Lab 8: Data Decompositions and Clustering

University of Washington

EE 596/AMATH 563

Spring 2021

Outline

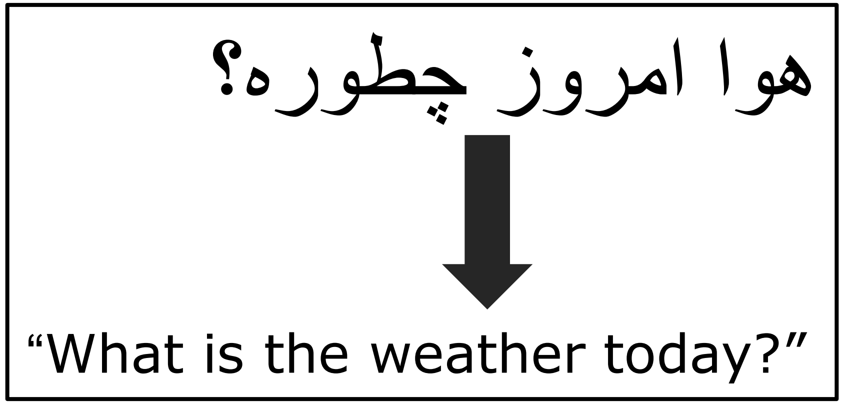
- Sequential Data Review
- Classical Data Decompositions
- Clustering Methods
- Example: Customer Clustering
- Assignment: Credit Card User Clustering and Classification

Sequential Data Review

Example Sequential Data and Tasks

- Written Language
  - Character Prediction
  - Machine Translation
- Audio
  - Speech Processing
  - Music Transcription
- Spatio-Temporal Data
  - Body capture data
  - Weather modelling
  - Neurological Models

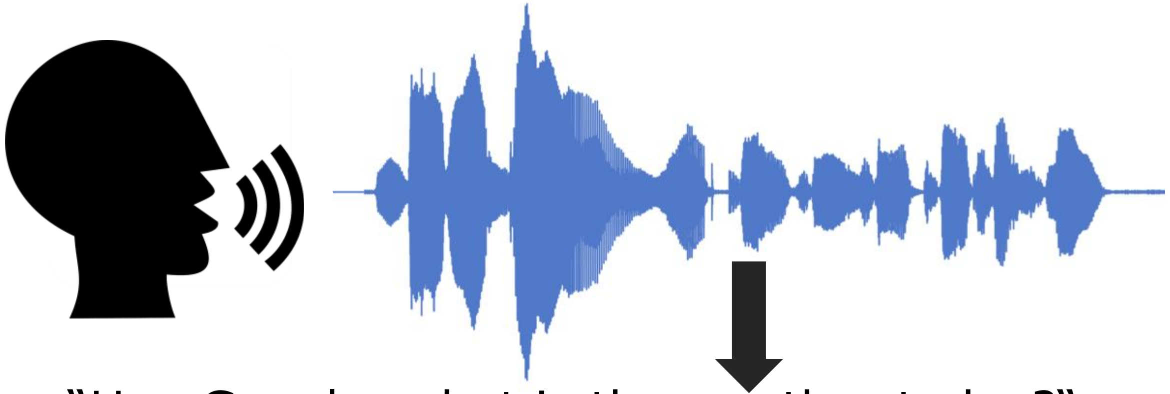
Language Modelling/Prediction



Machine Translation

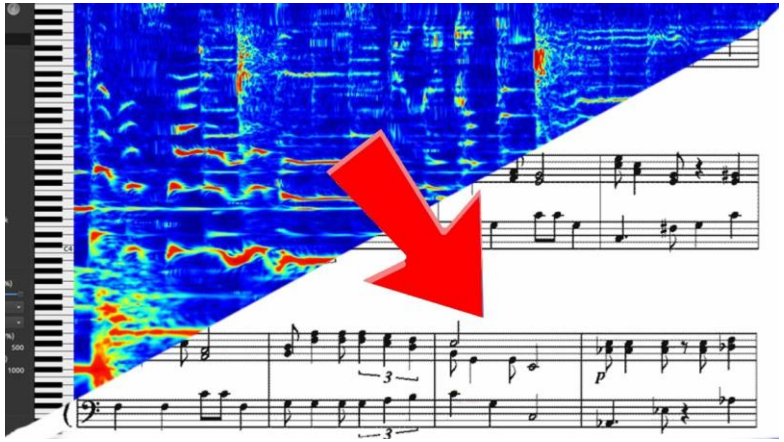
Example Sequential Data and Tasks

- Written Language
  - Character Prediction
  - Machine Translation
- Audio
  - Speech Recognition
  - Music Transcription
- Spatio-Temporal Data
  - Body capture data
  - Weather modelling
  - Neurological Models



“Hey Google, what is the weather today?“

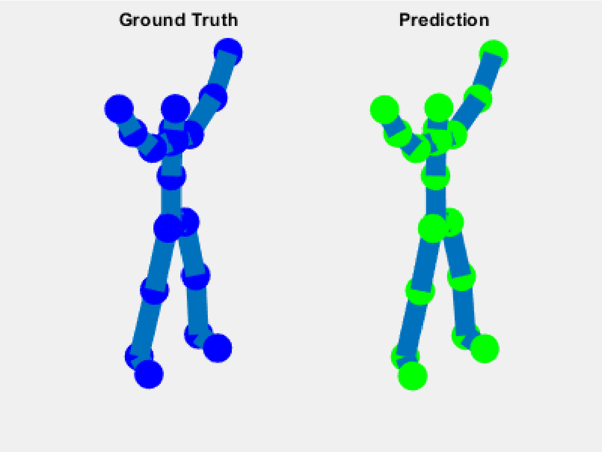
Speech Recognition



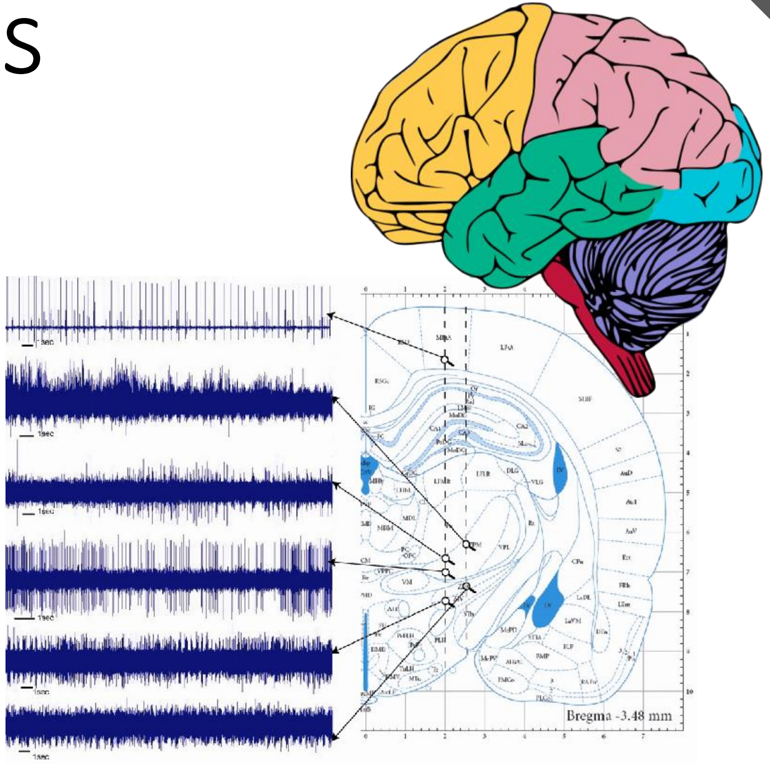
Music Transcription

Example Sequential Data and Tasks

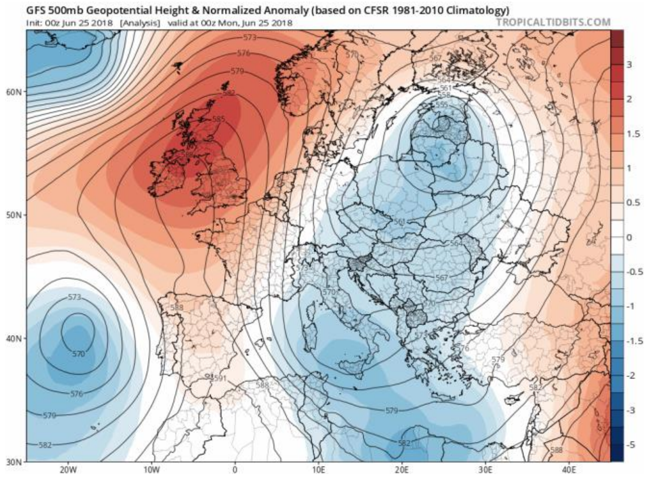
- Written Language
  - Character Prediction
  - Machine Translation
- Audio
  - Speech Recognition
  - Music Transcription
- Spatio-Temporal Data
  - Body capture data
  - Weather modelling
  - Neurological Data



Body Motion Capture



Brain Activity



Weather Modelling

Features of Sequential Data

- Order matters
  - Unlike a group of vectors, the order in which data appear in sequential data is important
- Variable length
  - Measurements are not always captured over the same number of time-steps
- Temporal dependence
  - Previous data values usually has impact on current value

Classical Data Decompositions

Proper Orthogonal Decomposition (POD)

- Measure a system at T discrete time steps and N finite spatial locations
- Approximate a high-rank matrix **X** by a *K*-rank matrix $\Phi \cdot A^T$

$$\mathbf{X} = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & x_1^{(3)} & \cdots & x_1^{(T)} \\ x_2^{(1)} & x_2^{(2)} & x_2^{(3)} & \cdots & x_2^{(T)} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_N^{(1)} & x_N^{(2)} & x_N^{(3)} & \cdots & x_N^{(T)} \end{bmatrix}$$

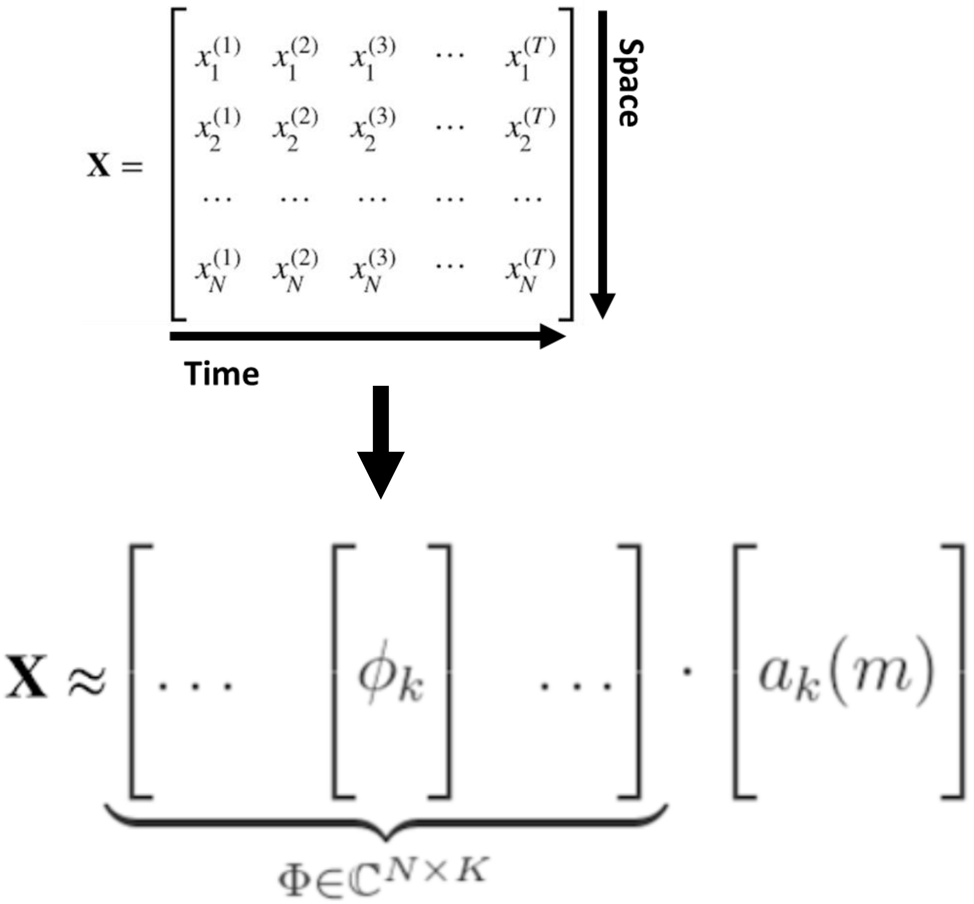
Space

Time

Proper Orthogonal Decomposition (POD)

- Goal: find $\Phi \cdot A^T$ of rank K which attains the shortest distance
$$||\boldsymbol{X} - \Phi \cdot A^T||$$
  - Looking for best orthogonal basis

- Three methods: SVD, PCA, KLD



Singular Value Decomposition (SVD)

- Extension of the eigenvalue decomposition for non-square, non-symmetric matrices

- Matrix $\mathbf{X}$ factorized into:
$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathbf{T}}$$

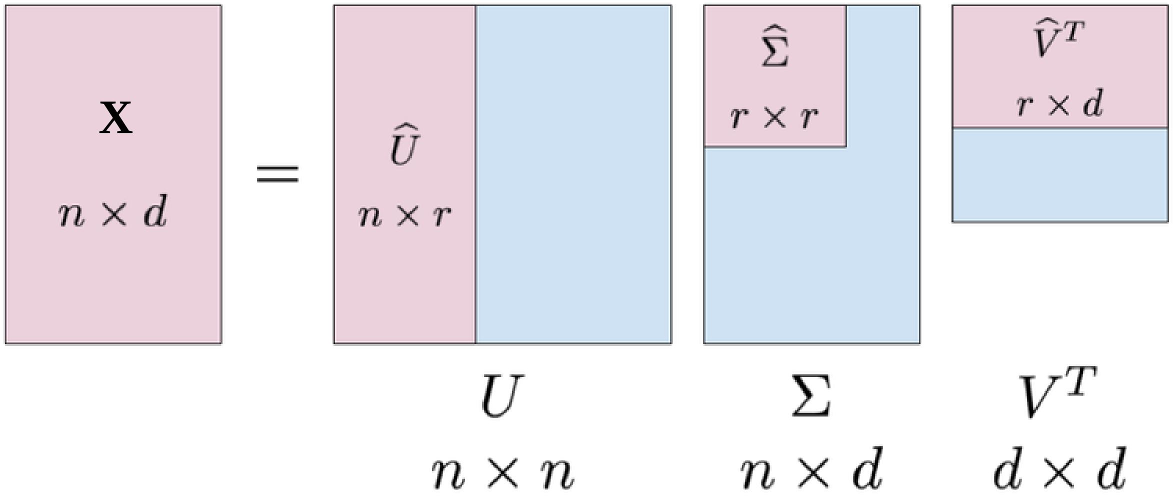


Image Source: https://intoli.com/blog/pca-and-svd/

Singular Value Decomposition (SVD)

- Matrix $\mathbf{X}$ factorized into:
$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathbf{T}}$$

- Diagonal entries of $\mathbf{\Sigma}$ are (ordered) singular values of $\mathbf{X}$. Columns of $\mathbf{U}$ and $\mathbf{V}$ are modes (singular vectors).
- Can reduce dimension by taking r-dimensional truncation of $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}$

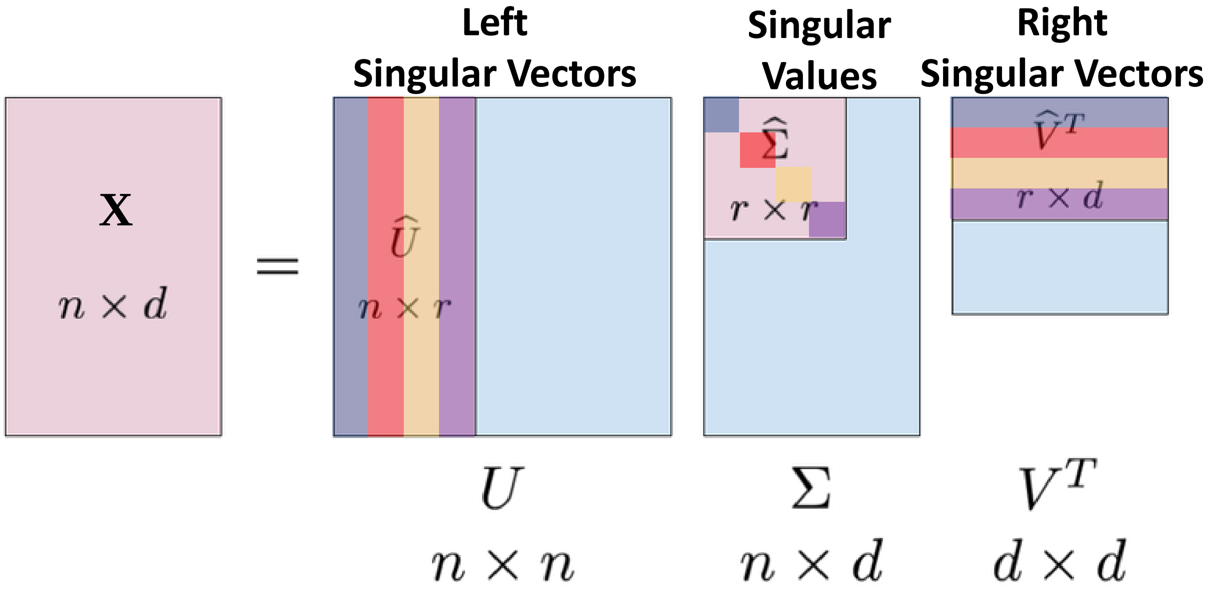


Image Source: https://intoli.com/blog/pca-and-svd/

Principal Component Analysis (PCA)

- Reduce dimensionality of data set while retaining as much variation in data as possible

- Principal Components (PCs) are the eigenvectors of covariance matrix $Cov(\mathbf{X})$
  - Project data onto this new basis

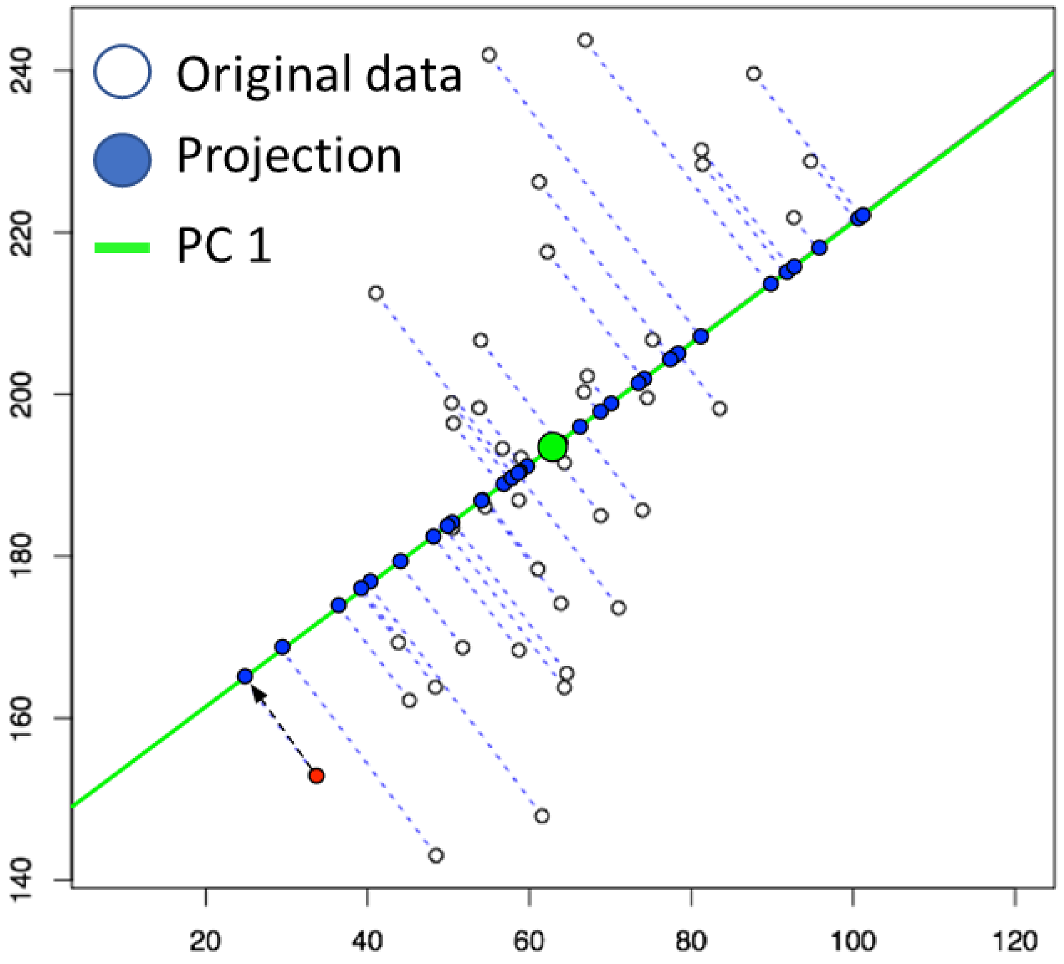


Image source: https://liorpachter.wordpress.com/2014/05/26/what-is-principal-component-analysis/

Principal Component Analysis (PCA)

- PCs are uncorrelated and ordered, so first few PCs capture much of the variation in data

- Equivalent to SVD up to a normalization factor and centering of data points

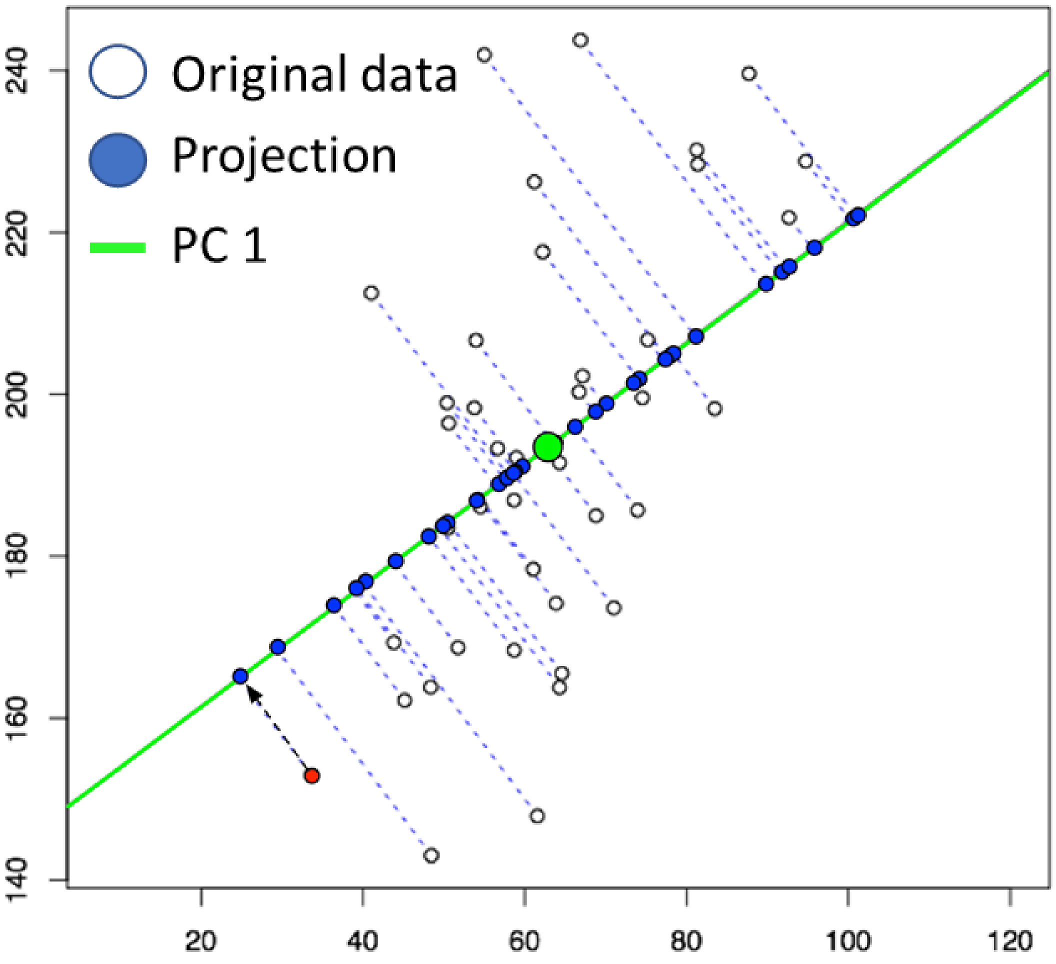


Image source: https://liorpachter.wordpress.com/2014/05/26/what-is-principal-component-analysis/

Karhunen–Loève Decomposition (KLD)

- Representation of stochastic process as a linear combination of orthogonal functions
- For discrete, finite process, this is equivalent to PCA

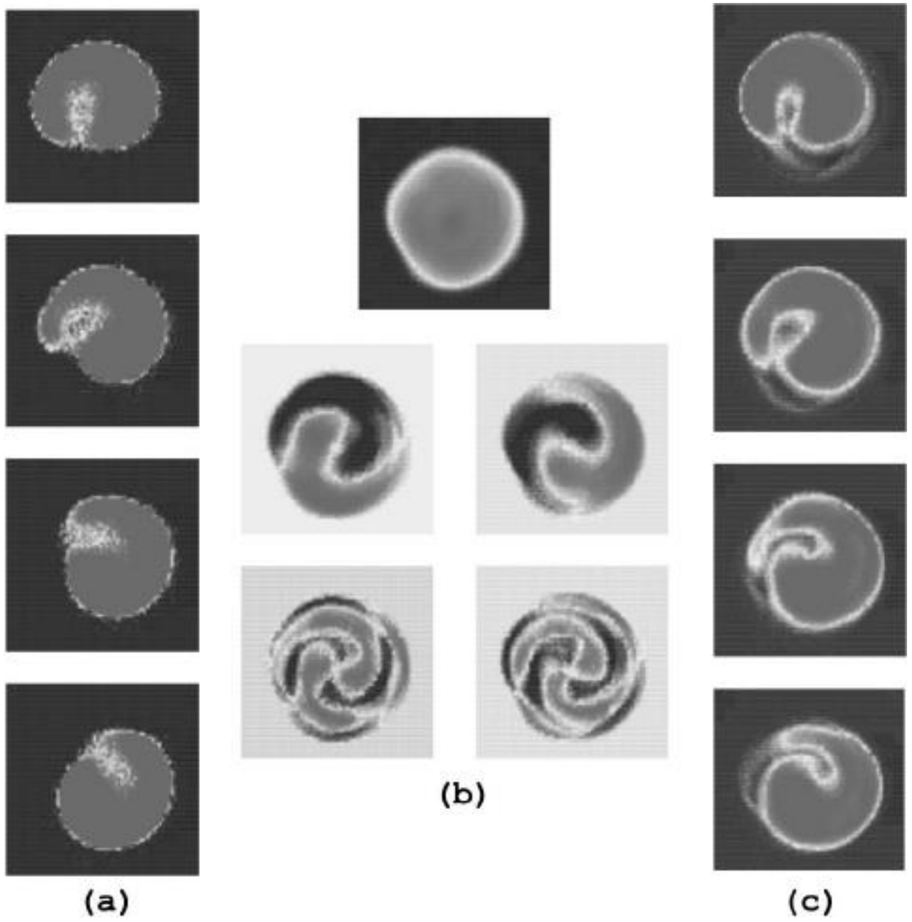


Image source: https://www.researchgate.net/figure/A-KL-decomposition-of-a-rotating-one-cell-state-from-the-experiment-a-four_fig3_235583042

SVD Implementation in PyTorch

```
torch.linalg.svd(input, full_matrices=True, compute_uv=True, *, out=None)
```

- **Output** is (U, S, V^T)
- **input** is the input Tensor
- **full_matrices** determines whether to output full-dimensional matrices or reduce dimension (for rectangular input)
- If **compute_uv** is False, *U* and *V* are empty tensors

```
#Calculate SVD
X = torch.rand(10, 20)
U, S, Vh = torch.linalg.svd(X)
V = Vh.T

#Low-dimensional Reconstruction
rd = 5 #Number of (reduced) dimensions to use
X_red = U[:, :rd]@(torch.diag(S)[:rd, :rd])@Vh[:rd, :]
```

PCA Implementation in PyTorch

```
torch.pca_lowrank(A, q=None, center=True, niter=2)
```

- **Output** is (U, S, V)
  - Note: V is not transposed as it is in SVD
- **A** is the input tensor
- **q** should be a slightly overestimated rank of *A*
- **center** (bool): If false, *A* should already be centered
- **niter** (int): number of iterations used for algorithm

```
#Calculate PCA
X = torch.rand(10, 20)
U, S, V = torch.pca_lowrank(X)

#Low-dimensional Reconstruction
rd = 5 #Number of (reduced) dimensions to use
X_red = U[:, :rd]@(torch.diag(S)[:rd, :rd])@V.T[:rd, :]
```

Dynamic Mode Decomposition (DMD)

- Method for approximating

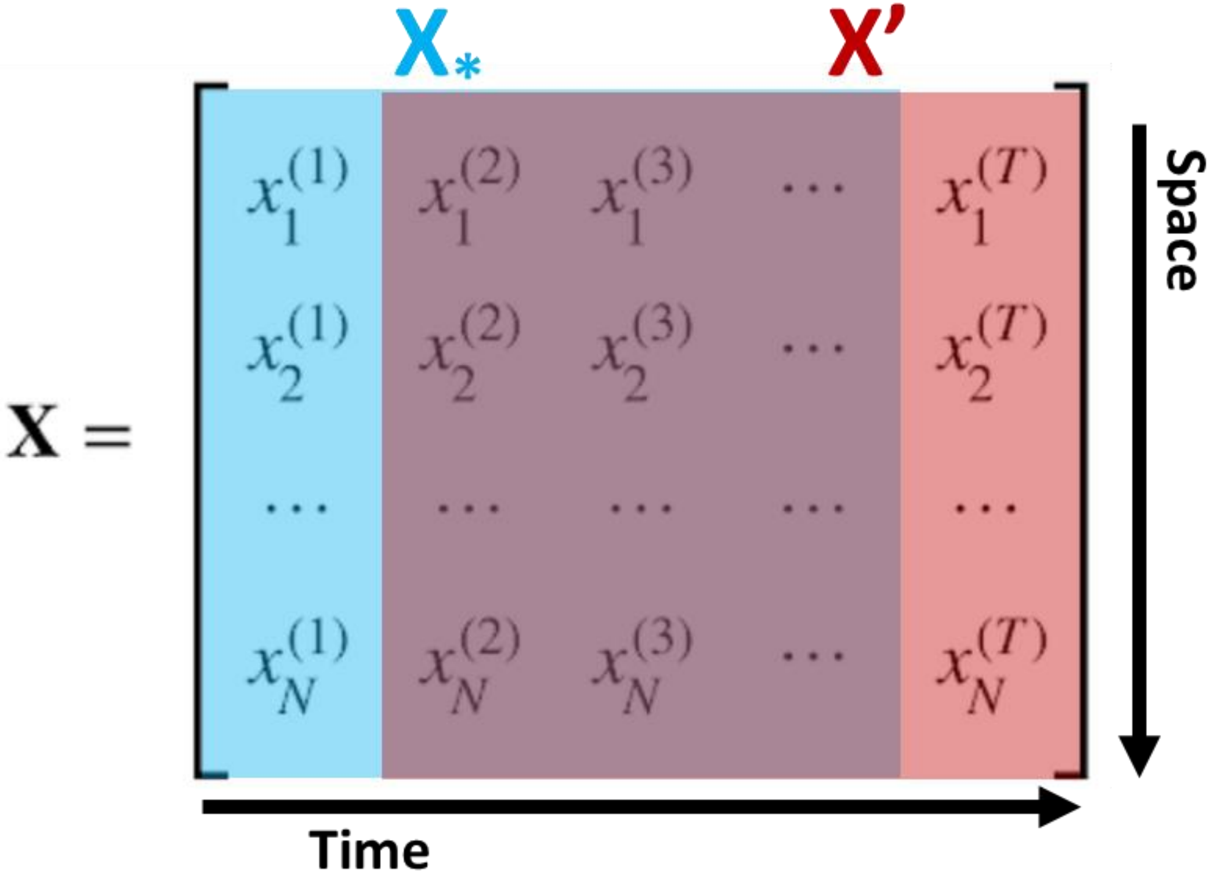
$$\frac{d\boldsymbol{x}}{dt} = f(x, t, \mu)$$

without knowing f

- Offsetting data by one time step allows approximation of derivative

$X_* = X[:-1]$

$X' = X[1:\ ]$



Dynamic Mode Decomposition (DMD)

- Relate X' to X_* by

$$\boldsymbol{X}' = A\boldsymbol{X}_*$$

- A represents a linearization of the function *f*

- Then, we can solve for

$$A = \boldsymbol{X}'\boldsymbol{X}_*^t$$

Where ^{t} indicates the pseudoinverse

DMD Algorithm

- Take SVD of X_* and reduce to order *r*
- Transform basis of A using reduced **U** from SVD, making $\tilde{A}$
- Find eigenvectors of $\tilde{A}$
- Project back into original state space to get DMD modes, $\boldsymbol{\Phi}$

```
#DMD Calculation
X = torch.rand(10, 20)
X_a = X[:-1]
X_b = X[1:]

#Take SVD
U, S, Vh = torch.linalg.svd(X_a)
#Reduce Order
r = 3
U_r = U[:, :r]
V_r = Vh[:r, :].T
S_i = torch.inverse(torch.diag(S[:r]))
#Transform A
A_tilde = U_r.T@X_a@V_r@S_i
#Eigendocomposition of A
lam, W = torch.eig(A_tilde, True)
#Transform back, get DMD Modes
Phi = X_b@V_r@S_i@W
```

Data Clustering Methods

K Means Clustering

- Method to cluster *n* data points into *k* separate groups- unsupervised
- *k* is defined ahead of time
- *k* points are randomly initialized as “means”
- Clusters are determined by closest mean
- Mean of each class is updated at each step until convergence

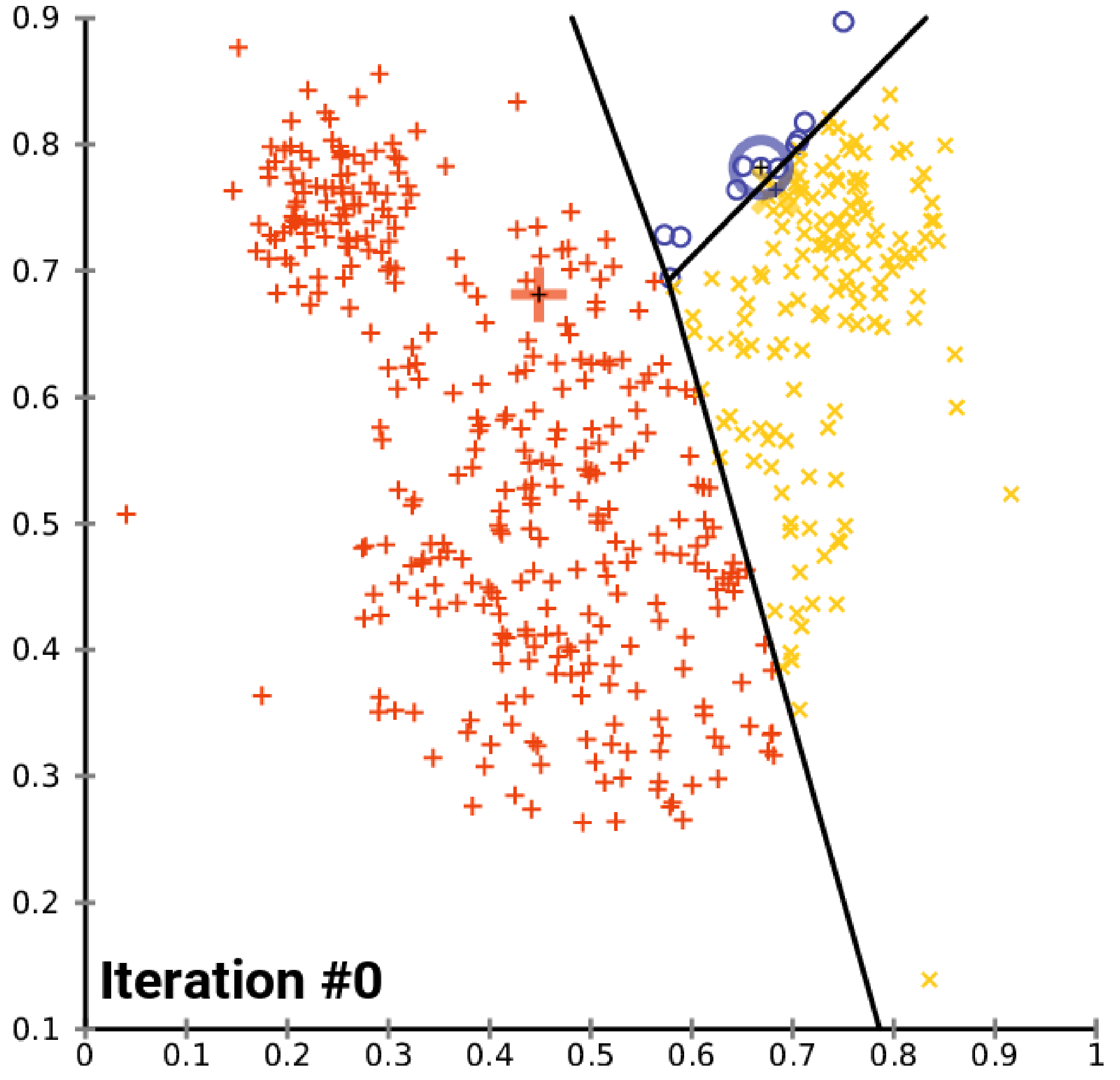


Image Source: https://commons.wikimedia.org/w/index.php?curid=59409335

K Means Implementation

- Start by defining the number of clusters, *K*
- Define number of iterations to update means
- Create Tensors to track means (centroids) for each cluster

```
x = torch.rand(50, 2)
# K means clustering
K = 3 #Define K ahead of time
Niter = 10 # Defne number of mean update iterations
N, D = x.shape  # Number of samples, dimension of the ambient space

c = x[:K, :].clone()  # Simplistic initialization for the centroids

x_i = torch.Tensor(x.view(N, 1, D))  # (N, 1, D) samples
c_j = torch.Tensor(c.view(1, K, D))  # (1, K, D) centroids

```

Code adapted from: https://www.kernel-operations.io/keops/_auto_tutorials/kmeans/plot_kmeans_torch.html

K Means Implementation

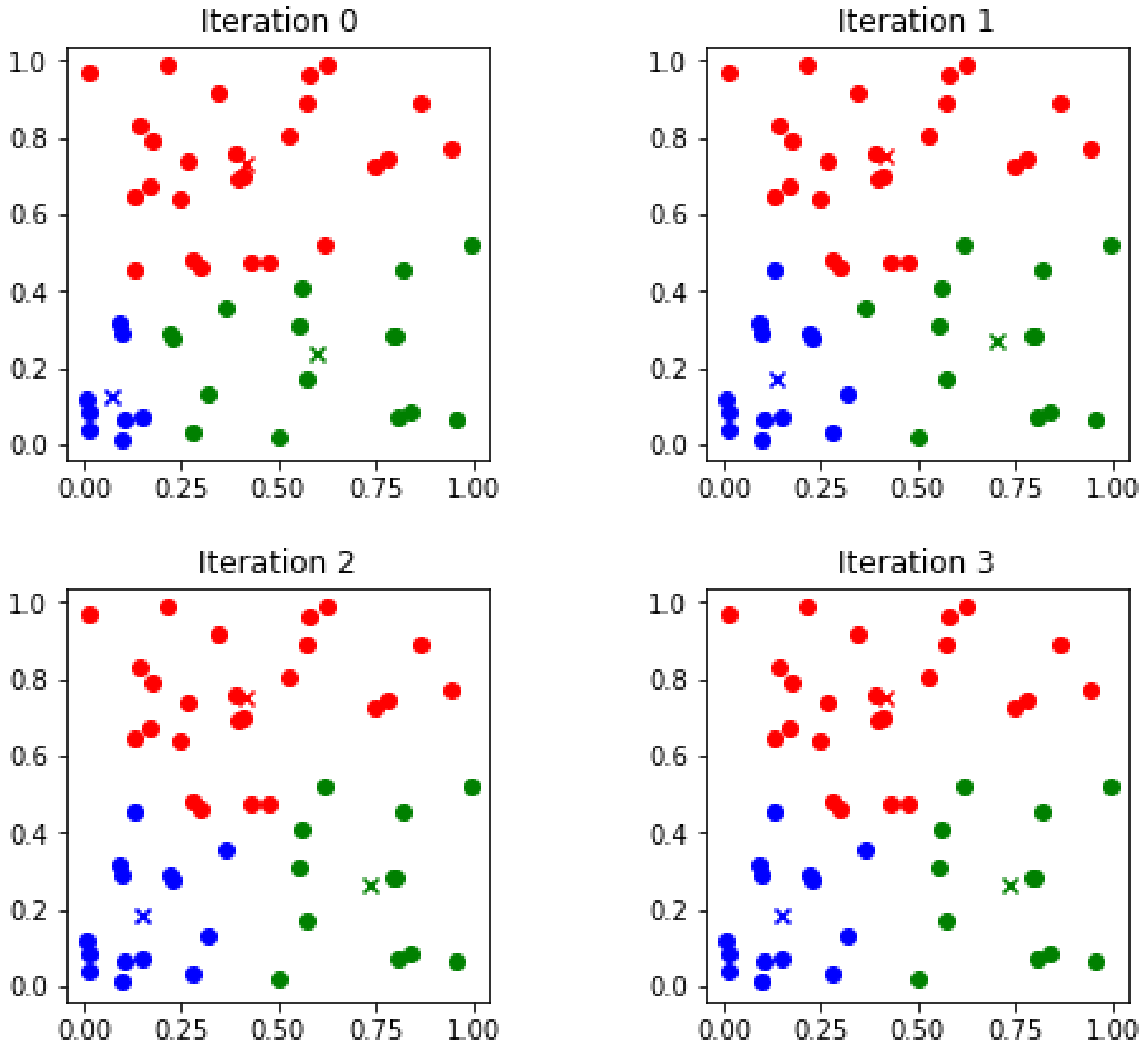
- Find squared distances between centroids and all points
- Cluster points by nearest mean/centroid
- Calculate new mean/centroid by taking mean of all data points in corresponding class

```
for i in range(Niter):

  # E step: assign points to the closest cluster -------------------------
  D_ij = ((x_i - c_j) ** 2).sum(-1)  # (N, K) symbolic squared distances
  cl = D_ij.argmin(dim=1).long().view(-1)  # Points -> Nearest cluster

  # M step: update the centroids to the normalized cluster average: ------
  # Compute the sum of points per cluster:
  c.zero_()
  c.scatter_add_(0, cl[:, None].repeat(1, D), x)

  # Divide by the number of points per cluster:
  Ncl = torch.bincount(cl, minlength=K).type_as(c).view(K, 1)
  c /= Ncl  # in-place division to compute the average
```

Code adapted from: https://www.kernel-operations.io/keops/_auto_tutorials/kmeans/plot_kmeans_torch.html

K Means implementation Visualization

```python

plt.figure(figsize = (3,3));
if i%3 == 0:
  cs = ['r', 'g', 'b']
  for j in range(K):
    mask = cl == j
    plt.scatter(x[:, 0][mask], x[:,1][mask], c = cs[j]);
    plt.title(f'Iteration {i}')
    plt.scatter(c[j, 0], c[j, 1], c = cs[j], marker = 'x');
```



K Nearest Neighbors

- Given labelled data points, **X**, predict the class of a new point ***y*** based on the class of its k nearest neighbors
- Whichever class has the most data points within the k nearest neighbors is assigned as the class of the new data point

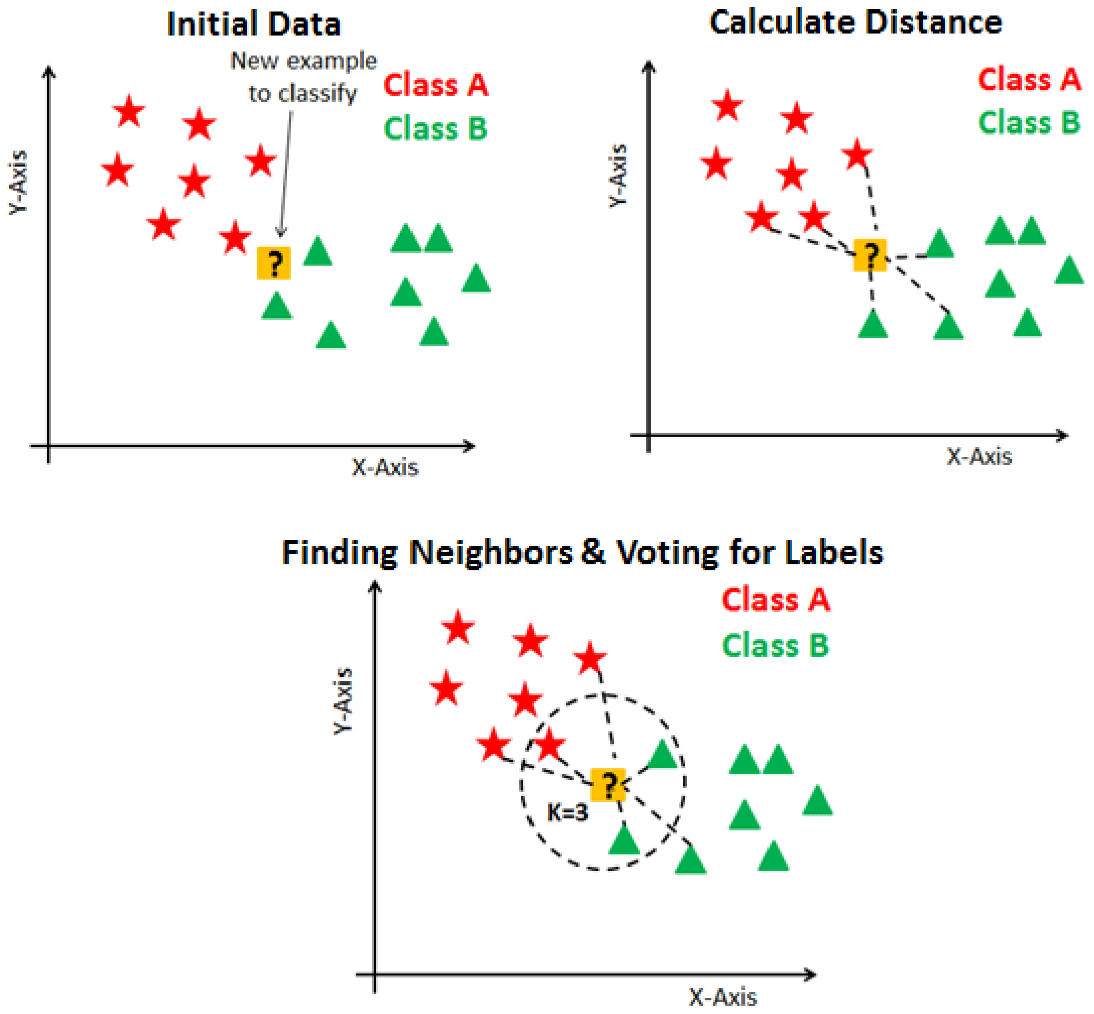


Image Source: https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn

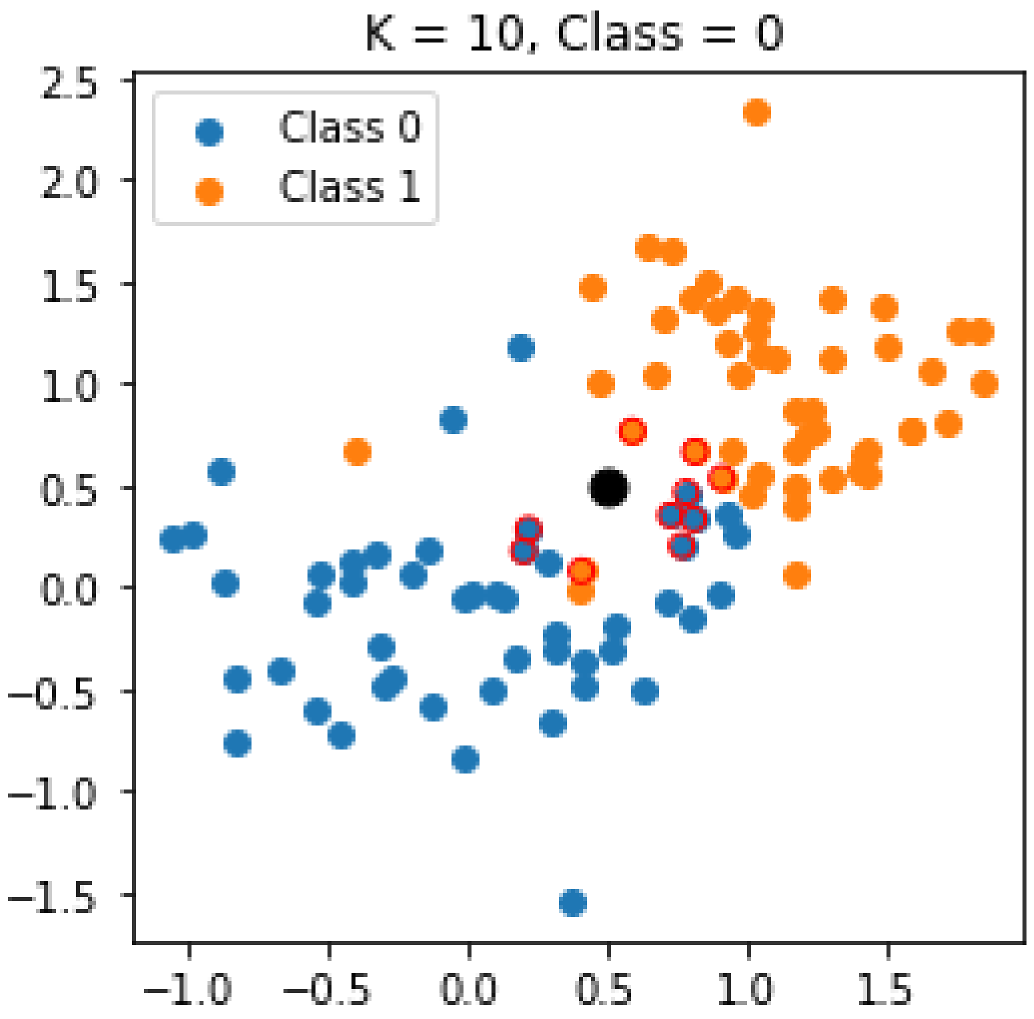
KNN Implementation Example

- If only one test point, can use simple subtraction and torch.norm() to find distance from each point
- torch.topk(___, largest=False) gives K smallest distances (nearest neighbors)
- Take torch.bincount() gives number of each index in an index tensor

```
# K Nearest Neighbor
#Generate Two classes
data_0 = torch.normal(torch.zeros(50, 2), 0.5*torch.ones(50,2))
labels_0 = torch.zeros(50).long()
data_1 = torch.normal(torch.ones(50, 2), 0.5*torch.ones(50, 2))
labels_1 = torch.ones(50).long()

full_data = torch.vstack((data_0, data_1))
labels = torch.cat((labels_0, labels_1))
mask = labels.bool()

#Test Data Point
test_point = torch.Tensor([[0.5,0.5]])

#Find distances
K = 10
dist = torch.norm(test_point - full_data, dim = 1)
k_dists, k_idcs = torch.topk(dist, k =K,  largest = False)
label = torch.argmax(torch.bincount(labels[k_idcs]))
```

KNN Implementation Example Visualization

```
plt.figure(figsize = (4,4))

#Plot each class
for l in [0,1]:
  d_mask = mask!=bool(l)
  plt.scatter(full_data[d_mask][:,0], full_data[d_mask][:, 1],
              label = f'Class {l}')
#Highlight K nearest neighbors
for k_idx in k_idcs:
  plt.scatter(full_data[k_idx, 0], full_data[k_idx, 1],
              facecolors = 'none', edgecolors= 'r')
plt.scatter(test_point[:, 0], test_point[:, 1], c = 'k', s = 100)
plt.title(f'K = {K}, Class = {label}')
plt.legend()
```

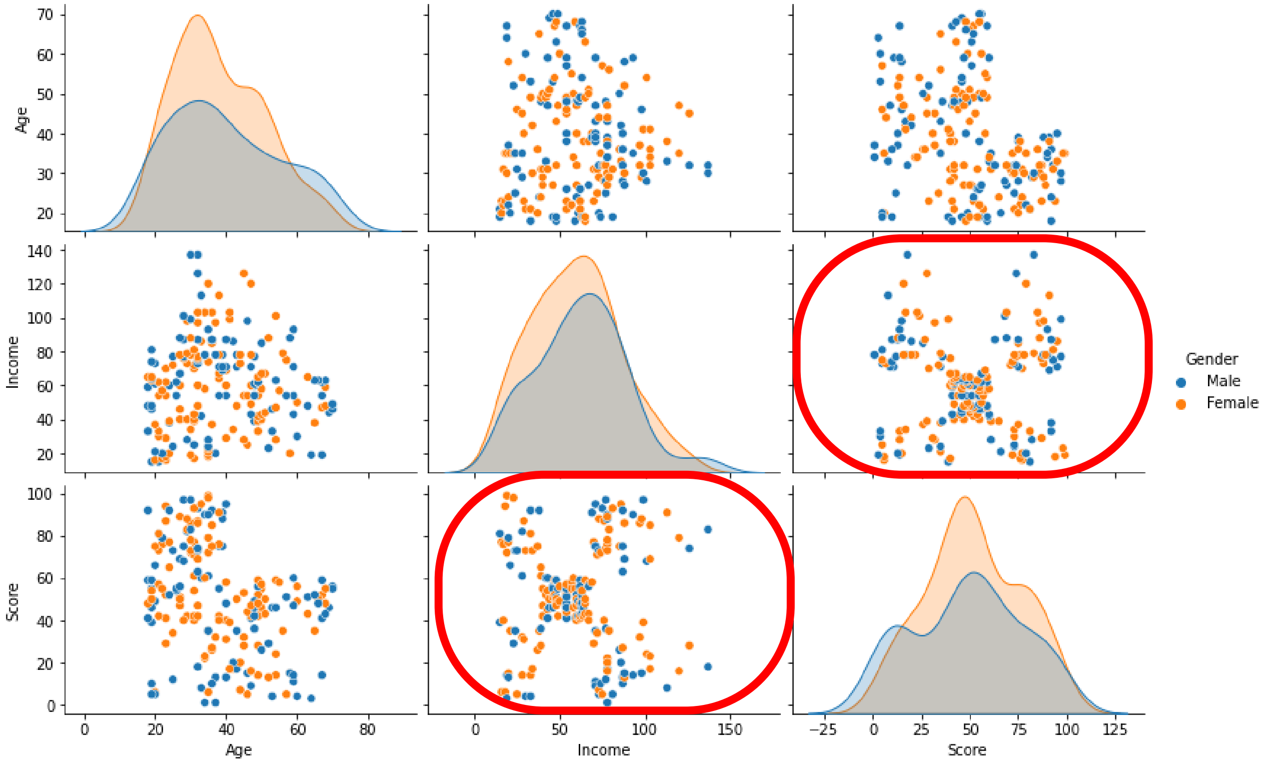


Example: Mall Customer Clustering

Source: https://www.kaggle.com/fazilbtopal/popular-unsupervised-clustering-algorithms/data

Data Structure

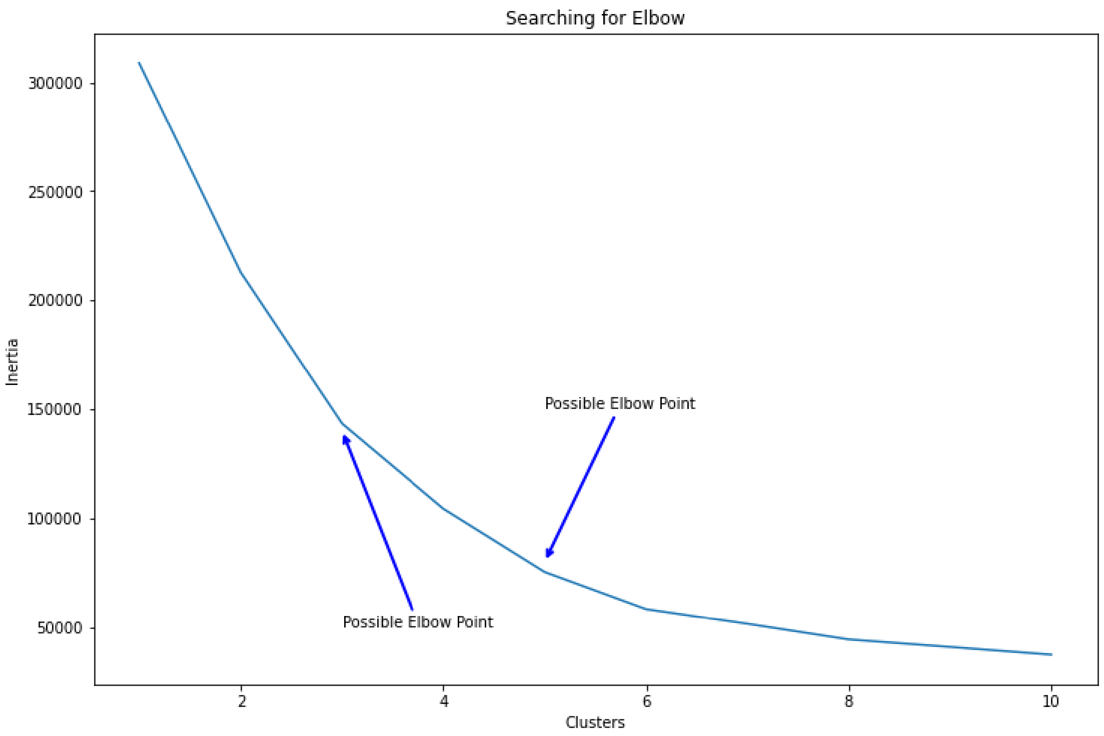
- Data contains the age, gender, income, and spending score for mall customers
- We can visualize this data along the different combinations of axes
- The combination that seems to have the best clustering is income and score



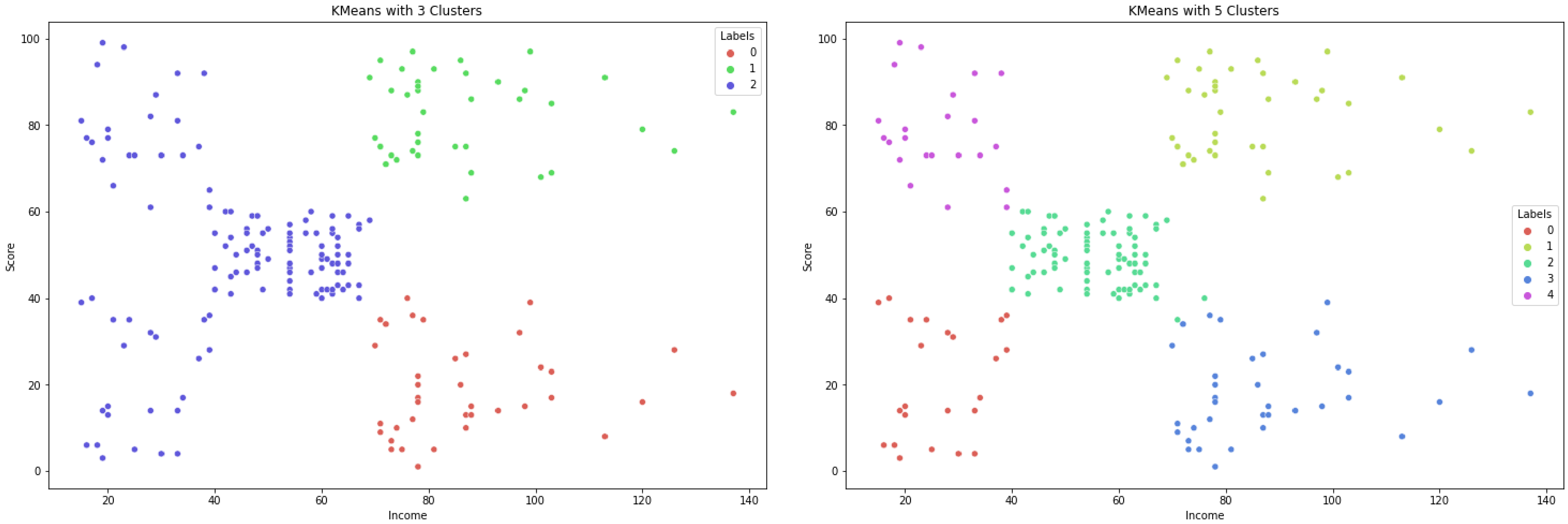
K-mean Clustering

- Find clusters for different values of k (from 1-11)
- Use plot to select value of k for which we see an “elbow”

```python
from sklearn.cluster import KMeans

clusters = []

for i in range(1, 11):
    km = KMeans(n_clusters=i).fit(X)
    clusters.append(km.inertia_)

fig, ax = plt.subplots(figsize=(12, 8))
sns.lineplot(x=list(range(1, 11)), y=clusters, ax=ax)
ax.set_title('Searching for Elbow')
ax.set_xlabel('Clusters')
ax.set_ylabel('Inertia')
```



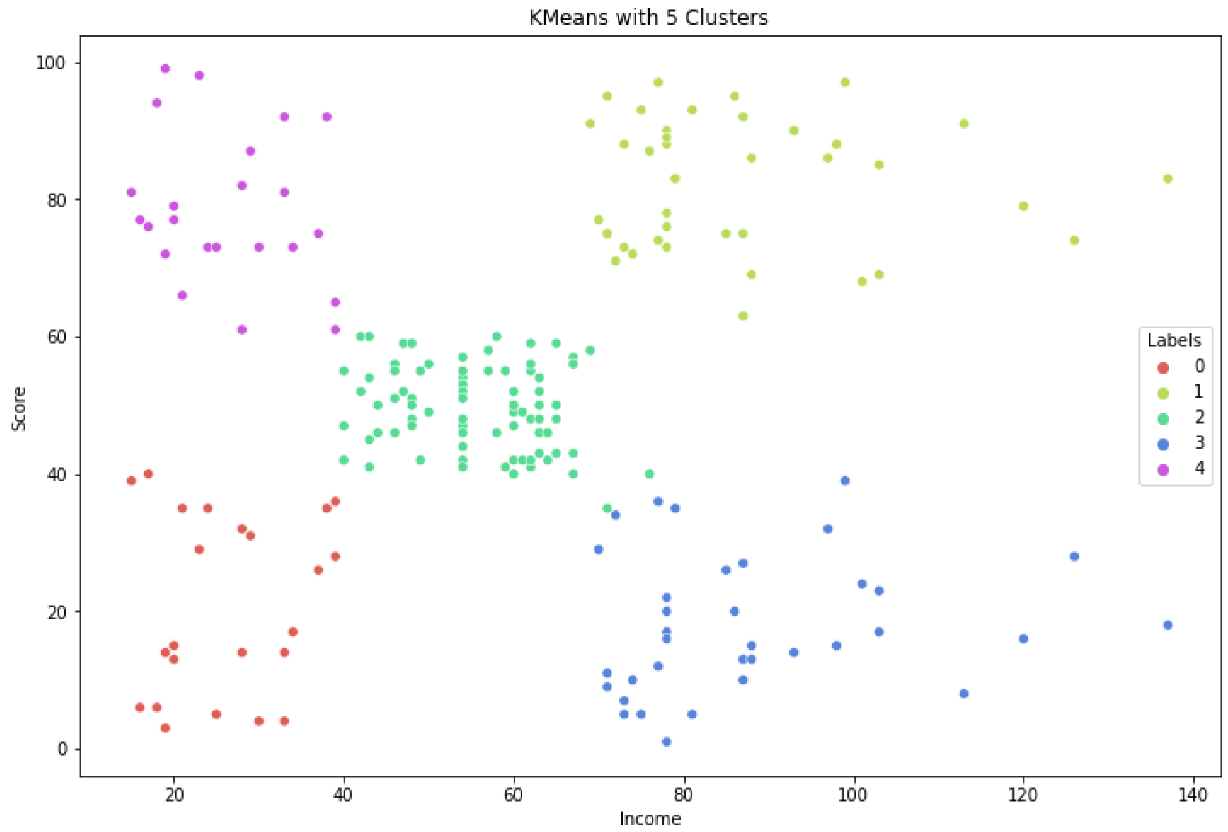
Visualizing Clusters



- Between the 3- and 5- cluster options, the one with 5 clusters looks like a better separation, leading to interpretable groups

Results Interpretation

- Label 0 is low income and low spending
- Label 1 is high income and high spending
- Label 2 is mid income and mid spending
- Label 3 is high income and low spending
- Label 4 is low income and high spending



Assignment: Credit Card Clustering and Classification

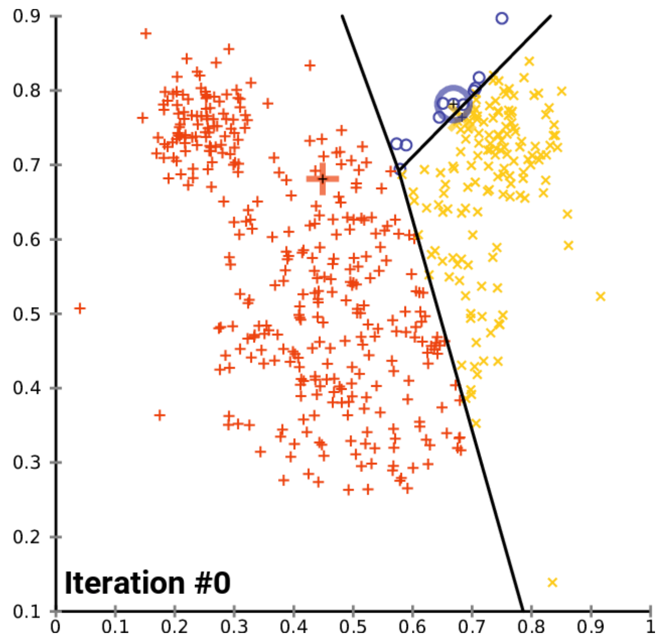
Data Source: https://www.kaggle.com/arjunbhasin2013/ccdata

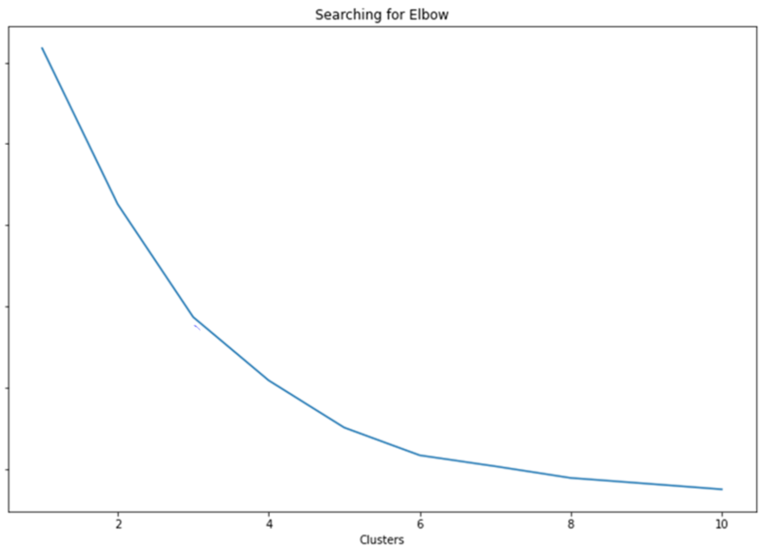
Data Exploration

- Data has 18 dimensions representing credit card customer habits, characteristics, etc.
- Scales of each dimension might be different, so it is recommended that you normalize each dimension so they are the same scale

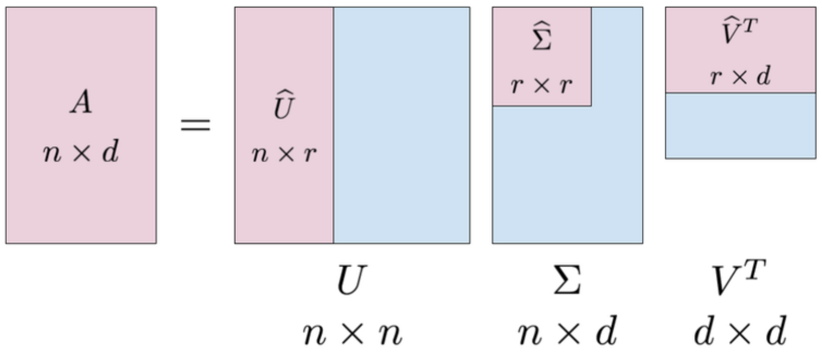
Assignment Details

1. Normalize each dimension of the data and split into train set (8000) and test set (1000)
2. Use k-means clustering to separate the customers into k groups
3. Find the "elbow" of the inertia vs. k plot to find the best value of k
4. Can you define what each of the clusters represents?
5. Take the PCA and SVD Decompositions of your training data and take the r-dimensional (choose r <=5) projection in these spaces.





Assignment Details



6. Transform the coordinates of your k centroids to the new space defined by the PCA and SVD.
7. For the test set, classify each data point by the nearest centroid in a) the original (full) coordinates, b) the PCA coordinates, c) the SVD coordinates
8. Use the original (full) coordinates as "truth", generate a confusion matrix for each of the other spaces. Which classes were most and least affected by coordinate transforms?