15-388/688 - Practical Data Science: Unsupervised learning

J. Zico Kolter
Carnegie Mellon University
Spring 2018

Outline

Unsupervised learning

K-means

Pricinple Component Analysis

Outline

Unsupervised learning

K-means

Pricinple Component Analysis

Supervised learning paradigm

Training Data

$$\begin{matrix}(x^{(1)}, y^{(1)}) \\ (x^{(2)}, y^{(2)}) \\ (x^{(3)}, y^{(3)}) \\ \vdots\end{matrix}$$

Machine learning algorithm

Hypothesis function

$$y^{(i)} \approx h(x^{(i)})$$

Predictions

New example x

$$\hat{y} = h(x)$$

Unsupervised learning paradigm

Training Data

$$\left(x^{(1)}\right)$$
$$\left(x^{(2)}\right)$$
$$\left(x^{(3)}\right)$$
$$\vdots$$

Machine learning algorithm

Hypothesis function

$$? \approx h\left(x^{(i)}\right)$$

Predictions

New example x

$$? = h(x)$$

Three elements of unsupervised learning

It turns out the virtually all unsupervised learning algorithms can be considered in the same manner as supervised learning:

1. Define hypothesis function

2. Define loss function

3. Define how to optimize the loss function

But, what do a hypothesis function and loss function signify in the unsupervised setting?

Unsupervised learning framework

Input features: $x^{(i)} \in \mathbb{R}^n, i = 1, \ldots, m$

Model parameters: $\theta \in \mathbb{R}^k$

Hypothesis function: $h_\theta \colon \mathbb{R}^n \to \mathbb{R}^n$, approximates *input* given input, i.e. we want $x^{(i)} \approx h_\theta(x^{(i)})$

Loss function: $\ell \colon \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}_+$, measures the difference between a hypothesis and actual input, e.g.: $\ell(h_\theta(x), x) = \|h_\theta(x) - x\|_2^2$

Similar canonical machine learning optimization as before:

$$\text{minimize}_\theta \quad \sum_{i=1}^{m} \ell(h_\theta(x^{(i)}), x^{(i)})$$

Hypothesis and loss functions

The framework seems odd, what does it mean to have a hypothesis function approximate the input?

Can't we just pick $h_\theta(x) = x$?

The goal of unsupervised learning is to pick some *restricted* class of hypothesis functions that extract some kind of structure from the data (i.e., one that does not include the identity mapping above)

In this lecture, we'll consider two different algorithms that both fit the framework: k-means and principal component analysis

Outline

Unsupervised learning

K-means

Pricinple Component Analysis

K-means graphically

The k-means algorithm is easy to visualize: given some collection of data points we want to find k centers such that all points are close to at least one center

K-means in unsupervised framework

Parameters of k-means are the choice of centers $\theta = \{\mu^{(1)}, \ldots \mu^{(k)}\}$, with $\mu^{(i)} \in \mathbb{R}^n$

Hypothesis function outputs the center closest to a point x

$$h_\theta(x) = \underset{\mu \in \{\mu^{(1)}, \ldots \mu^{(k)}\}}{\operatorname{argmin}} \|\mu - x\|_2^2$$

Loss function is squared error between input and hypothesis

$$\ell(h_\theta(x), x) = \|h_\theta(x) - x\|_2^2$$

Optimization problem is thus

$$\underset{\mu^{(1)}, \ldots \mu^{(k)}}{\operatorname{minimize}} \sum_{i=1}^{m} \left\|h_\theta(x^{(i)}) - x^{(i)}\right\|_2^2$$

Optimizing k-means objective

The k-means objective is non-convex (possibility of local optima), and does not have a closed form solution, so we resort to an approximate method, by repeating the following (Lloyd's algorithm, or just "k-means")

1. Assign points to nearest cluster
2. Compute cluster center as mean of all points assigned to it

Given: Data set $\left(x^{(i)}\right)_{i=1,\ldots,m}$, # clusters k

Initialize:

$$\mu^{(j)} \leftarrow \text{Random}\left(x^{(i)}\right), \quad j = 1, \ldots, k$$

Repeat until convergence:

Compute cluster assignment:

$$y^{(i)} = \underset{j}{\text{argmin}} \left\| \mu^{(j)} - x^{(i)} \right\|_2^2, \; i = 1, \ldots, m$$

Re-compute means:

$$\mu^{(j)} \leftarrow \text{Mean}\left(\left\{x^{(i)} | y^{(i)} = j\right\}\right), j = 1, \ldots, k$$

Poll: k-means loss

When running the k-means algorithm (compute clusters for each point by finding closest center, set center to be mean of all associated points), what happens to the sum of loss after each iteration,

$$\sum_{i=1}^{m} \ell\left(h_\theta\left(x^{(i)}\right), x^{(i)}\right)$$

1. Each loss $\ell\left(h_\theta\left(x^{(i)}\right), x^{(i)}\right)$ will decrease for all i

2. The sum of losses will decrease

3. The sum of losses may increase or decrease

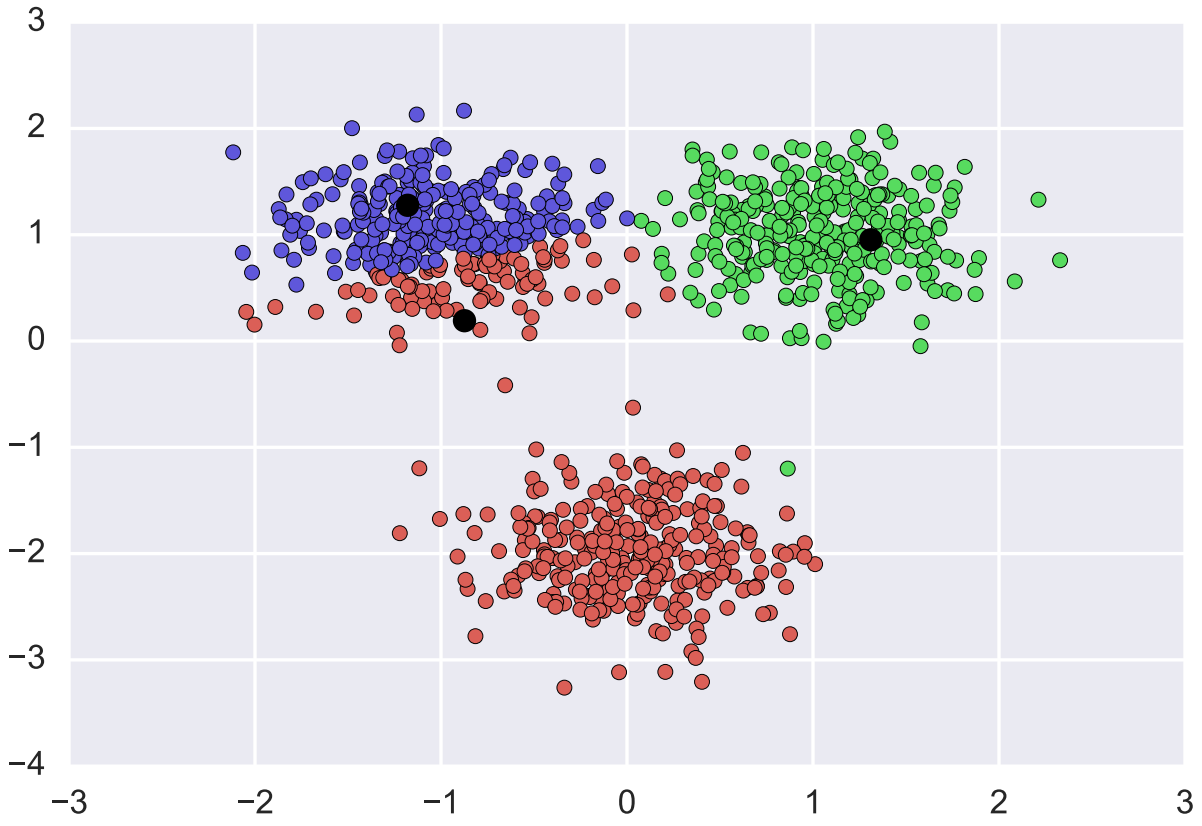
K-means in a few lines of code

Scikit-learn, etc, contains k-means implementations, but again these are pretty easy to write

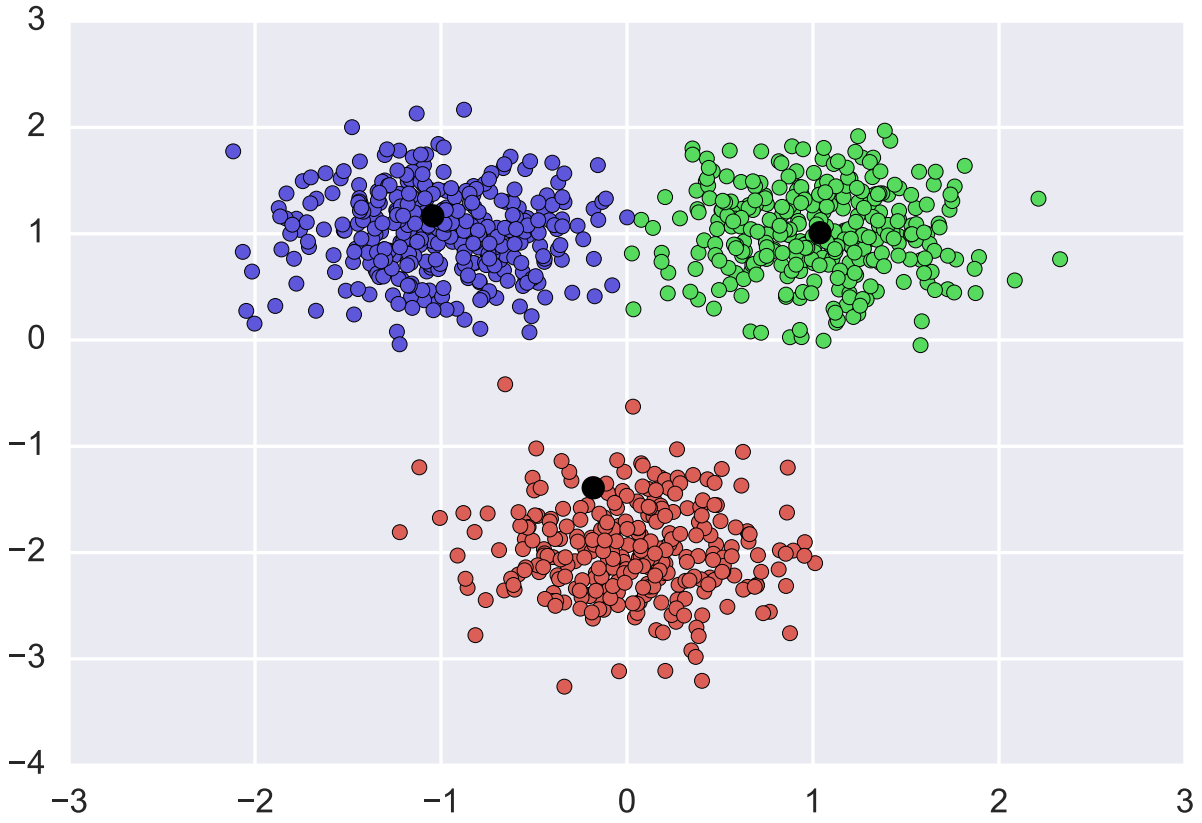
For better implementation, want to check for convergence as well as max number of iterations

```
def kmeans(X, k, max_iter=10):
    Mu = X[np.random.choice(X.shape[0],k),:]
    for i in range(max_iter):
        D = -2*X@Mu.T + (X**2).sum(axis=1)[:,None] + (Mu**2).sum(axis=1)
        C = np.eye(k)[np.argmin(D,axis=1),:]
        Mu = (C.T @ X)/np.sum(C,axis=0)[:,None]
    loss = np.linalg.norm(X - Mu[np.argmin(D,axis=1),:])**2
    return Mu, C, loss
```

Convergence of k-means



Convergence of k-means



Convergence of k-means

Possibility of local optima

Since the k-means objective function has local optima, there is the chance that we convert to a less-than-ideal local optima

Especially for large/high-dimensional datasets, this is not hypothetical: k-means will usually converge to a different local optima depending on its starting point

Convergence of k-means (bad)

Convergence of k-means (bad)

Convergence of k-means (bad)

Addressing poor clusters

Many approaches to address potential poor clustering: e.g. randomly initialize many times, take clustering with lowest loss

A common heuristic, k-means++: when initializing means, don't select $\mu^{(i)}$ randomly from all clusters, instead choose $\mu^{(i)}$ sequentially, sampled with probability proportion to the minimum squared distance to all other centroids

After these centers are initialized, run k-means as normal

K-means++

Given: Data set $\left(x^{(i)}\right)_{i=1,\ldots,m}$, # clusters k

Initialize:

$$\mu^{(1)} \leftarrow \mathrm{Random}\left(x^{(1:m)}\right)$$

For $j = 2, \ldots, k$:

Select new cluster:

$$\mu^{(j)} \leftarrow \mathrm{Random}\left(x^{(1:m)}, p^{(1:m)}\right)$$

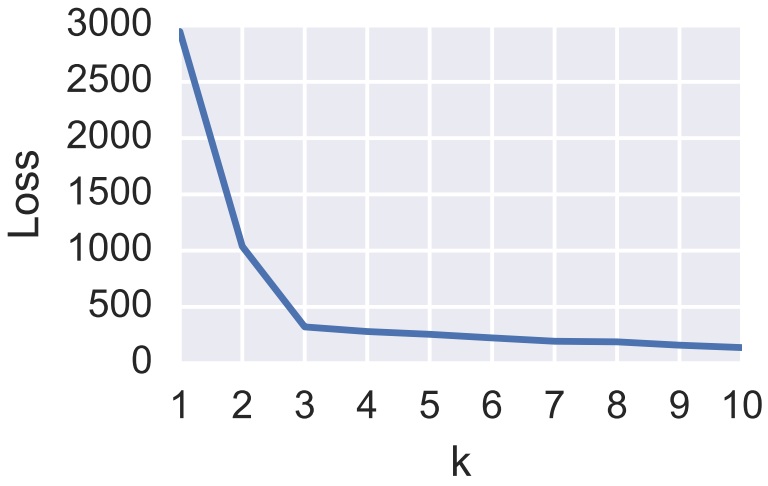
where probabilities $p^{(i)}$ given by

$$p^{(i)} \propto \min_{j'<j} \left\| \mu^{(j')} - x^{(i)} \right\|_2^2$$

How to select k?

There's no "right" way to select k (number of clusters): larger k virtually always will have lower loss than smaller k, even on a hold out set

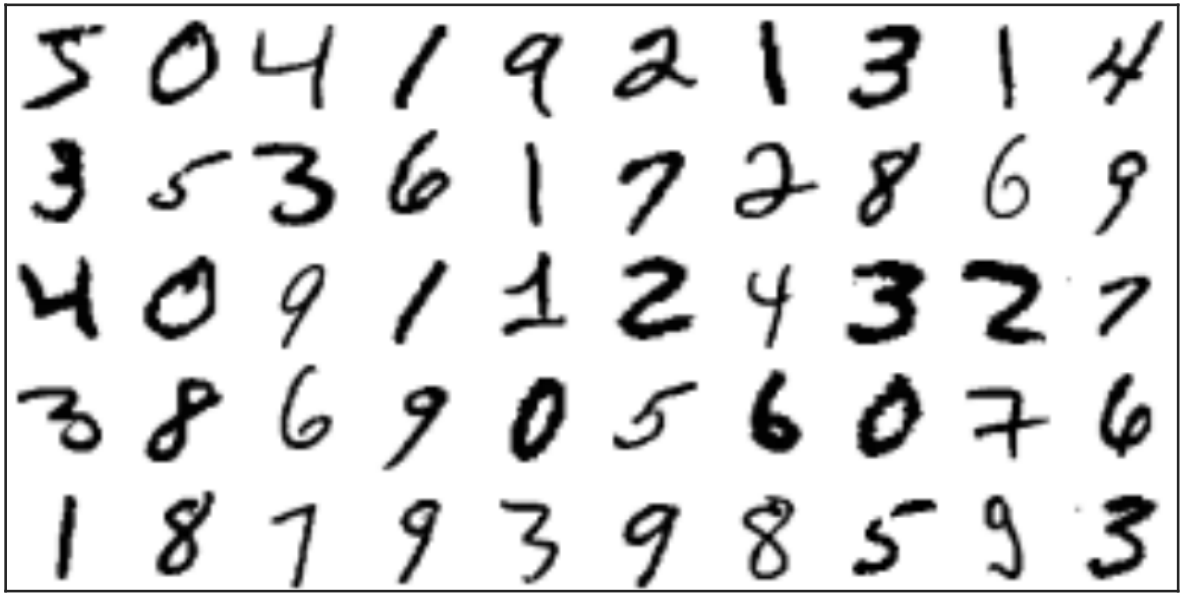
Instead, it's common to look at the loss function as a function of increasing k, and stop when things look "good" (lots of other heuristics, but they don't convincingly outperform this)



Example on real data

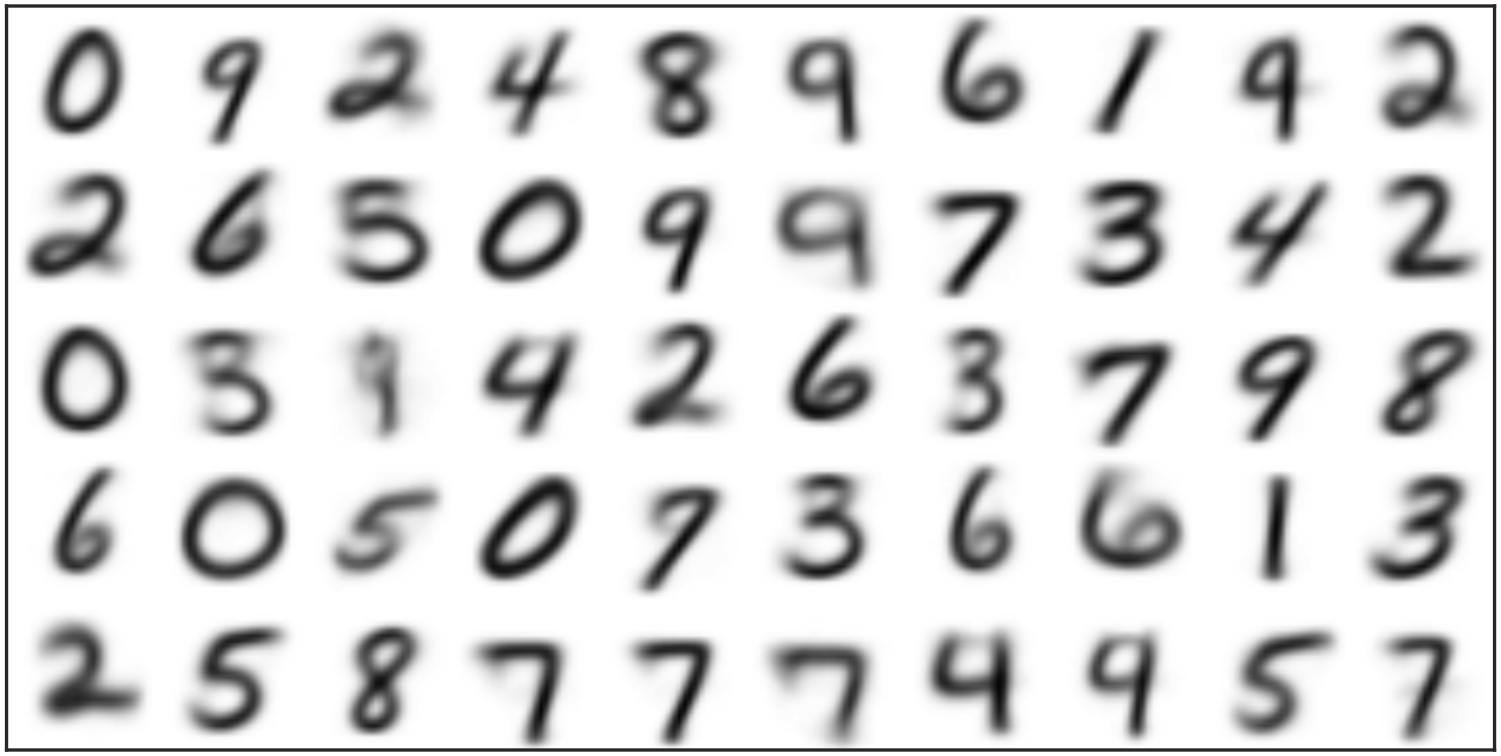
MNIST digit classification data set (used in question for 688 HW4)

60,000 images of digits, each 28x28



K-means run on MNIST

Means for k-means run with k=50 on MNIST data



Outline

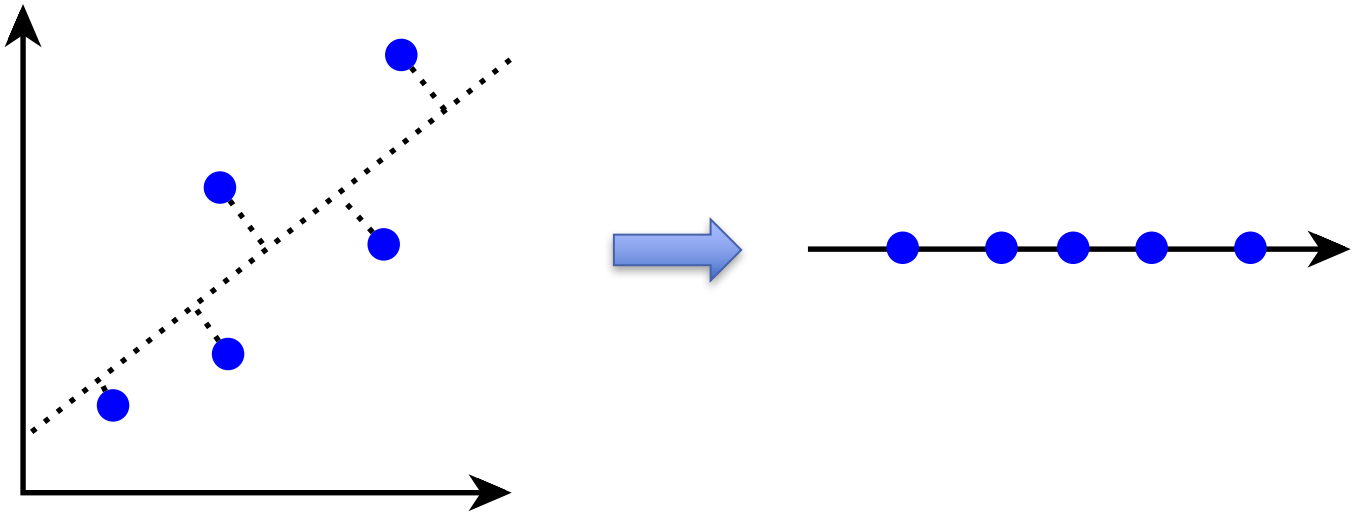
Unsupervised learning

K-means

Pricinple Component Analysis

Principal component analysis graphically

Principal component analysis (PCA) looks at “simplifying” the data in another manner, by preserving the axes of major variation in the data



PCA in unsupervised setting

We'll assume our data is normalized (each feature has zero mean, unit variance, otherwise normalize it)

Hypothesis function:

$$h_\theta(x) = UWx, \theta = \{U \in \mathbb{R}^{n \times k}, W \in \mathbb{R}^{k \times n}\}$$

i.e., we are "compressing" input by multiplying by a *low rank* matrix

Loss function: same as for k-means, squared distance

$$\ell(h_\theta(x), x) = \|h_\theta(x) - x\|_2^2$$

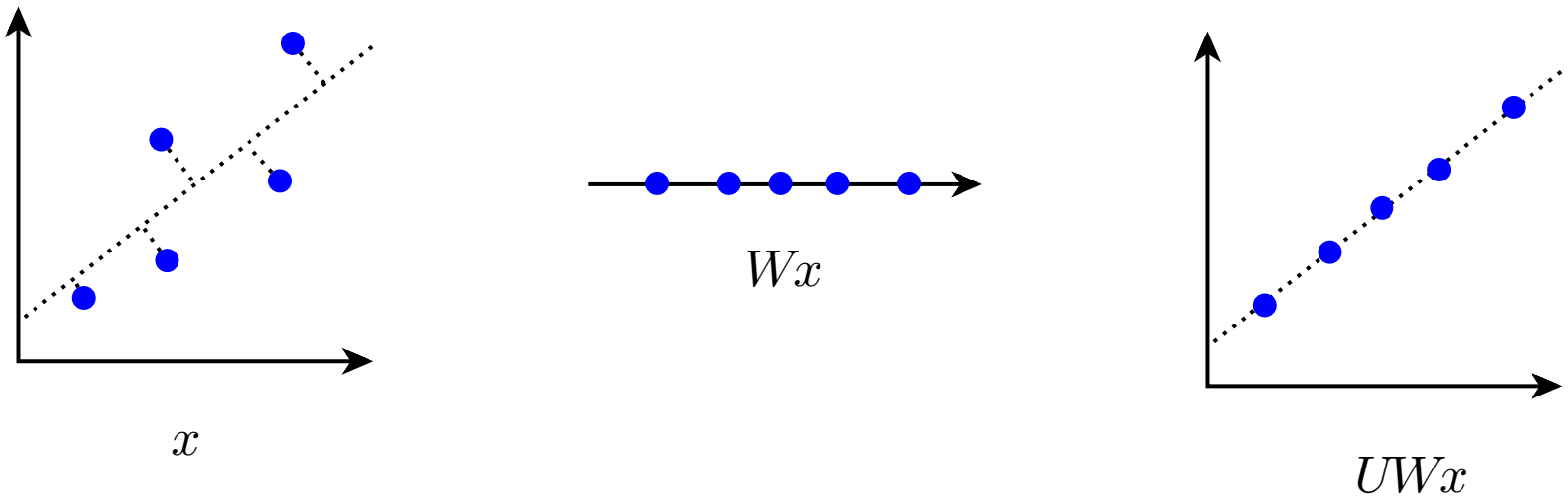
Optimization problem:

$$\underset{U,W}{\text{minimize}} \sum_{i=1}^{m} \left\|UWx^{(i)} - x^{(i)}\right\|_2^2$$

Dimensionality reduction with PCA

One of the standard uses for PCA is to reduce the dimension of the input data (indeed, we motivated it this way)

If $h_\theta(x) = UWx$, then $Wx \in \mathbb{R}^k$ is a "reduced" representation of x



Solving PCA optimization problem

The PCA optimization problem is also not convex, subject to local optima if we use e.g. gradient descent

However, amazingly, we *can* solve this problem exactly using what is called a singular value decomposition (all stated without proof)

Given: normalized data matrix X, # of components k

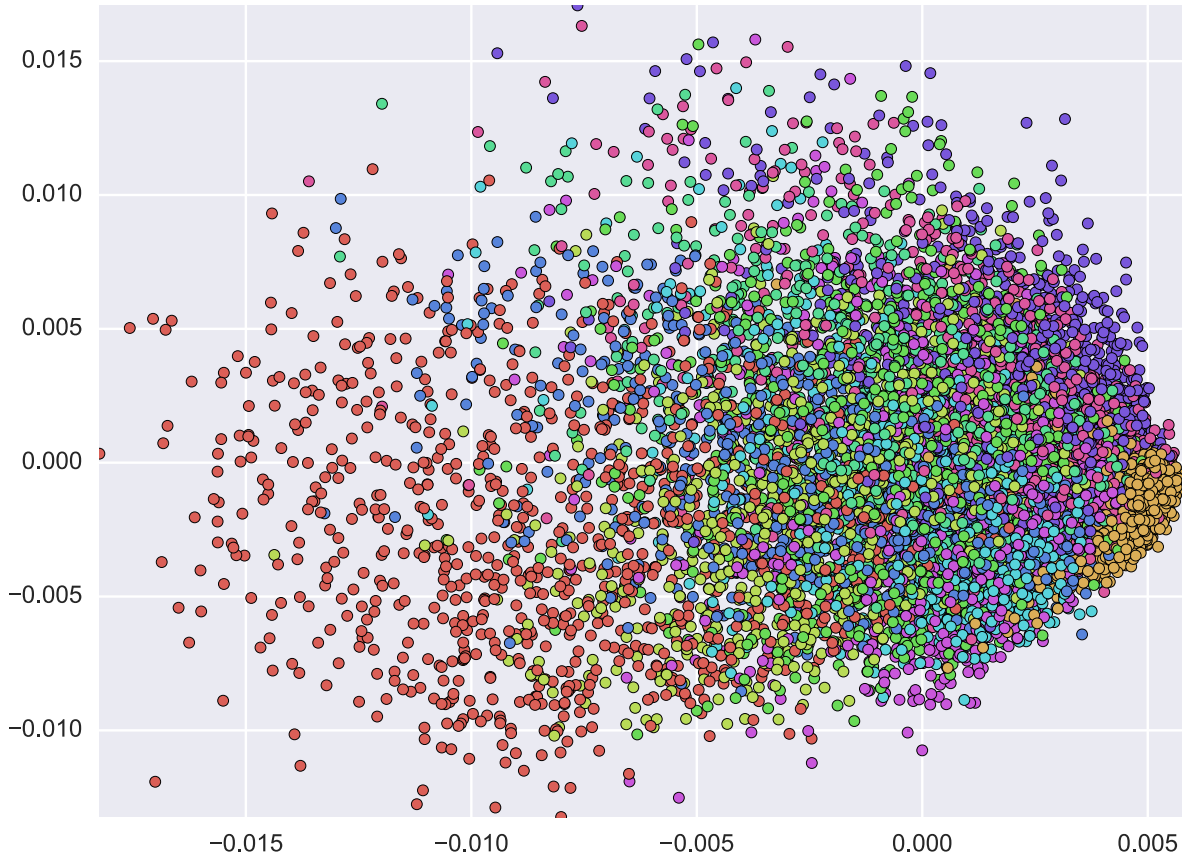
1. Compute singular value decomposition $USV^T = X$ where U, V is orthogonal and S is diagonal matrix of singular values
2. Return $U = V_{:,1:k} S^{-1}_{1:k,1:k}$, $W = V^T_{:,1:k}$
3. Loss given by $\sum_{i=k+1}^{n} S^2_{ii}$

Code to solve PCA

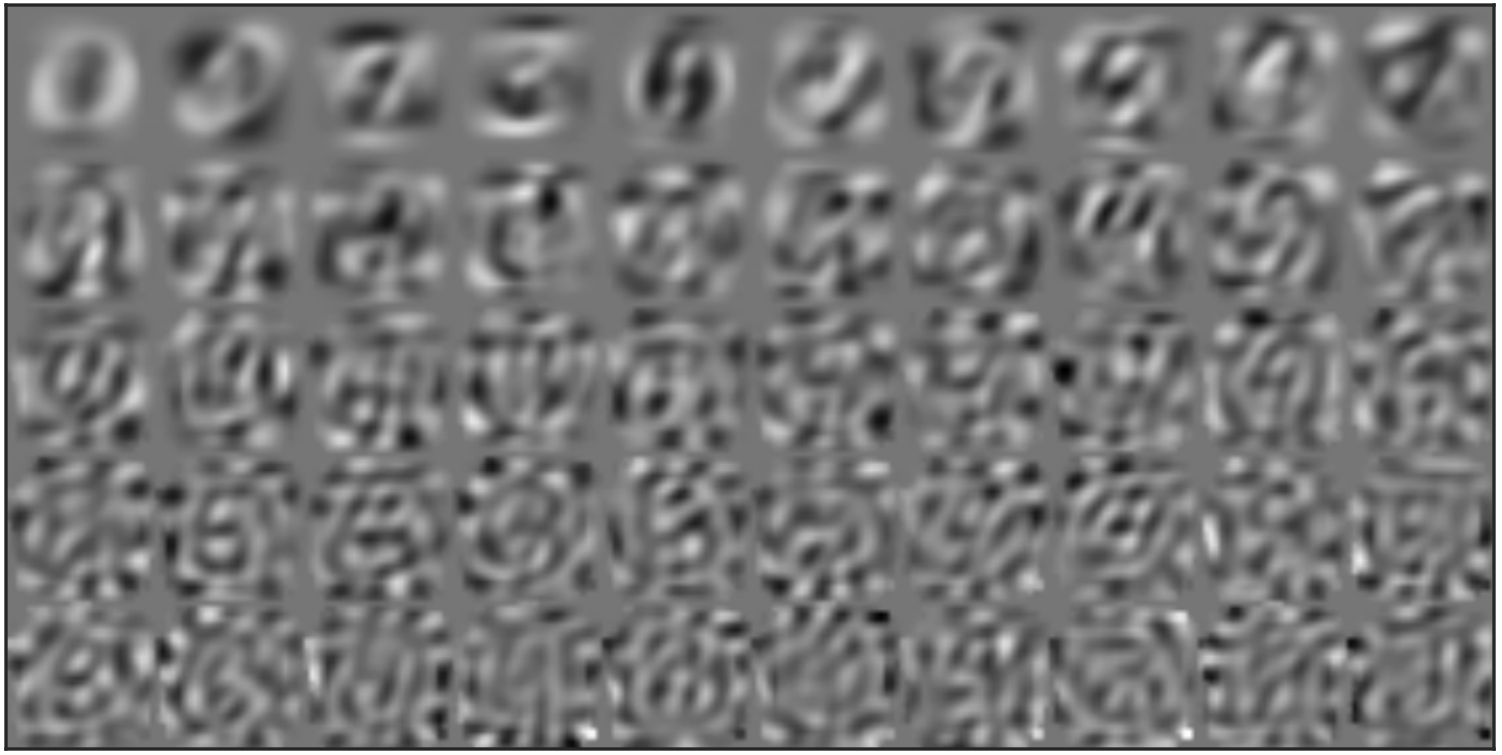
PCA is just a few lines of code, but all the actual interesting elements are the SVD call, if you're not familiar with this (which is fine), then there won't be too much insight

```
def pca(X,k):
    X0 = (X - np.mean(X, axis=0)) / np.std(X,axis=0) + 1e-8)
    U,s,VT = np.linalg.svd(X0, compute_uv=True, full_matrices=False)
    loss = np.sum(s[k:]**2)
    return VT.T[:,:k]/s[:k], VT.T[:,:k], loss
```

Dimensionality reduction on MNIST

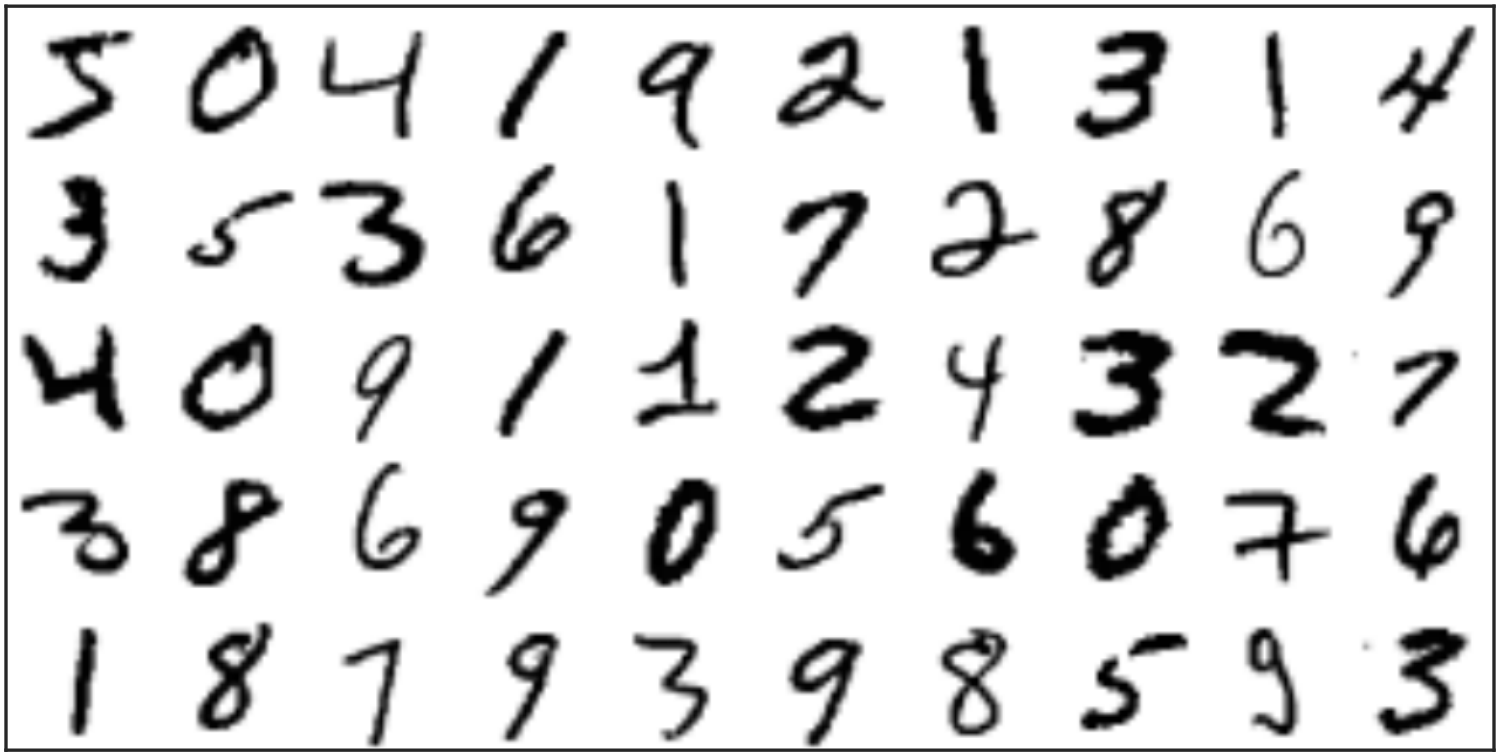


Top 50 principal components for MNIST



Images are reconstructed as linear combination of principal components

Reconstructed images, original



Reconstructed images, k=2

Reconstructed images, k=10

Reconstructed images, k=100

K-means and PCA in data preparation

Although useful in their own right as unsupervised algorithms, K-means and PCA are also useful in data preparation for supervised learning

Dimensionality reduction with PCA:

Run PCA, get W matrix

Transform inputs to be $\tilde{x}^{(i)} = Wx^{(i)}$

Radial basis functions with k-means

Run k-means to extract k centers, $\mu^{(1)}, \ldots, \mu^{(k)}$

Create radial basis function features $\phi_j^{(i)} = \exp\left(-\frac{\|x^{(i)} - \mu^{(j)}\|_2^2}{2\sigma^2}\right)$