W4995 Applied Machine Learning

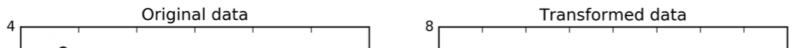
Dimensionality Reduction

PCA, Discriminants, Manifold Learning

03/25/19

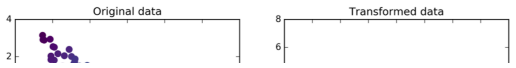
Andreas C. Müller

Principal Component Analysis


Original data
Transformed data
4
8

PCA objective(s)

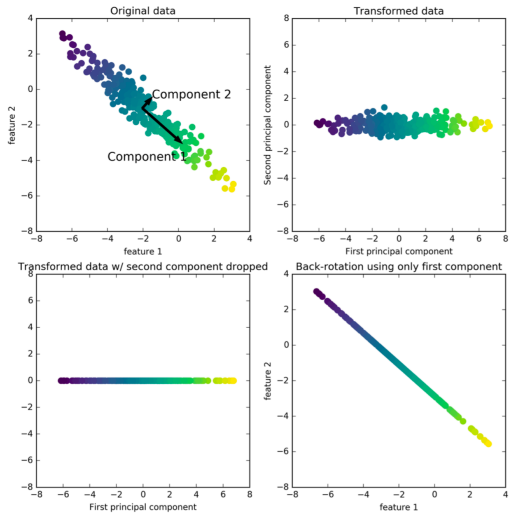
$$\min_{X', \text{rank}(X')=r} \|X - X'\|$$



PCA objective(s)

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

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



PCA Computation

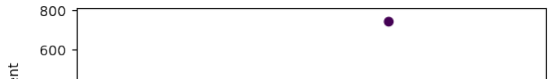
- Center X (subtract mean).
- In practice: Also scale to unit variance.
- Compute singular value decomposition:

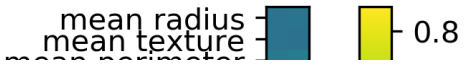
Diagonal (containing singular values)

Whitening

PCA for Visualization

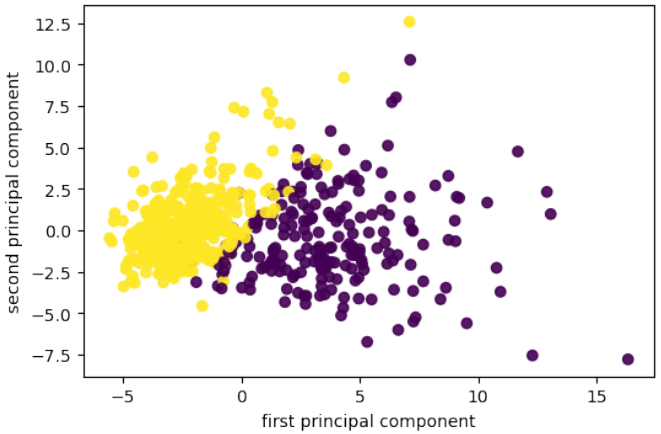
```
from sklearn.decomposition import PCA
print(cancer.data.shape)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(cancer.data)
print(X_pca.shape)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=cancer.target)
plt.xlabel("first principal component")
plt.ylabel("second principal component")
components = pca.components_
plt.imshow(components.T)
plt.yticks(range(len(cancer.feature_names)), cancer.feature_names)
plt.colorbar()
```





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")
```



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()
```

PCA for regularization

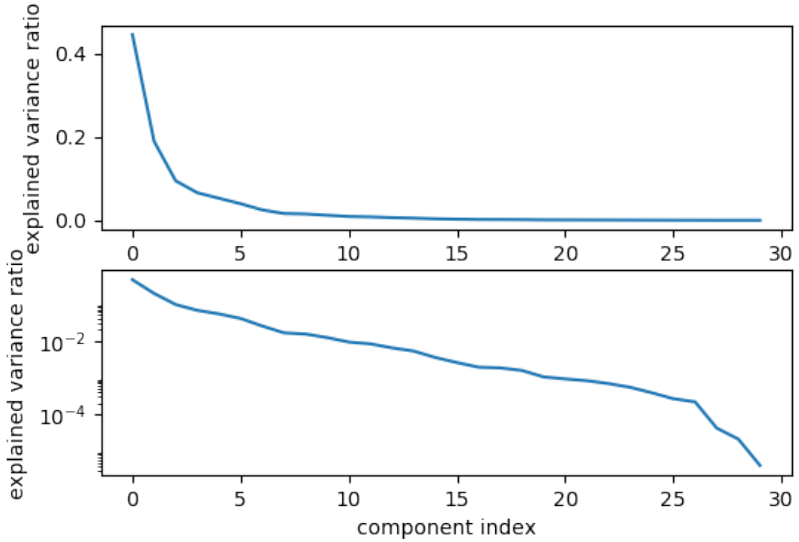
```
from sklearn.linear_model import LogisticRegression
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target, stratify=cancer.target, random_state=0)
lr = LogisticRegression(C=10000).fit(X_train, y_train)
print(lr.score(X_train, y_train))
print(lr.score(X_test, y_test))
```

```
0.993
0.944
```

```
pca_lr = make_pipeline(StandardScaler(), PCA(n_components=2), 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))
```

```
0.961
0.923
```

Variance covered



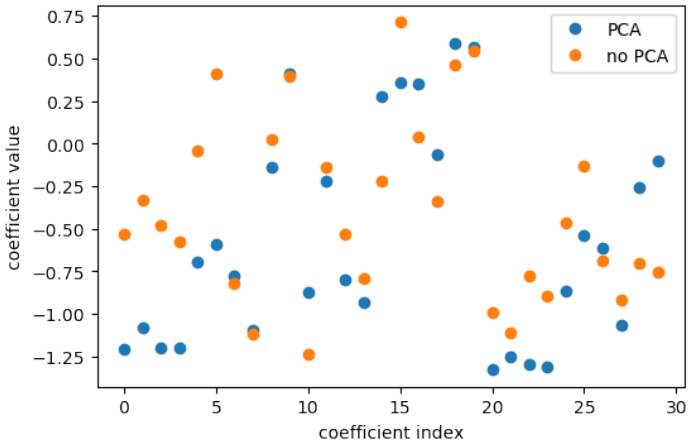

```
pca_lr = make_pipeline(StandardScaler(), PCA(n_components=6), LogisticRegression(C=10000))
pca_lr.fit(X_train, y_train)
print(pca_lr.score(X_train, y_train))
print(pca_lr.score(X_test, y_test))
```

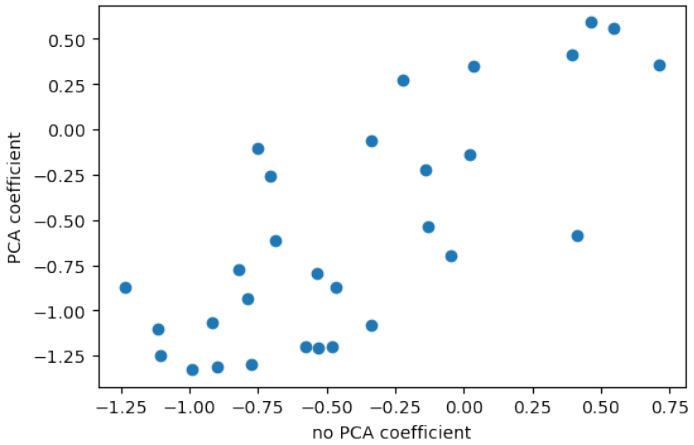
```
0.981
0.958
```

Interpreting coefficients

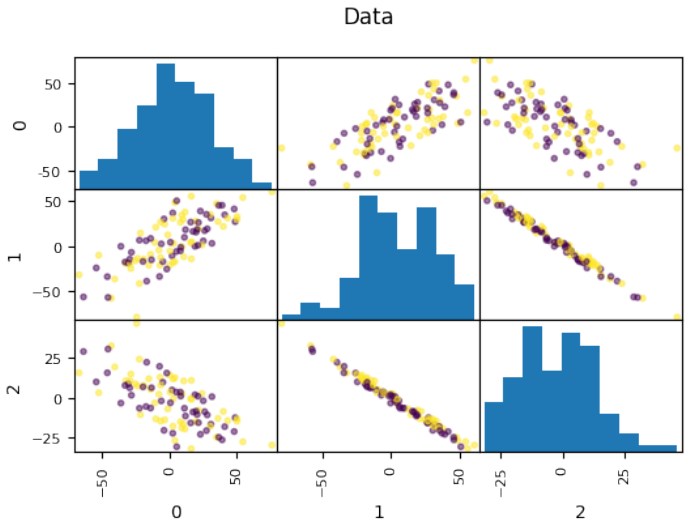
```
pca = pca_lr.named_steps['pca']
lr = pca_lr.named_steps['logisticregression']
coef_pca = pca.inverse_transform(lr.coef_)
```

Comparing PCA + Logreg vs plain Logreg:

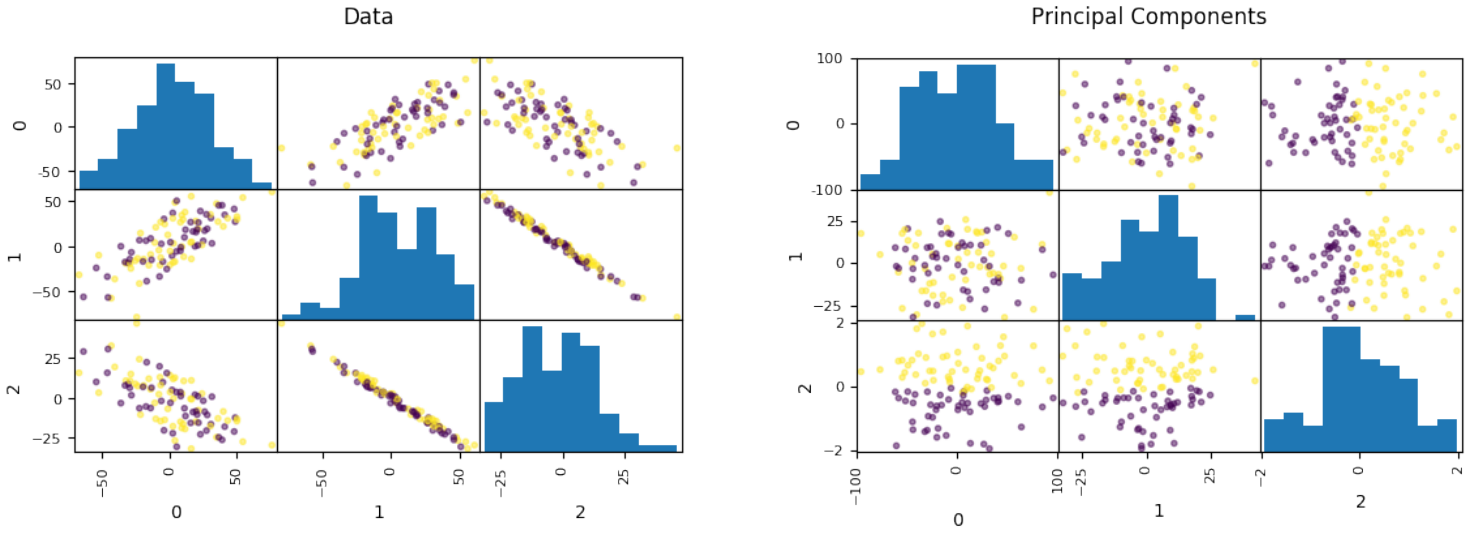




PCA is Unsupervised!



PCA is Unsupervised!



PCA for feature extraction

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)
print(X_train.shape)
# build a KNeighborsClassifier using one neighbor
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
knn.score(X_test, y_test)
```

```
(1547, 5655)
0.23
```

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)
print(X_train.shape)
# build a KNeighborsClassifier using one neighbor
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
knn.score(X_test, y_test)
```

```
(1547, 5655)
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)
X_train_pca.shape
```

```
(1547, 100)
```

```
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train_pca, y_train)
knn.score(X_test_pca, y_test))
```

```
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

Discriminant Analysis

Linear Discriminant Analysis aka Fisher Discriminant

$$P(y = k|X) = \frac{P(X|y = k)P(y = k)}{P(X)} = \frac{P(X|y = k)P(y = k)}{\sum_l P(X|y = l) \cdot P(y = l)}$$

Linear Discriminant Analysis aka Fisher Discriminant

$$P(y = k|X) = \frac{P(X|y = k)P(y = k)}{P(X)} = \frac{P(X|y = k)P(y = k)}{\sum_l P(X|y = l) \cdot P(y = l)}$$

$$p(X|y = k) = \frac{1}{(2\pi)^n |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(X - \mu_k)^t \Sigma^{-1} (X - \mu_k)\right)$$

Linear Discriminant Analysis aka Fisher Discriminant

$$P(y=k|X) = \frac{P(X|y=k)P(y=k)}{P(X)} = \frac{P(X|y=k)P(y=k)}{\sum_l P(X|y=l) \cdot P(y=l)}$$

$$p(X|y=k) = \frac{1}{(2\pi)^n |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(X-\mu_k)^t \Sigma^{-1}(X-\mu_k)\right)$$

$$\log\left(\frac{P(y=k|X)}{P(y=l|X)}\right) = 0 \Leftrightarrow (\mu_k - \mu_l)\Sigma^{-1}X = \frac{1}{2}(\mu_k^t \Sigma^{-1}\mu_k - \mu_l^t \Sigma^{-1}\mu_l)$$

Quadratic Discriminant Analysis

LDA:

$$p(X|y=k) = \frac{1}{(2\pi)^n |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(X-\mu_k)^t \Sigma^{-1}(X-\mu_k)\right)$$

QDA:

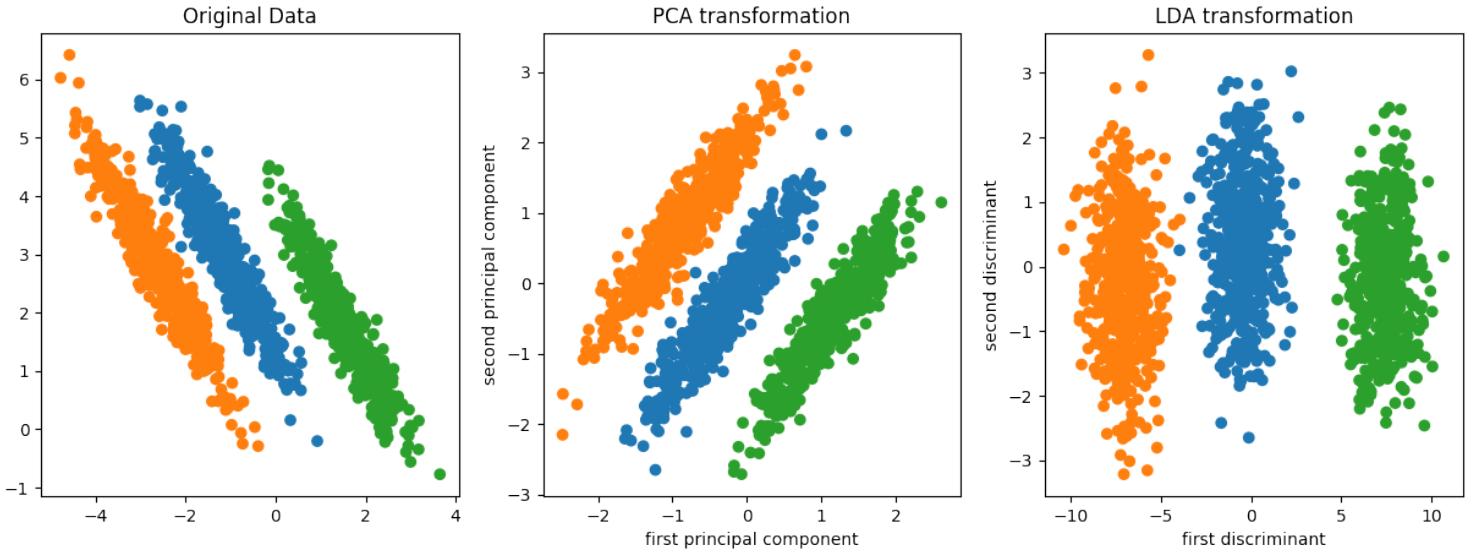
$$p(X|y=k) = \frac{1}{(2\pi)^n |\Sigma_k|^{1/2}} \exp\left(-\frac{1}{2}(X-\mu_k)^t \Sigma_k^{-1}(X-\mu_k)\right)$$

Linear Discriminant Analysis vs Quadratic Discriminant Analysis

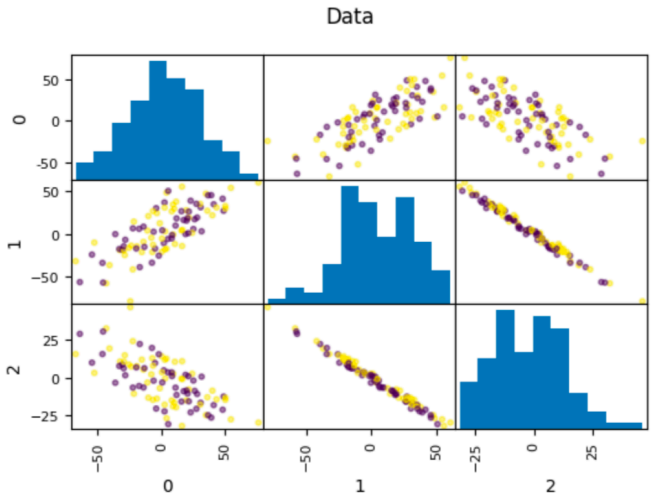
Discriminants and PCA

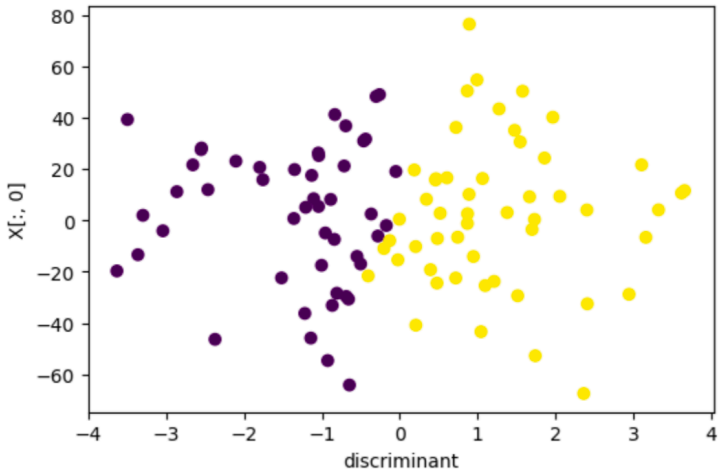
- Both fit Gaussian model
- PCA for the whole data
- LDA multiple Gaussians with shared covariance
- Can use LDA to transform space!
- At most as many components as there are classes - 1 (needs between class variance)

PCA vs Linear Discriminants



Data where PCA failed





Summary

- PCA good for visualization, exploring correlations
- PCA can sometimes help with classification as regularization or for feature extraction.
- LDA is a supervised alternative to PCA.

Manifold Learning

15 32 / 48

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

$$p_{j|i} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|\mathbf{x}_i - \mathbf{x}_k\|^2/2\sigma_i^2)}$$

t-SNE

$$p_{j|i} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|\mathbf{x}_i - \mathbf{x}_k\|^2/2\sigma_i^2)}$$

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}$$

t-SNE

$$p_{j|i} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|\mathbf{x}_i - \mathbf{x}_k\|^2/2\sigma_i^2)}$$

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}$$

$$q_{ij} = \frac{(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}}{\sum_{k \neq i}(1 + \|\mathbf{y}_i - \mathbf{y}_k\|^2)^{-1}}$$

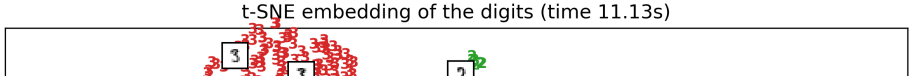
t-SNE

$$p_{j|i} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-\|\mathbf{x}_i - \mathbf{x}_k\|^2/2\sigma_i^2)}$$

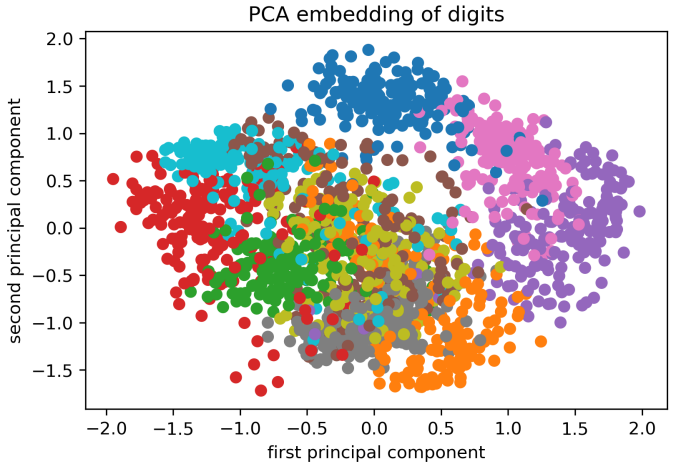
$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}$$

$$q_{ij} = \frac{(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}}{\sum_{k\neq i}(1 + \|\mathbf{y}_i - \mathbf{y}_k\|^2)^{-1}}$$

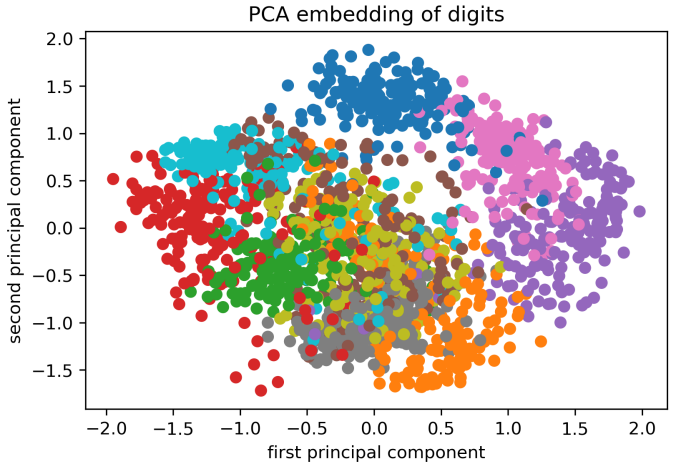
$$KL(P||Q) = \sum_{i\neq j} p_{ij}\log\frac{p_{ij}}{q_{ij}}$$

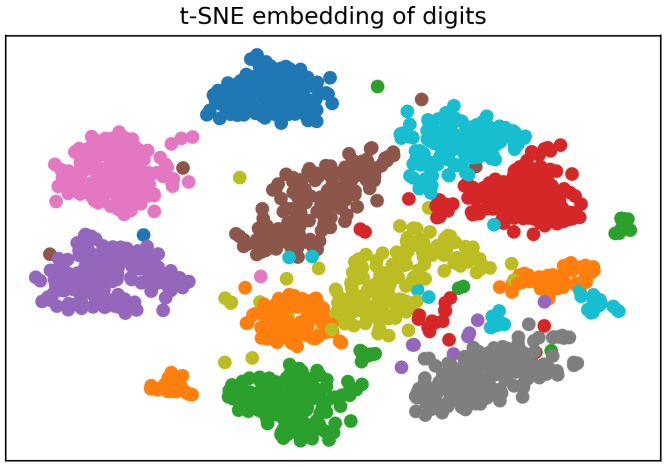

t-SNE embedding of the digits (time 11.13s)

```
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)
```



```
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)
```

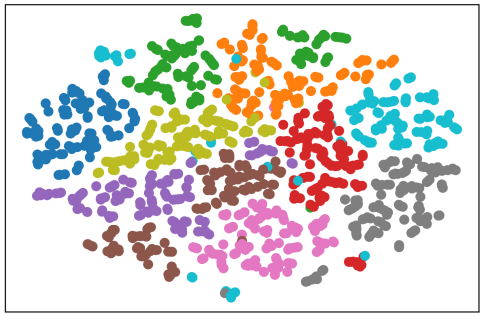




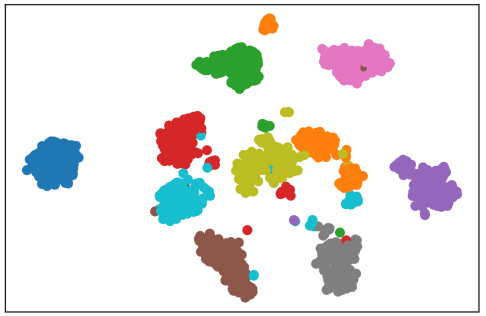
Showing algorithm progress

Tuning t-SNE perplexity

perplexity = 2

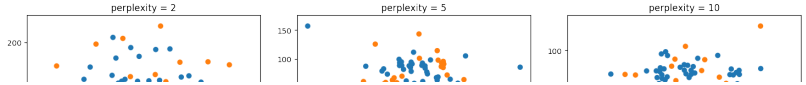


perplexity = 30



perplexity = 5

perplexity = 300


perplexity = 2
perplexity = 5
perplexity = 10
200
150
100
100

Play around online

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

UMAP

https://github.com/lmcinnes/umap

- Construct graph from simplices, use graph layout algorithm
- Much faster than t-SNE: MNIST takes 2 minutes vs 45 minutes for t-SNE

LargeVis

https://github.com/elbamos/largeVis

- Use nearest neighbor graph constructed very efficiently (and approximately)
- Much faster than t-SNE

Questions ?