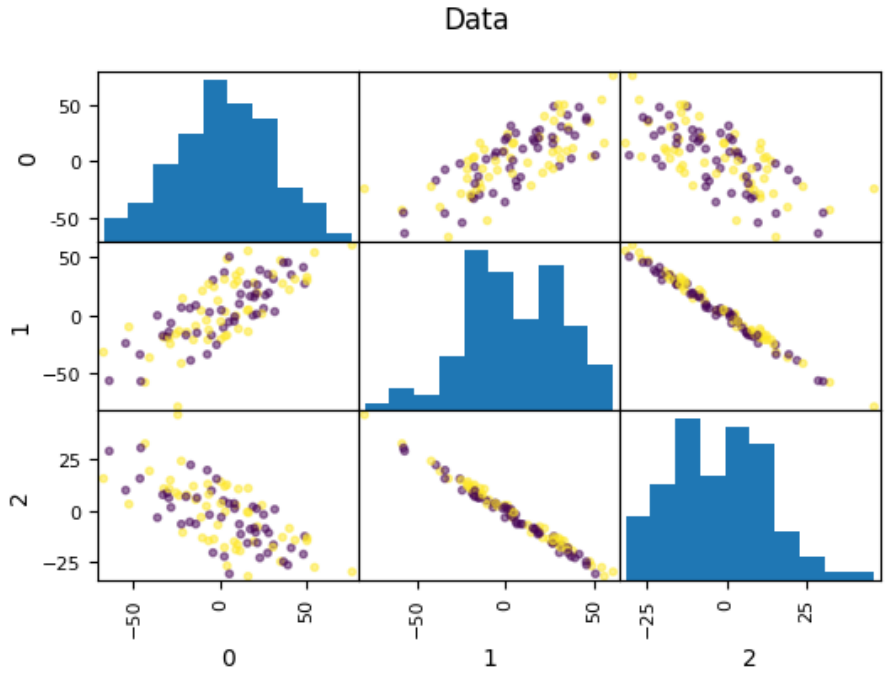

Linear Discriminant Analysis
Quadratic Discriminant Analysis
Data with fixed covariance
Data with varying covariances

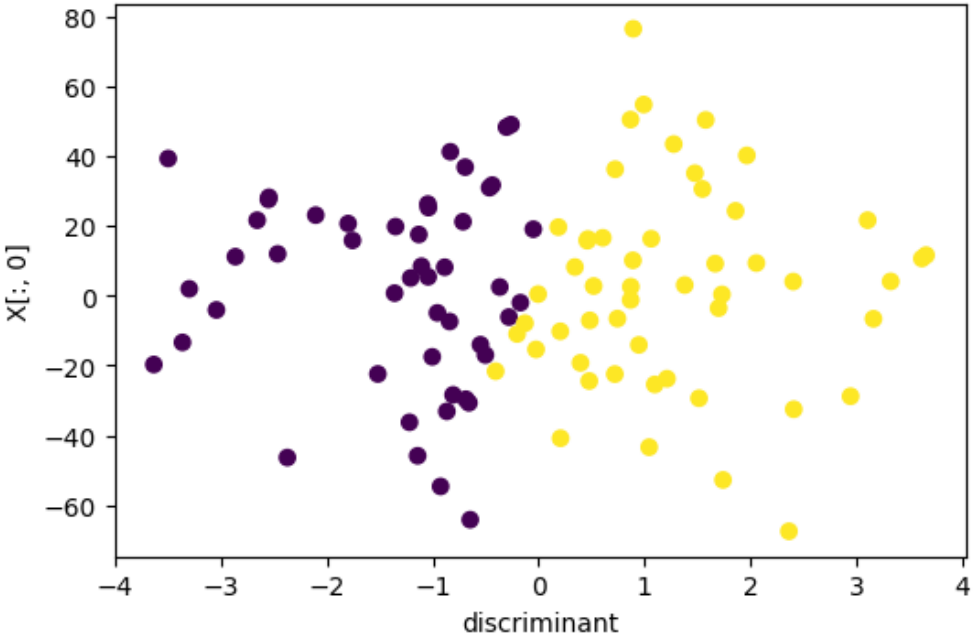
Discriminants and PCA

- Both fit Gaussian model
- PCA for the whole data
- LDA multiple Gaussians with shared covariance
- Can use LDA to transform space!
- At most as many components as there are classes (needs between class variance)


Original Data
PCA transformation
LDA transformation
second principal component
first principal component
second discriminant
first discriminant

Data where PCA failed





Summary

- PCA good for visualization, exploring correlations
- PCA can sometimes help with classification as regularization or for feature extraction.
- Manifold learning makes nice pictures.
- LDA is a handy supervised alternative to PCA that also yields a rotation of the input space.