K-Means in Image Clustering

Angela Cao

Mentored by Shane McQuarrie


HOPE

Nearest Neighbors

- Nearest neighbor problem: given a dataset $X = \{\boldsymbol{x}_i\}^k_{i=1}$ and a new point $\boldsymbol{z}$, what is the point in X that is "nearest" to $\boldsymbol{z}$?

$$\min_i ||\boldsymbol{x}_i - \boldsymbol{z}||$$

- K-Nearest Neighbors Classifier: given a dataset X, corresponding labels $\boldsymbol{y}$, and a new point $\boldsymbol{z}$, to classify point $\boldsymbol{z}$ and give it a label, find the k labeled data points in $\boldsymbol{X}$ that are “nearest” to $\boldsymbol{z}$ and assign to $\boldsymbol{z}$ the majority label of those nearest neighbors. This is a *supervised learning algorithm* because we have the labels $\boldsymbol{y}$.

Find the subset $\{\boldsymbol{x}_{ij}\}^k_{j=1}$ that minimizes $\Sigma^k_{j=1} || \boldsymbol{x}_{ij} - \boldsymbol{z}||^2$

Then set $y = \text{mode}(y_{ij})$

K-Means

- K-Means algorithm: given a dataset X, determine k points (“cluster centers”) that define clusters in the data. A point $\boldsymbol{x}_j$ is in center i if it is closer to the ith cluster center than to any of the other cluster centers.
- This is an *unsupervised learning algorithm* because we do not have any labels (in general, this is much harder than supervised learning).

The Algorithm

Input: a data set X and a number of clusters k.

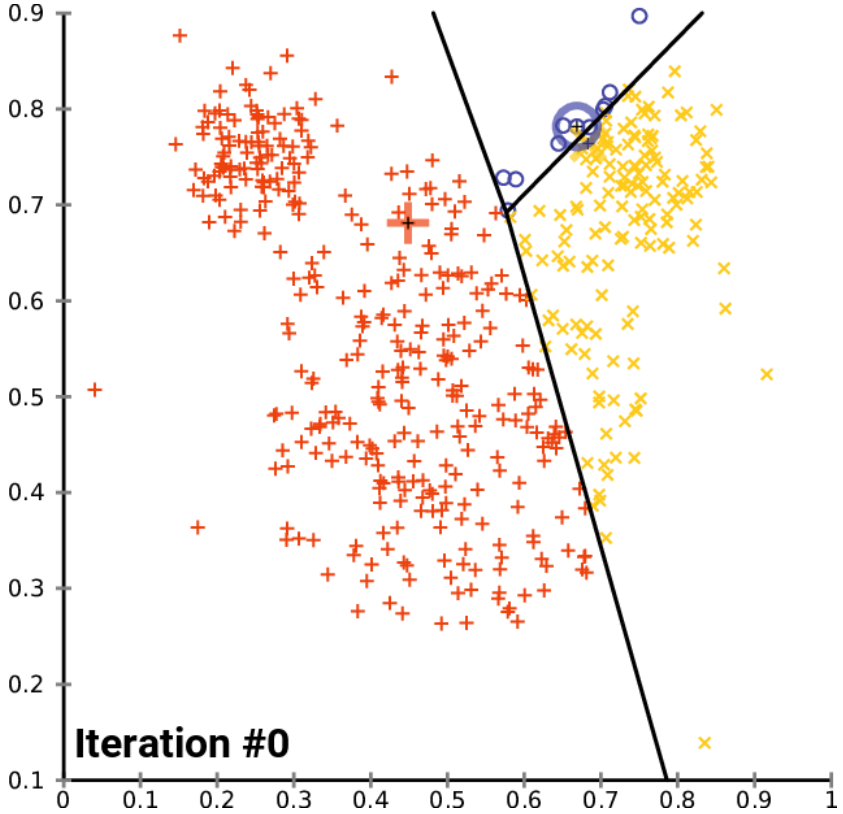
- Select k initial cluster centers $\boldsymbol{\mu}_1 \ldots \boldsymbol{\mu}_k$ (e.g., choose k points from X).
- Until the cluster centers stop changing, iterate:
  - For each point $\boldsymbol{x}_i$ in the data,
    - Calculate which cluster center $\boldsymbol{\mu}_j$ the data point $\boldsymbol{x}_i$ is closest to (NN)
  - Define clusters $S_1, \ldots S_k$, where S_j is the collection of points that are closer to $\boldsymbol{\mu}_j$ than to any of the other cluster centers.
  - Choose new cluster centers as the average of the points in each cluster, that is, $\boldsymbol{\mu}_j = \text{mean}(S_j)$.

Output: k final cluster centers $\boldsymbol{\mu}_1 \ldots \boldsymbol{\mu}_k$.

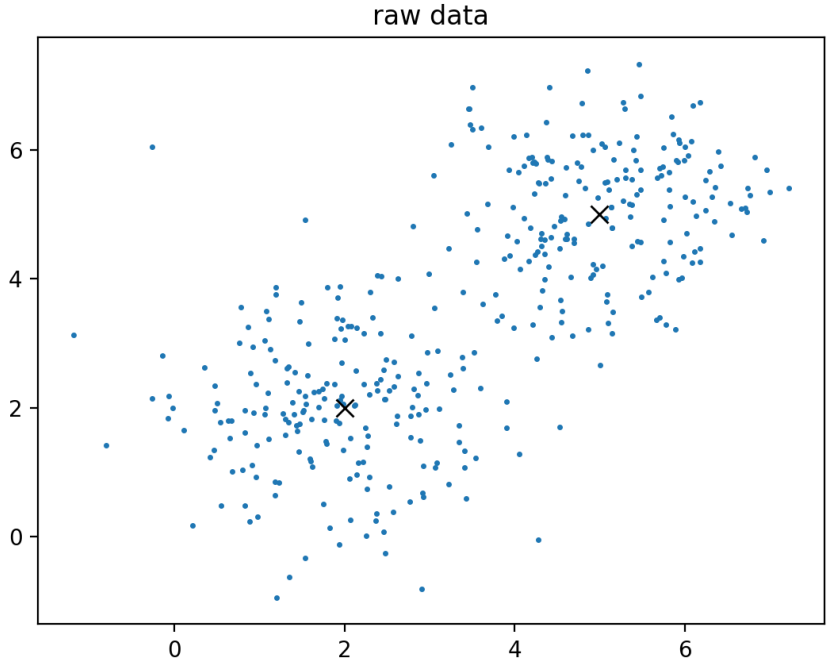
```
import numpy as np
from scipy import linalg as la

def kmeans(X, k, tol=1e-6):
    """Do k-means clustering!

    O(mnki) cost where i is the number of iterations.

    Parameters
    ----------
    X : (m,n) ndarray
        The data to cluster. Each row is an individual
        entry in the data set (with n features).

    k : int
        How many clusters (we have to choose this).

    tol : float
        Stopping tolerance. Stop the iteration when
        |centers_old - centers_new| < tol.

    Returns
    -------
    y : (m,) ndarray
         Labels for the clusters, e.g., y[i] = 2 means
         X[i,:] belongs to cluster 2.

    centers : (k,n) ndarray
        The centers of the final clusters.
    """
```

```
def kmeans(X, k, tol=1e-6):
    m,n = X.shape

    # Get k initial cluster centers by choosing points in X.
    centers = X[np.random.choice(m, k, replace=False)]

    # Do the iteration: decide which center each row is closest to.
    distances = np.empty((m,k))     # distances[i,j] = |X[i] - center[j]|
    diff = np.inf
    while diff > tol:
        # Calculate distances.
        for j in range(k):
            distances[:,j] = la.norm(X - centers[j], axis=1, ord=2)

        # Choose the smallest distance for each point.
        y = np.argmin(distances, axis=1)

        # Update the centers.
        centers_new = np.empty((k,n))
        for j in range(k):
            centers_new[j] = np.mean(X[y == j], axis=0)

        # Calculate the difference between old and new centers.
        diff = la.norm(centers - centers_new, ord=2)

        # Update.
        centers = centers_new

    return y, centers
```

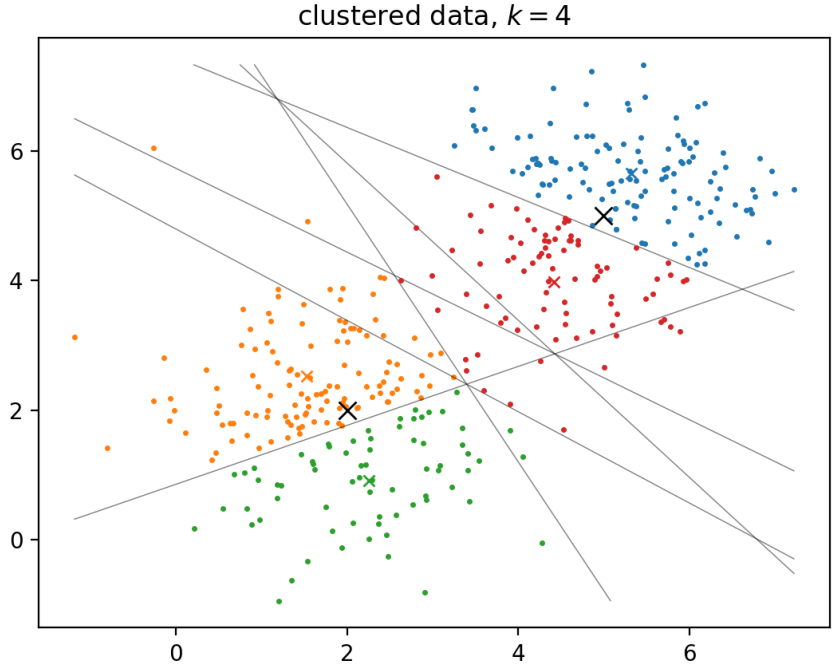
The Iteration



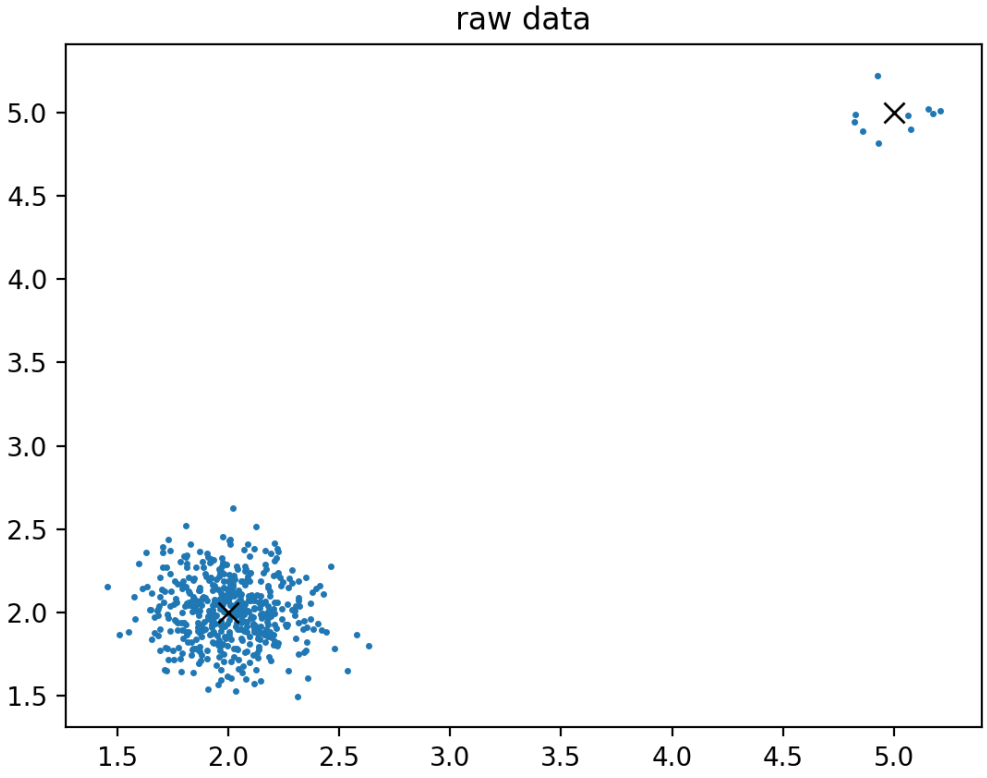
Example: 2D Data



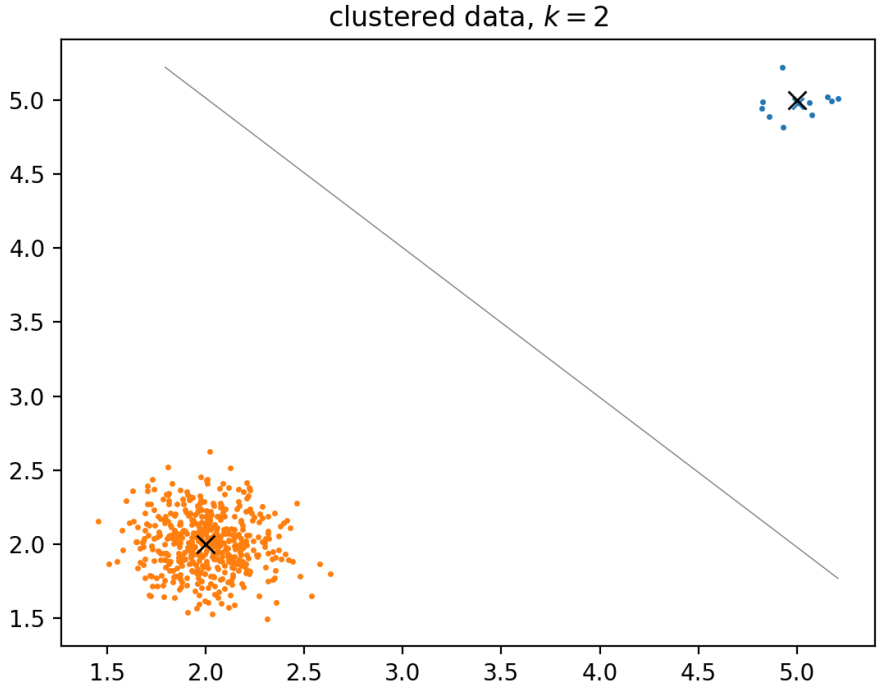
Example: 2D Data (bad number of clusters)

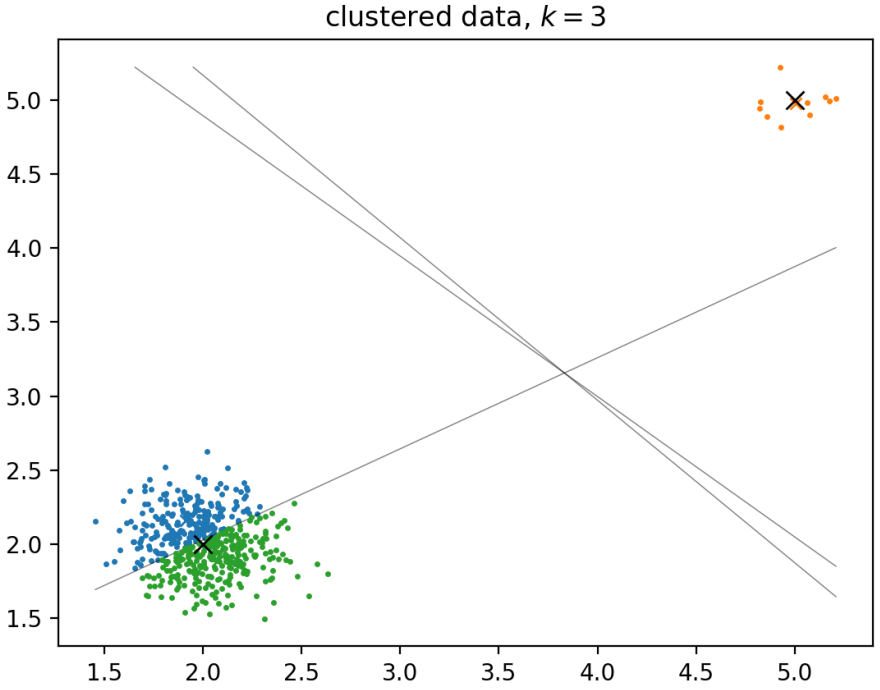


Example: 2D Data (class imbalance)



Example: 2D Data





Pros

- It does clustering, which is very hard in general.
- Relatively inexpensive with an algorithmic complexity of O(*mnkd*).
- Simple and easy to implement.

Cons

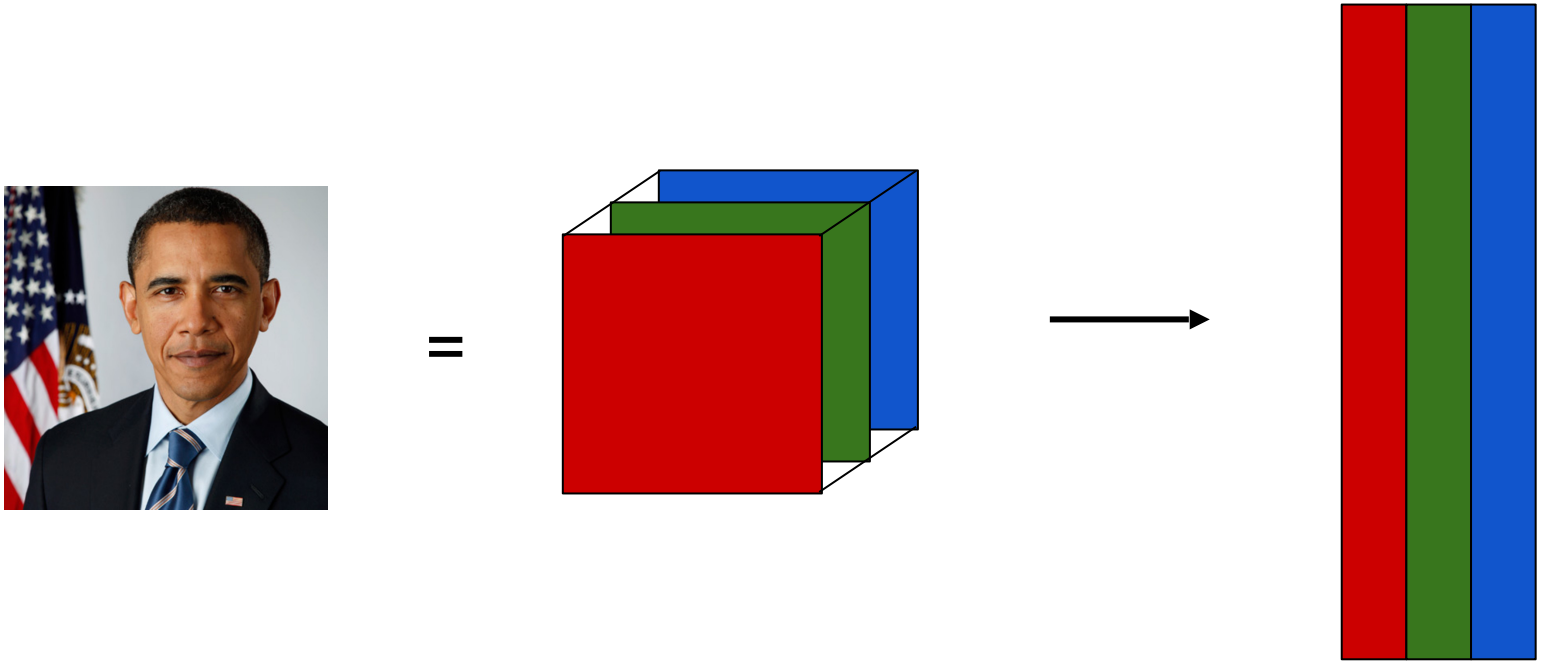
- The general problem of finding the best clusters is NP-hard and our solution is only an approximation.
- We have to choose *k* beforehand and there isn't always a great way to determine if your choice was good or not.
- You can get different results if you choose different starting points for the cluster centers (though there are ways to try to cleverly choose "good" starting points).
- If you have one huge cluster (lots of points) and one small cluster (few points), you can get unsatisfactory results.

Applications of K-Means: Pixel Quantization

Our goal is to approximate a natural image with only k colors (this can be good for compression, and it also just looks cool). Specifically, we will use k-means clustering on image data to organize all the pixels into k color groups.

The data set X is the original image; each point $\boldsymbol{x}_i$ in the data is a single pixel. The cluster centers that we get from k-means should be representative of the image (and much better than choosing random colors from the image).

Applications of K-Means: Pixel Quantization



```
import imageio
import numpy as np
from scipy import linalg as la
From sklearn.cluster import KMeans

def cluster_image(im, k):
    """Cluster image pixels.

    Parameters
    ----------
    im : (r,c,d) ndarray
        Original color image.

    k : int > 0
        Number of clusters.


    Returns
    -------
    imnew : (r,c,d) ndarray
        Clustered color image.
    """
```

```
# Get the dimensions of the image.
nrows, ncols, ndim = im.shape
m = nrows * ncols                    # number of pixels.
n = ndim                             # number of features per pixel.
X = im.reshape((m,n))                # reshape the data for clustering.

# Do the clustering algorithm with our KMeans function.
y, centers = kmeans(X, k)

# # Alternatively, do the clustering algorithm with Scikit-learn.
# Xclustered = KMeans(n_clusters=k, n_init=5).fit(X)
# centers = Xclustered.cluster_centers_
# y = Xclustered.predict(X)

# Post-process the cluster centers for image purposes.
centers = np.clip(centers, 0, 255).astype(np.uint8)

# Create the new picture.
Xnew = np.empty_like(X)
for j in range(k):
    Xnew[y == j] = centers[j]
imnew = Xnew.reshape(im.shape)

return imnew
```

Results

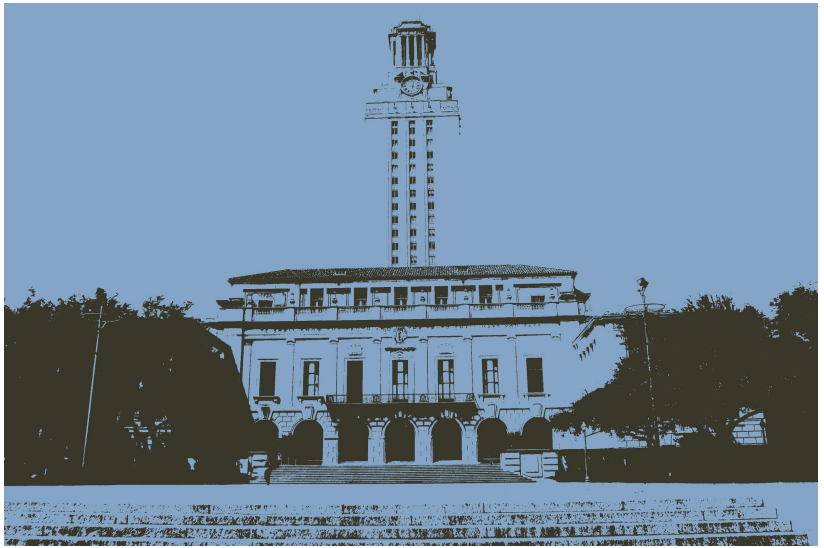
Original

Clustered, $k = 5$

Results

Original

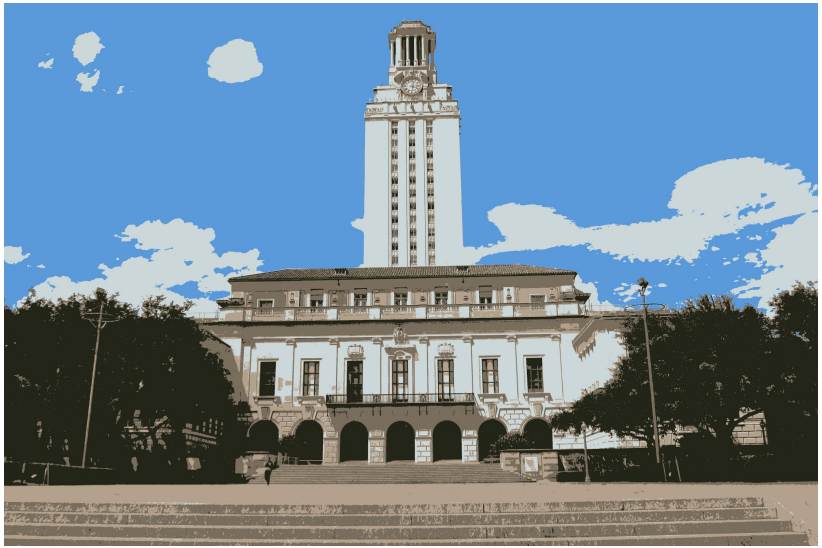
Clustered, $k = 2$



Results

Original

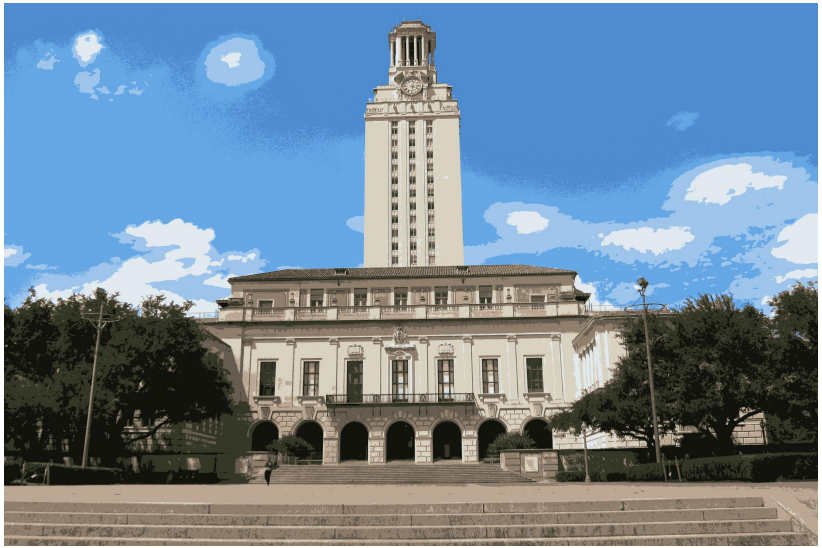
Clustered, $k = 5$



Results

Original

Clustered, *k = 10*



Results

Original

Clustered, $k = 2$

Results

Original

Clustered, $k = 5$

Results

Original

Clustered, *k = 20*

Summary

- Nearest Neighbors is a supervised learning problem that shows up often in computer vision, image processing, and other machine learning tasks.
- K-Means is an unsupervised learning algorithm for clustering. It can be thought of as the inverse of the *k*-nearest neighbors algorithm.
- We can apply *K*-Means to Pixel Quantization, resulting in an image filter algorithm. The larger the constant *k*, the more accurate the altered picture will be to the original.