Dimensionality Reduction: PCA and low-D embeddings

McInnes et al. (UMAP, 2000): Visualization of 30,000,000 integers as represented by binary vectors of prime divisibility, colored by integer value of the point

Go to tinyurl.com/cs441chat to ask questions during lecture (CampusWire in-lecture chat room)
Last Class - KNN

• K-nearest neighbor classification and regression

• Measuring and understanding error
Which of these tend to be decreased as the number of training examples increases?

a) Training error  
b) Test error  
c) Generalization error
This class – dimensionality reduction

• Goal: We want to represent high dimensional data with fewer dimensions, e.g. for:
  – Compression: reduced storage, faster retrieval
  – Visualization: plot in two dimensions

• A good dimensionality reduction can be defined in different ways
  – Be able to reproduce the original data
  – Preserve discriminative features
  – Preserve the neighborhood structure

• One main strategy is linear projection, e.g. a street map projects everything onto the 2D ground dimensions and ignores height

• Another strategy is embedding or manifold fitting, where we try to preserve relationships in the data
This class – PCA and manifolds

• Linear projection
  – PCA: Principal Components Analysis
  – Reduce dimension while preserving variance of data

• Embedding/manifold learning
  – MDS (multidimensional scaling), IsoMap and t-SNE
  – Preserve point distances and/or local structure
Key terms

• Vectors and matrices
• Translation
• Projection
• Scaling
• Rotation
• Rank
• Eigenvectors/eigenvalues
• SVD
Key terms

Vector can represent a data point $\mathbf{x}$ or a projection $\mathbf{w}$ onto a coordinate

- $\mathbf{w}^T \mathbf{x}$ projects data point $\mathbf{x}$ onto the axis defined by $\mathbf{w}$
- E.g. suppose $d = 2$
  - $\mathbf{w} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ selects the first value of $\mathbf{x}$
  - $\mathbf{w} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ adds the two values of $\mathbf{x}$ together

Matrix can represent a set of data points or a set of projection vectors
Key terms

**Translation** is a transformation that adds a constant value to each coordinate
• E.g. centering is $x_c = x - \mu_x$, where $\mu_x$ is the mean of $x$

**Scaling** is a transformation that multiplies each coordinate by a constant value
• E.g. $W = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$ will double the value of the first coordinate of $x$ when applied as $Wx$ (blanks are assumed to be zero)
Key terms

**Rotation** is a set of projections that preserves the distances between points and distances to the origin

- Each row and column represents a **basis vector**
- The basic vectors must have a unit norm and be orthogonal to each other: $r_i^T r_i = 1, r_i^T r_j = 0, \forall (i, j \neq i)$
Key terms

**Rank** of matrix $M$ is the number of linearly independent vectors in the rows or columns of $M$

- $\text{Rank}(M)$ is at most the smaller of the number of rows and columns in $M$

- $\text{Rank}\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = 2$

- $\text{Rank}\begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix} = 1$ because one of the rows (or columns) can be composed of a weighted sum of the others
Key terms

**Eigenvector** $\mathbf{v}$ and corresponding **eigenvalue** $\lambda$ of matrix $\mathbf{M}$ are defined by having the special property $\mathbf{Mv} = \lambda \mathbf{v}$

- Eigenvectors characterize matrices, and appear as part of a solution to many linear algebra problems

**SVD** (singular value decomposition) is a factorization of a matrix $\mathbf{A}$ into $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, where:
- Columns of $\mathbf{U}$ are eigenvectors of $\mathbf{AA}^T$
- Columns of $\mathbf{V}$ are eigenvectors of $\mathbf{A}^T\mathbf{A}$
- $\mathbf{\Sigma}$ is a diagonal (scaling) matrix of singular values, which are square roots of the eigenvalues of both $\mathbf{AA}^T$ and $\mathbf{A}^T\mathbf{A}$
PCA

- General dimensionality reduction technique
  - Finds major orthogonal directions of variation

- Preserves most of variance with a much more compact representation
  - Lower storage requirements
  - Faster matching/retrieval
  - Easier to work in low dimensions, e.g. for probability estimation
Center and rotate the axes

Original 2D Representation

(3,4)
(0.5,-2)
(-1,-3)

New 2D Representation

(4.2,0.04)
(-2.1,0.1)
(-4.1,-0.03)
Now data is well-approximated by dropping the coordinates with the least variance

New 2D Representation

\[ (4.2, 0.04) \]

\[ (-2.1, 0.1) \]

\[ (-4.1, -0.03) \]

New 1D Representation

\[ (4.2, 0) \]

\[ (-2.1, 0) \]

\[ (-4.1, 0) \]
Principal Component Analysis

- Given a point set $\{\mathbf{r}_j\}_{j=1}^P$, in an $M$-dim space, PCA finds a basis such that
  - The most variation is in the first basis vector
  - The second most, in the second vector that is orthogonal to the first vector
  - The third...
Principal Component Analysis (PCA)

• Given: N data points \( x_1, \ldots, x_N \) in \( \mathbb{R}^d \)

• We want to find a new set of features that are linear combinations of original ones:

\[
u(x_i) = u^T(x_i - \mu)
\]

(\( \mu \): mean of data points)

• Choose unit vector \( u \) in \( \mathbb{R}^d \) that captures the most data variance
Principal Component Analysis

Direction that maximizes the variance of the projected data:

Maximize \[
\frac{1}{N} \sum_{i=1}^{N} u^T (x_i - \mu)(u^T (x_i - \mu))^T
\]
subject to \( ||u||=1 \)

Projection of data point

\[
= u^T \left[ \frac{1}{N} \sum_{i=1}^{N} (x_i - \mu)(x_i - \mu)^T \right] u
\]

Covariance matrix of data

\[
= u^T \Sigma u
\]

The direction that maximizes the variance is the eigenvector associated with the largest eigenvalue of \( \Sigma \) (can be derived using Raleigh’s quotient or Lagrange multiplier)
PCA in Python

```python
print(X.shape)
x_mu = np.mean(X, axis=0)
X_cov = np.matmul((X-x_mu).transpose(), X-x_mu)/X.shape[0]
[lam, v] = np.linalg.eig(X_cov)
print(lam[:3])
v = v.transpose()
from sklearn.decomposition import PCA
pca_transform = PCA().fit(X)
print(np.max(np.abs(v[0]-pca_transform.components_[0])))
print(pca_transform.explained_variance_[3])
```

Printout shows that eigenvalues are the same as the explained variance, and the first component is identical (up to numerical precision)
Principal Component Analysis

First $r < M$ principal component vectors provide an approximate basis that minimizes the mean-squared-error (MSE) of reconstructing the original points.

Choosing subspace dimension $r^*$:
- Look at decay of the eigenvalues as a function of $r$.
- Larger $r$ means lower expected error in the subspace data approximation.
Example on aligned faces

$x_1, ..., x_N$ are pixel values of each face
PCA of aligned face images (called “eigenfaces”)

Top eigenvectors: $u_1, \ldots, u_k$

Mean: $\mu$
Visualization of eigenfaces (appearance variation)

Principal component (eigenvector) $u_k$

$\mu + 3\sigma_k u_k$

$\mu - 3\sigma_k u_k$
Representation and reconstruction

• Face \( x \) in “face space” coordinates:

\[
x \rightarrow [u_1^T (x - \mu), \ldots, u_k^T (x - \mu)] = w_1, \ldots, w_k
\]
Representation and reconstruction

- Face $\mathbf{x}$ in “face space” coordinates:

$$
\mathbf{x} \rightarrow [\mathbf{u}_1^T (\mathbf{x} - \mu), \ldots, \mathbf{u}_k^T (\mathbf{x} - \mu)] = w_1, \ldots, w_k
$$

- Reconstruction:

$$
\hat{\mathbf{x}} = \mu + w_1 \mathbf{u}_1 + w_2 \mathbf{u}_2 + w_3 \mathbf{u}_3 + w_4 \mathbf{u}_4 + \ldots
$$
After computing eigenfaces using 400 face images from ORL face database
Preserving variance (minimizing MSE) does not necessarily lead to perceptually good reconstruction.

$P = 200$

Plot of eigenvalues for each eigenvector of the covariance matrix, equivalent to the variance contained along each principal component.
Another example, representing MNIST 4’s

Variance per component

Cumulative % variance explained

Reconstructions with varying # PCs

```
from sklearn.decomposition import PCA
x4 = x_train[y_train==4]
pca = PCA().fit(x4)
pca_x4 = pca.transform(x4)
plt.plot(pca.explained_variance_)
plt.show()
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.show()
ims = np.zeros((10, 28*28))
nc = np.int32([0, 1, 5, 10, 20, 50, 100, 200, 400, 768])
for i in range(len(nc)):
    c = nc[i]
    ims[i] = pca.mean_ + np.matmul(pca_x4[0][:,c], pca.components_[:,c])
display_mnist(ims, 1, 10)
```
Two minute break (3 questions)

For each plot of data, what is the direction of the first principal component?

If X consists of N data points of d dimensions, what is the maximum number of PCA components that would be needed to perfectly reconstruct the data?

\[ \text{Rank}(X) \leq \min(N, d) \]
PCA: MNIST at 2 dimensions

Note: I'm only plotting a subset
Non-Linear Scaling and Manifold Estimation

We may care less about being able to reconstruct each data point than representing the relationships between data points

• MDS: Preserve Euclidean pairwise distances

• Non-metric MDS: Preserve distance orderings

• ISOMAP: Define distances in terms of “geodesic” (graph-based) similarity
MultiDimensional Scaling (MDS)

- For all data points, solve for new coordinate positions that preserve some input set of pairwise distances
- Classic case (equations on right) uses Euclidean distance and has closed form solution
- More generally, distance can be defined arbitrarily (e.g. from user surveys)
- Major drawback is that the solution is most influenced by points that are far from each other

\[ \sum_{ij} \left( D_{ij}^{(2)}(x) - D_{ij}^{(2)}(y) \right)^2 \]

where

\[ D_{ij}^{(2)}(x) = (x_i - x_j)^T (x_i - x_j) \]

\( x \) are old coordinates, \( y \) are new coordinates

See Forsyth AML 6.2 for details
MDS on MNIST

- 30 PCA components
- MDS to 2 dimensions

Note: For MDS and others, manifolds are fit on 500 pts for speed.
MDS on MNIST

Pre-process with PCA to 30 dim

No PCA
Non-metric MDS

• Optimize position of data points so that Euclidean distance preserves the ordering of input pairwise distances
• Requires only an order of dissimilarities

• Slow because this is a complicated optimization
ISOMAP

- Same as MDS but define distance in a graph
  - Compute adjacency graph (e.g. 5 nearest neighbors)
  - Distance is shortest path in graph between two points
t-SNE

Map to 2 or 3 dimensions while preserving similarities of nearby points

1. Assign probability $p_{ij}$ that pairs of points are similar, e.g. with Gaussian weighted distance

2. Define similarity $q_{ij}$ in new coordinates

3. Minimize KL divergence between $p$ and $q$ (i.e. they should have similar distributions) using gradient descent

For $i \neq j$, define

$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2/2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|x_i - x_k\|^2/2\sigma_i^2)}$$

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

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_k \sum_{l \neq k} (1 + \|y_k - y_l\|^2)^{-1}}$$

$$\text{KL} (P \parallel Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$
t-SNE

- In high dimensions, each point tends to be similarly distant to many points, so we often use PCA before applying t-SNE.
Comparison

Comparison on MNIST

- **PCA**
- **MDS**
- **IsoMap**
- **tSNE**
UMAP (McInnes, 2020)

• Assumes data is uniformly distributed on an underlying manifold that is locally connected. Goal is to preserve that local structure.

• Algorithm: relatively complicated, incorporates many ideas from other methods, has strong mathematical foundations

• Hyperparameters:
  – number of neighbors to consider
  – dimension of target embedding
  – desired separation between close points
  – number of training epochs

UMAP on MNIST

5000 points, 100 neighbors, 34s

500 pts, 10 neighbors

```python
# !pip install umap-learn
from umap import UMAP
ind = train_indices['s']
x_umap = UMAP(n_components=2, n_neighbors=100).fit_transform(x[ind,:30])
sns.scatterplot(x=x_umap[ind,0], y=x_umap[ind,1], hue=y_train[ind], palette="colorblind")
```
UMAP on MNIST

50,000 points, 100 neighbors, 200s
Things to remember

• PCA reduces dimensions by linear projection
  – Preserves variance to reproduce data as well as possible, according to mean squared error
  – May not preserve local connectivity structure or discriminative information

• Other methods try to preserve relationships between points
  – MDS: preserve pairwise distances
  – IsoMap: MDS but using a graph-based distance
  – t-SNE: preserve a probabilistic distribution of neighbors for each point (also focusing on closest points)
  – UMAP: incorporates k-nn structure, spectral embedding, and more to achieve good embeddings relatively quickly
Next class: Linear Regression and Regularization