Homework Assignment 3

CSE 251A: ML - Learning Algorithms

Due: May 10th, 2022, 9:30am (Pacific Time)

Instructions: Please answer the questions below, attach your code in the document, and insert figures to create **a single PDF file**.

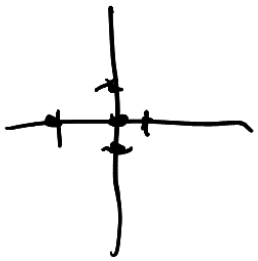
Grade: ____ out of 100 points

1 (20 points) SVM and kernel

1. (Linearly separability)(5 points) In lecture, we mentioned **Hard SVM** only works for **linearly separable** data points. Now you are given dataset $S = \{(\mathbf{x}_i, y_i), i = 1, .., N\}$ where each data point $(\mathbf{x}, y)$ contains a feature vector $\mathbf{x} \in \mathbb{R}^2$ and a label $y \in \{+, -\}$. Is the dataset linearly separable? If yes, please specify a decision boundary that linearly separates the data points with positive label from the data points with negative label.

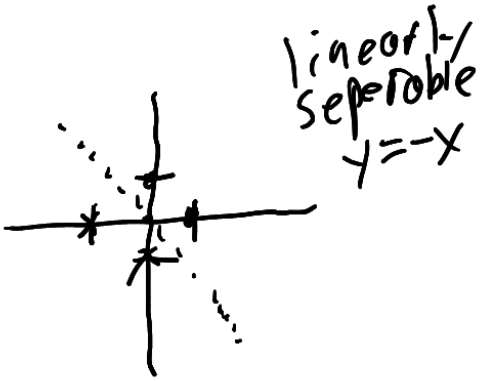
   - Case 1:

label (y)	features ($\mathbf{x}$)
Positive (+)	$(0, 0)$
Negative (−)	$(1, 0), (0, 1), (-1, 0), (0, -1)$



   - Case 2:

label (y)	features ($\mathbf{x}$)
Positive (+)	$(0, 1), (1, 0)$
Negative (−)	$(-1, 0), (0, -1)$



2. (Feature map and kernel)(10 points) Sometimes, data points that are not linearly separable in one feature space might be linearly separable in another. Therefore, we need a **transformation function (feature map)** ϕ, that maps features from one space to another space. Suppose for feature vector $\mathbf{x} = (x_1, x_2) \in \mathbb{R}^2$ we define **feature map** $\phi : \mathbb{R}^2 \to \mathbb{R}^3$ as: $\phi(\mathbf{x}) = \left(x_1^2, x_2^2, \sqrt{2}x_1x_2\right)$. Given the **feature map** ϕ, the corresponding **kernel** is defined as: $k(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{z})$, where $\mathbf{x}, \mathbf{z} \in \mathbb{R}^2$. Prove that $k(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z})^2$ is the **kernel** for ϕ. In other words, prove that $\phi(\mathbf{x}) \cdot \phi(\mathbf{z}) = (\mathbf{x} \cdot \mathbf{z})^2$.

Hint: for vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, $\mathbf{x} \cdot \mathbf{y} = x_1y_1 + x_2y_2 + \ldots + x_ny_n$.

$\phi x = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$ $\quad (x_1z_1 + x_2z_2)^2$

$\phi z = (z_1^2, z_2^2, \sqrt{2}z_1z_2)$ $\quad (x_1z_1 + x_2z_2)(x_1z_1 + x_2z_2)$

$x_1^2z_1^2 + x_2^2z_2^2 + \sqrt{2}\sqrt{2}x_1z_1x_2z_2$ $\quad x_1^2z_1^2 + x_2^2z_2^2 + 2x_1z_1x_2z_2$

$= x_1^2z_1^2 + x_2^2z_2^2 + 2x_1z_1x_2z_2$

3. (5 points) Recall Case 1 in part 1. Apply ϕ defined in part 2 to the 5 data points in Case 1 and compute the new features in $\mathbb{R}^3$. Are they now linearly separable in $\mathbb{R}^3$? (Optional: specify a valid decision boundary in $\mathbb{R}^3$ if linearly separable)

$(0,0) \to (0,0,0)$

$(1,0) \to (1,0,0)$

$(0,1) \to (0,1,0)$

$(-1,0) \to (-1,0,0)$

not linearly separable

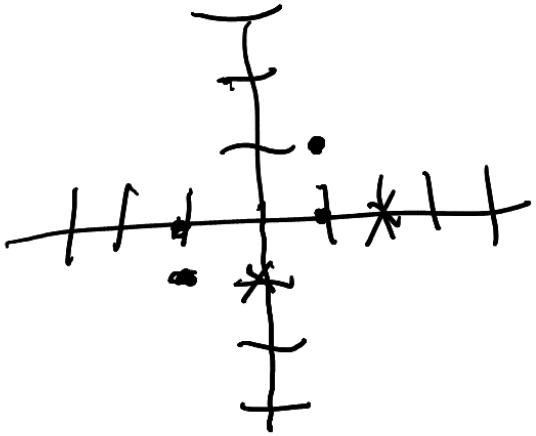
2 (10 points) K-means

Suppose you are given 4 data points: $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4 \in \mathbb{R}^2$, where $\mathbf{x}_1 = (1, 0), \mathbf{x}_2 = (1, 1)$, $\mathbf{x}_1 = (-1, 0), \mathbf{x}_1 = (-1, -1)$. Assume we know these points can be put into 2 clusters ($k = 2$). Suppose we start with **centroids** $\mathbf{c}_1 = (2, 0), \mathbf{c}_2 = (0, -1)$. Calculate where the new centroids would be after 1 iteration of **K-means**. Do the new centroids locations make sense?

dist euc.

c_1 $x_1 = 1$

$x_2 = \sqrt{2}$



c_2 $x_3 = \sqrt{2}$

$x_4 = 1$

new centroids

$$c_1 \left(\frac{1+1}{2}, \frac{0+1}{2} \right) = (1, .5)$$

$$c_2 \left(\frac{-1-1}{2}, \frac{0-1}{2} \right) = (-1, -.5)$$

3 (10 points) Gaussian Mixture Models

Suppose we have a set of probabilistic clusters, $\mathbf{C} = (C_1, C_2)$ .C_1 has probability density function $f_1 = \mathcal{N}(0, 1)$, that is, a Gaussian distribution with mean 0 and variance 1 , and C_2 has probability density function $f_2 = \mathcal{N}(1, 1)$. The weights of these clusters are given as: $\mathbb{P}(C_1) = \mathbb{P}(C_2) = 0.5$. Given the following data points in 1D, calculate the probability $\mathbb{P}(x \mid \mathbf{C})$ that a data point x is generated by this set of clusters $\mathbf{C}$, for the following 2 data points: $(1)x = 0.7, (2)x = 1.5$.

Hint: The probability density function of $\mathcal{N}(\mu, \sigma)$ is: $f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp \frac{-(x-\mu)^2}{2\sigma^2}$

$P(C|x) = P(x|C) \cdot P(C) / P(x)$

$x = .7$

$P(x|C1) = \frac{1}{\sqrt{2\pi}} e^{\frac{-(x-0)^2}{2}} = \frac{e^{\frac{-(.7)^2}{2}}}{\sqrt{2\pi}} = \frac{e^{-.245}}{\sqrt{2\pi}}$

$P(x|C2) = \frac{1}{\sqrt{2\pi}} e^{\frac{-(x-1)^2}{2}} = \frac{e^{\frac{-(.7-1)^2}{2}}}{\sqrt{2\pi}} = \frac{e^{-.045}}{\sqrt{2\pi}}$

$x = 1.5$

$P(x|C1) = \frac{1}{\sqrt{2\pi}} e^{\frac{-(1.5-0)^2}{2}} = \frac{e^{\frac{-(1.5)^2}{2}}}{\sqrt{2\pi}} = \frac{e^{-1.125}}{\sqrt{2\pi}}$

$P(x|C2) = \frac{1}{\sqrt{2\pi}} e^{\frac{-(1.5-1)^2}{2}} = \frac{e^{\frac{-(.5)^2}{2}}}{\sqrt{2\pi}} = \frac{e^{-.125}}{\sqrt{2\pi}}$

$P(x=.7) = \frac{e^{-.245}}{\sqrt{2\pi}} \cdot (.5) + \frac{e^{-.045}}{\sqrt{2\pi}} \cdot (.5) = \frac{.391}{1.77} + \frac{.477}{1.77}$

$= .22 + .26 = .48$

$P(x=1.5) = \frac{e^{-1.125}}{\sqrt{2\pi}} \cdot (.5) + \frac{e^{-.125}}{\sqrt{2\pi}} \cdot (.5) = \frac{.164}{1.77} + \frac{.44}{1.77}$

$= .09 + .24 = .33$

$x = .7$

$P(C1|x) = \frac{.44 \cdot .5}{.48} = .45$

$P(C2|x) = \frac{.54 \cdot .5}{.48} = .55$

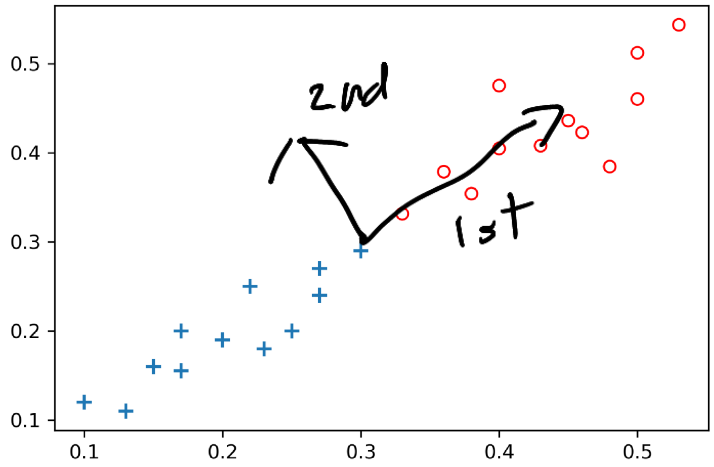
$x = 1.5$

$P(C1|x) = \frac{.18 \cdot .5}{.33} = .25$

$P(C2|x) = \frac{.44 \cdot .5}{.33} = .75$

4 (10 points) Principle Component Analysis (PCA)

You are now given some data points in 2D, with 2 kinds of labels ("+" and "o").



(1) (5 points) On the scatter plot, roughly draw the direction for the 1st and 2nd principle axes for this dataset.

(2) (5 points) Which principle axis would you choose to project onto, if we want to project the 2D features onto 1D, and still want good separation between the 2 labels?

1st axis

5 (50 points) Implementing the K-means algorithm

Now, you will implement a K-Means model from scratch. We have provided a skeleton code file (i.e. KMeans.py) for you to implement the algorithm as well as a notebook file (i.e. KMeans.ipynb) for you to conduct experiments and answer relevant questions. Libraries such as numpy and pandas may be used for auxiliary tasks (such as matrix multiplication, matrix inversion, and so on), but not for the algorithms. That is, you can use numpy to implement your model, but cannot directly call libraries such as scikit-learn to get a K-Means model for your skeleton code. We will grade this question based on the three following criteria:

1. Your implementation in code. Please do not change the structure of our skeleton code.
2. Your model's performance (we check if your model behaves correctly based on the results from multiple experiments in the notebook file).
3. Your written answers for questions in the notebook file.

Import necessary packages

You'll be implement your model in `KMeans.py` which should be put under the same directory as the location of `KMeans.ipynb`. Since we have enabled `autoreload`, you only need to import these packages once. You don't need to restart the kernel of this notebook nor rerun the next cell even if you change your implementation for `KMeans.py` in the meantime.

A suggestion for better productivity if you never used jupyter notebook + python script together: you can split your screen into left and right parts, and have your left part displaying this notebook and have your right part displaying your `KMeans.py`

In [5]:
```
%load_ext autoreload
%autoreload 2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from KMeans import KMeans

from sklearn import datasets
from sklearn.datasets import make_blobs
from sklearn.metrics.cluster import adjusted_mutual_info_score
from sklearn.cluster import KMeans as Ref
from sklearn.manifold import TSNE, MDS
from sklearn.metrics import mean_squared_error
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

In [ ]:
```
import numpy as np

class KMeans():
    # This function initializes the KMeans class
    def __init__(self, k = 3, num_iter = 1000, order = 2):
        # Set a seed for easy debugging and evaluation
        np.random.seed(42)

        # This variable defines how many clusters to create
        # default is 3
        self.k = k

        # This variable defines how many iterations to recompute centroids
        # default is 1000
        self.num_iter = num_iter

        # This variable stores the coordinates of centroids
        self.centers = None

        # This variable defines whether it's K-Means or K-Medians
        # an order of 2 uses Euclidean distance for means
        # an order of 1 uses Manhattan distance for medians
        # default is 2
        if order == 1 or order == 2:
            self.order = order
        else:
            raise Exception("Unknown Order")

    # This function fits the model with input data (training)
    def fit(self, X):
        # m, n represent the number of rows (observations)
        # and columns (positions in each coordinate)
        m, n = X.shape

        # self.centers are a 2d-array of
        # (number of clusters, number of dimensions of our input data)
        self.centers = np.zeros((self.k, n))

        # self.cluster_idx represents the cluster index for each observation
        # which is a 1d-array of (number of observations)
        self.cluster_idx = np.zeros(m)

        ##### TODO 1 ######
        #
        # Task: initialize self.centers
        #
        # Instruction:
        # For each dimension (feature) in X, use the 10th percentile and
        # the 90th percentile to form a uniform distribution. Then, we will initialize
        # the values of each center by randomly selecting values from the distributions.
        #
        # Note:
        # This method is by no means the best initialization method. However, we would
        # like you to follow our guidelines in this HW. We will ask you to discuss some better
        # initializaiton methods in the notebook.
        #
        # Hint:
        # 1. np.random.uniform(), np.percentile() might be useful
        # 2. make sure to look over its parameters if you're not sure
        ####################
        for i in range(n):
            low, high = np.percentile(X[:, i], [10, 90])
```

```
            dist = np.random.uniform(low, high, self.k)

            self.centers[:,i] = dist
        ##### END TODO 1 #####

        for i in range(self.num_iter):
            # new_centers are a 2d-array of
            # (number of clusters, number of dimensions of our input data)
            new_centers = np.zeros((self.k, n))

            ##### TODO 2 ######
            #
            # Task: calculate the distance and create cluster index for each observation
            #
            # Instruction:
            # You should calculate the distance between each observation and each centroid
            # using specified self.order. Then, you should derive the cluster index for
            # each observation based on the minimum distance between an observation and
            # each of the centers.
            #
            # Hint:
            # 1. np.linalg.norm() might be useful, along with parameter axis, ord
            # for that function
            # 2. You can transpose an array using .T at the end
            # 3. np.argmin() might be useful along with parameter axis in finding
            # the desired cluster index of all observations
            #
            # IMPORTANT:
            # Copy-paste this part of your implemented code
            # to the predict function, and return cluster_idx in that function
            ####################
            # distance = np.array()
            if self.order == 1:
            # Use Manhattan distance for medians
                distances = np.abs(X[:, None] - self.centers).sum(axis=2)
            elif self.order == 2:
            # Use Euclidean distance for means
                distances = np.linalg.norm(X[:, None] - self.centers, axis=2)

            # Get the index of the closest centroid for each observation
            cluster_idx = np.argmin(distances, axis=1)

            ##### END TODO 2 #####

            ##### TODO 3 ######
            #
            # Task: calculate the coordinates of new_centers based on cluster_idx
            #
            # Instruction:
            # You should assign the coordinates of the new_center by calculating
            # mean/median of the coordinates of observations belonging to the same
            # cluster.
            #
            # Hint:
            # 1. np.mean(), np.median() with axis might be helpful
            ####################

            for idx in range(self.k):
                cluster_coordinates = X[cluster_idx == idx]
                if self.order == 2:
                    cluster_center = np.mean(cluster_coordinates, axis=0)
                elif self.order == 1:
                    cluster_center = np.median(cluster_coordinates, axis=0)
                new_centers[idx, :] = cluster_center

            ##### END TODO 3 #####

            ##### TODO 4 ######
            #
            # Task: determine early stop and update centers and cluster_idx
            #
            # Instructions:
            # You should stop tranining as long as cluster index for all
            # observations is the same as the previous iteration
            # Hint:
            # 1. .all() might be helpful
            ####################

            if (cluster_idx == self.cluster_idx).all():
                print(f"Early Stopped at Iteration {i}")
                return self

            self.centers = new_centers
            self.cluster_idx = cluster_idx
            ##### END TODO 4 #####
        return self


    # This function makes predictions with input data
    # Copy-paste your code from TODO 2 and return cluster_idx
    def predict(self, X):
            if self.order == 1:
            # Use Manhattan distance for medians
```

```
            distances = np.abs(X[:, None] - self.centers).sum(axis=2)
        else:
        # Use Euclidean distance for means
            distances = np.linalg.norm(X[:, None] - self.centers, axis=2)

        # Get the index of the closest centroid for each observation
        cluster_idx = np.argmin(distances, axis=1)

        return cluster_idx
```

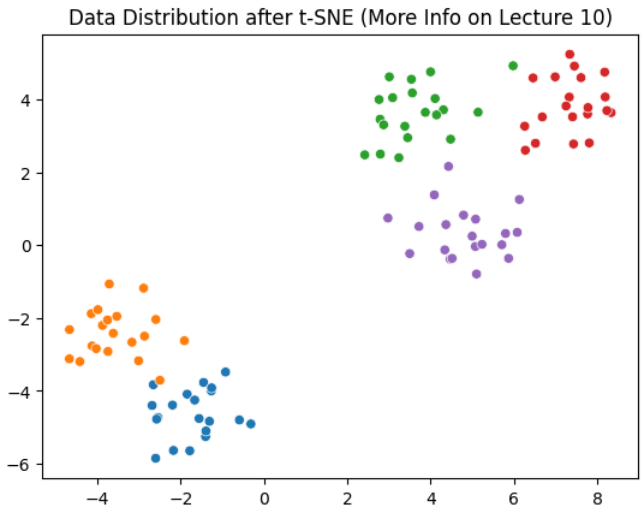
ATTENTION: THERE ARE A TOTAL OF 6 QUESTIONS THAT NEED YOUR ANSWERS

Experiment: Synthetic Data

First, let's play with our model on some synthetic data that have clear separation for different clusters. Here, let's make a dataset of 100 elements in 5 different clusters with 10 dimensions and visualize it by manifolding it into a 2D space with t-SNE.

Note: The distance that you can observe from the t-SNE visualization may be significantly different from the real distance due to the manifold embedding. Please refer to the PCA and T-SNE lecture for more details.

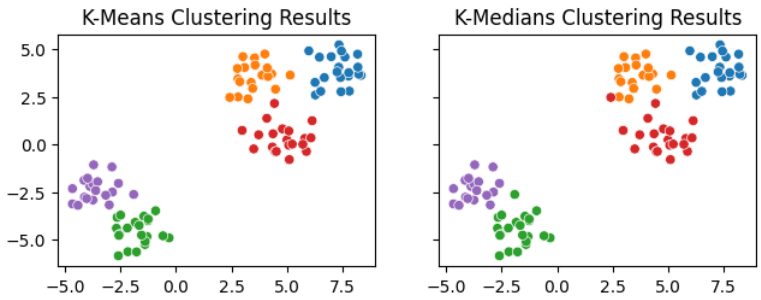
In [6]:
```
X, y = make_blobs(n_samples=100, centers=5, n_features=10, random_state=42, cluster_std=2, center_box=(0, 10))

dims = TSNE(random_state=42).fit_transform(X)
dim1, dim2 = dims[:, 0], dims[:, 1]
sns.scatterplot(x=dim1, y=dim2, hue=y, palette='tab10', legend=False)
plt.title('Data Distribution after t-SNE (More Info on Lecture 10)');
```



Now, let's see how our algorithm performs compared to the ground truth.

In [7]:
```
fig, axes = plt.subplots(1, 2, figsize=(7.5, 2.5), sharey=True)

# This is a reference of KMeans from sklearn's implementation, which we will be using later to evaluate our model
ref_kmeans = Ref(5, init='random').fit(X).predict(X)

# This is to evaluate our KMeans model predictions
y_pred_kmeans = KMeans(5, order=2).fit(X).predict(X)
sns.scatterplot(x=dim1, y=dim2, hue=y_pred_kmeans, palette='tab10', ax=axes[0], legend=False)
axes[0].set_title('K-Means Clustering Results')

# This is to evaluate our KMedians model predictions
y_pred_kmedians = KMeans(5, order=1).fit(X).predict(X)
sns.scatterplot(x=dim1, y=dim2, hue=y_pred_kmedians, palette='tab10', ax=axes[1], legend=False)
axes[1].set_title('K-Medians Clustering Results');
```

```
Early Stopped at Iteration 3
Early Stopped at Iteration 3
c:\Users\mwood\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init`
will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```



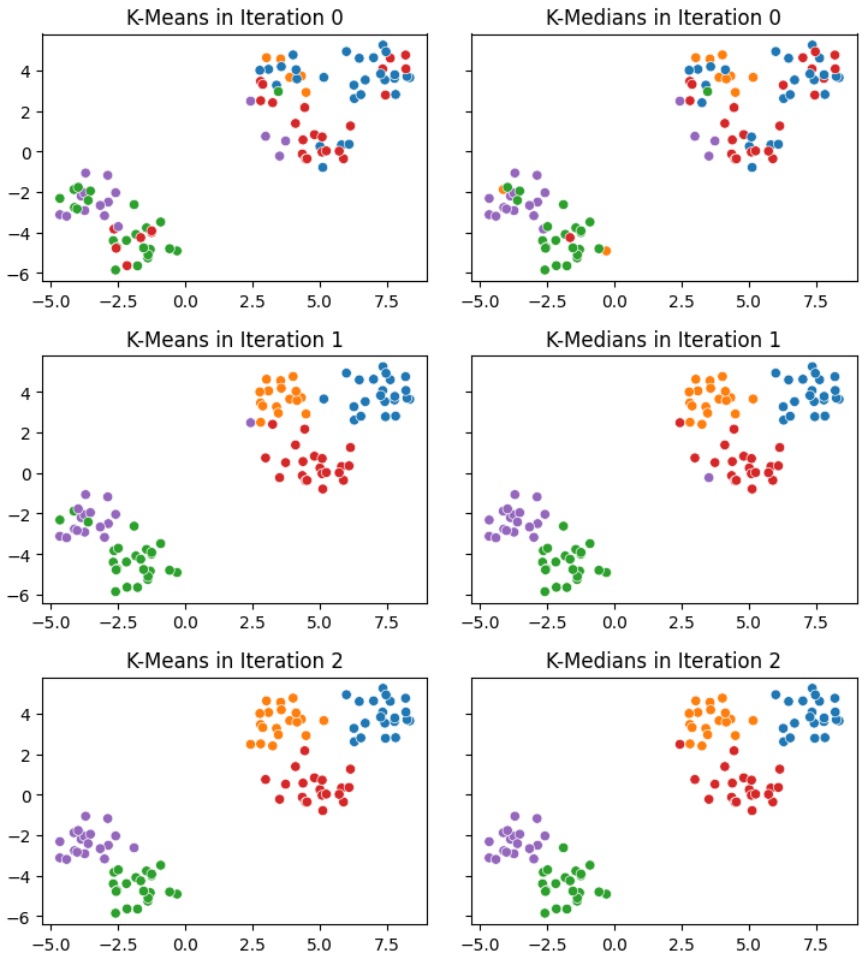
Question 1: From the above two figures, which one seems better compared to the original data distribution with actual cluster indices? Can you list some possible reasons why one way performs better than the other way?

Hint: Think of how we make the synthetic data. Also, next cell block shows the detailed clustering progress over each iteration.

Answer:The K-means seems to work better due to the similarity it shares with the original data. The K-medians method is classifying some points that don't look like they belong to their specific clusters, especially compared to the original data. This could be due to how sensitive the K-median method is to outliers. The K-medians clustering method chooses a point that should belong to orange as one belonging to zero. Since outliers are better handled with k-medians, it will make it more likely to classify an outlier as part of a cluster.

Let's see how the clustering goes over each iteration

In [8]:
```
fig, axes = plt.subplots(3, 2, figsize=(7.5, 8), sharey=True)
fig.tight_layout()
plt.subplots_adjust(hspace=0.3)

# Don't worry about the fact that we train a separate model for each iteration
# since we used a fixed random seed to ensure initialization consistency
for i in range(3):
    y_pred = KMeans(5, num_iter=i, order=2).fit(X).predict(X)
    ax = axes[i][0]
    ax.title.set_text(f'K-Means in Iteration {i}')
    sns.scatterplot(x=dim1, y=dim2, hue=y_pred, palette='tab10', ax=ax, legend=False)

    y_pred = KMeans(5, num_iter=i, order=1).fit(X).predict(X)
    ax = axes[i][1]
    ax.title.set_text(f'K-Medians in Iteration {i}')
    sns.scatterplot(x=dim1, y=dim2, hue=y_pred, palette='tab10', ax=ax, legend=False);
```



Let's now evaluate our models with respect to sklearn's model. Here, we will be using adjusted mutual information score as our metric to evaluate the performance of clustering.

Hint: If your model is correctly implemented, you should have one of the models (K-Means, K-Medians) to have the same mutual info score as sklearn's implementation.

In [9]:
```
pd.DataFrame({'Reference K-Means from Sklearn vs Ground Truth': adjusted_mutual_info_score(ref_kmeans, y),
              'Our K-Means vs Ground Truth': adjusted_mutual_info_score(y_pred_kmeans, y),
              'Our K-Medians vs Ground Truth': adjusted_mutual_info_score(y_pred_kmedians, y)},
             index=['Mutual Info Score']).T
```

Out[9]:

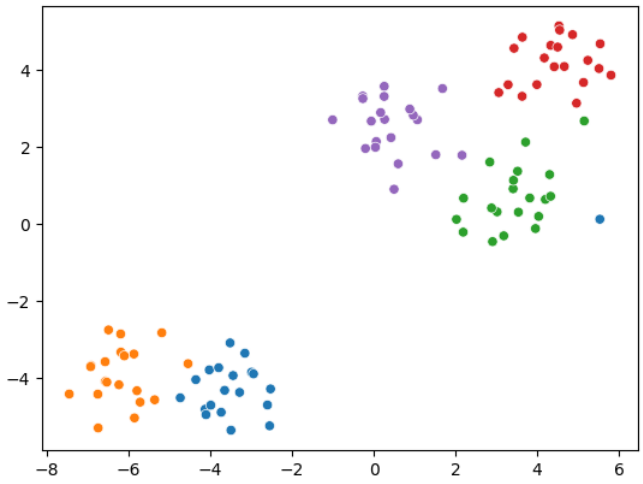
	Mutual Info Score
Reference K-Means from Sklearn vs Ground Truth	0.947515
Our K-Means vs Ground Truth	0.947515
Our K-Medians vs Ground Truth	0.904241

Wait... What happens when we have just one outlier?

Now, let's change one observation in the dataset to be an outlier. That is, we'll set the value of the first dimension of the first point in the dataset to be `100`. Other than that, the rest of the dataset is kept completely the same as before.

As you can see in the below visualization, the manifolded figrue with t-SNE shows that there started to have points belonging to a particular cluster (according to the ground truth) appearing in the side of anothor cluster. However, if we discard the coloring of the below figure, we can still see that our dataset roughly has 5 clusters and each cluster contains an equal size of observations. We will see if this will affect the performance of our models.

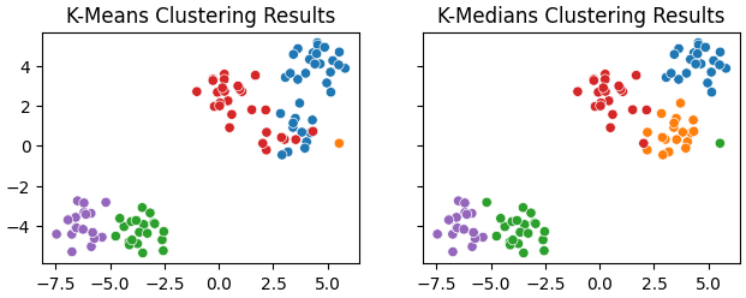
In [10]:
```
# let's make an outlier here
X[0][0] = 100

dims = TSNE(random_state=42).fit_transform(X)
dim1, dim2 = dims[:, 0], dims[:, 1]
sns.scatterplot(x=dim1, y=dim2, hue=y, palette='tab10', legend=False);
```



Now, let's see how our algorithm performs compared to the ground truth.

In [11]:
```
fig, axes = plt.subplots(1, 2, figsize=(7.5, 2.5), sharey=True)

# This is a reference of KMeans from sklearn's implementation, which we will be using later to evaluate our model
ref_kmeans = Ref(5, init='random').fit(X).predict(X)

# This is to evaluate our KMeans model predictions
y_pred_kmeans = KMeans(5, order=2).fit(X).predict(X)
sns.scatterplot(x=dim1, y=dim2, hue=y_pred_kmeans, palette='tab10', ax=axes[0], legend=False)
axes[0].set_title('K-Means Clustering Results')

# This is to evaluate our KMedians model predictions
y_pred_kmedians = KMeans(5, order=1).fit(X).predict(X)
sns.scatterplot(x=dim1, y=dim2, hue=y_pred_kmedians, palette='tab10', ax=axes[1], legend=False)
axes[1].set_title('K-Medians Clustering Results');
```

```
Early Stopped at Iteration 4
Early Stopped at Iteration 3
c:\Users\mwood\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init`
will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```



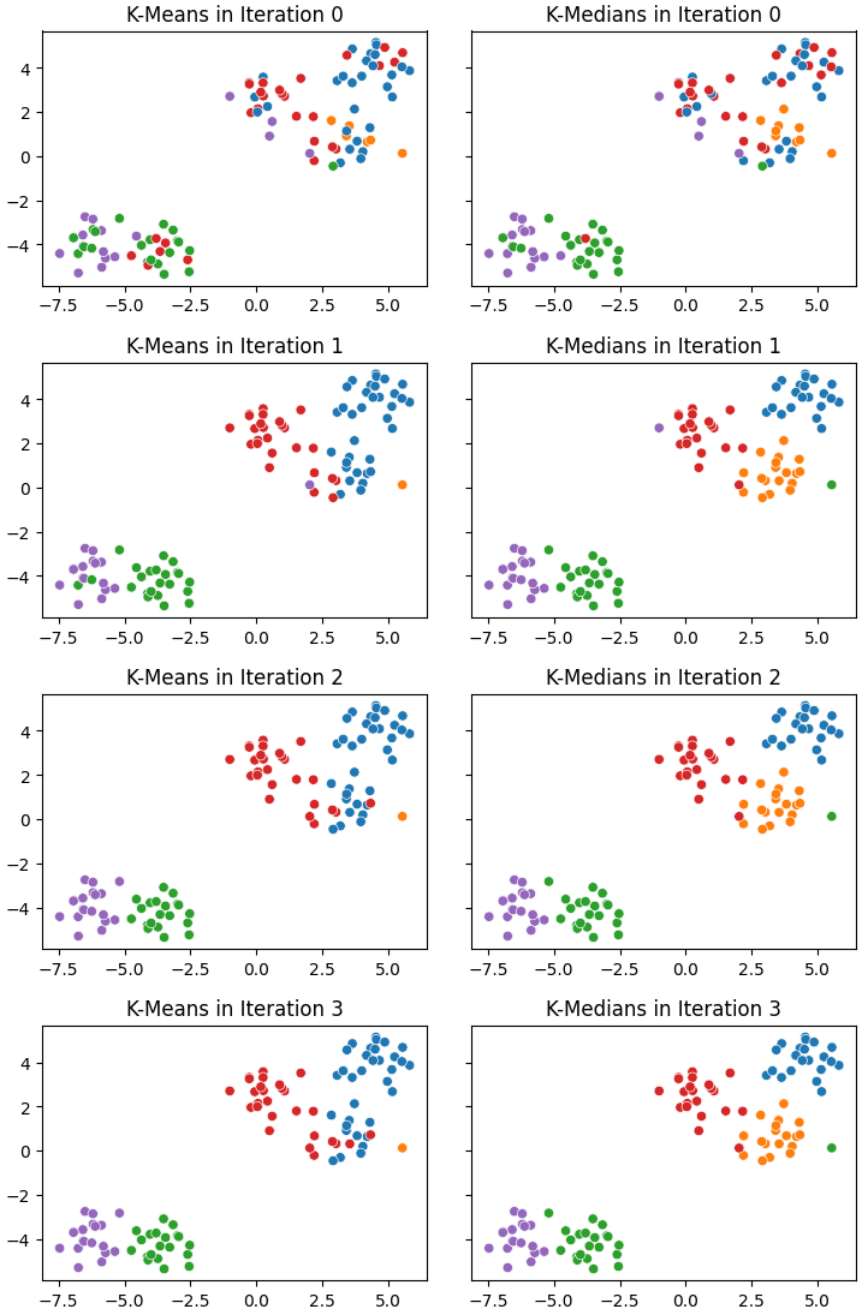
Question 2: From the above two figures, which one seems better compared to the original data distribution with actual cluster indices? Can you list some possible reasons why one way performs better than the other way?

Hint: Think of how we make the synthetic data. Also, think of the consequences of using means vs using medians in finding the centers.

Answer: The K-median result looks to be more in line with the original data. This is because K-median clustering is more sensitive to outliers which can better help classify data as seen with the green outlier matching the original data cluster in terms of classification. This is due to the K-median looking at the median or the middle value in the dataset instead of taking all points into account like the mean. With this, the influence of outliers in the data is not as pronounced when calculating spread for clusters.

Let's see how the clustering goes over each iteration

In [12]:
```
fig, axes = plt.subplots(4, 2, figsize=(7.5, 11), sharey=True)
fig.tight_layout()
plt.subplots_adjust(hspace=0.3)

for i in range(4):
    y_pred = KMeans(5, num_iter=i, order=2).fit(X).predict(X)
    ax = axes[i][0]
    ax.title.set_text(f'K-Means in Iteration {i}')
    sns.scatterplot(x=dim1, y=dim2, hue=y_pred, palette='tab10', ax=ax, legend=False)

    y_pred = KMeans(5, num_iter=i, order=1).fit(X).predict(X)
    ax = axes[i][1]
    ax.title.set_text(f'K-Medians in Iteration {i}')
    sns.scatterplot(x=dim1, y=dim2, hue=y_pred, palette='tab10', ax=ax, legend=False);
```



Let's now evaluate our models with respect to sklearn's model. Here, we will be using adjusted mutual information score as our metric to evaluate the performance of clustering.

Hint: If your model is correctly implemented, you should one of the scores higher than the reference score, and the other score lower than the reference score.

In [13]:
```
pd.DataFrame({'Reference KMeans from Sklearn vs Ground Truth': adjusted_mutual_info_score(ref_kmeans, y),
              'Our KMeans vs Ground Truth': adjusted_mutual_info_score(y_pred_kmeans, y),
              'Our KMedians vs Ground Truth': adjusted_mutual_info_score(y_pred_kmedians, y)},
              index=['Mutual Info Score']).T
```

Out[13]:

	Mutual Info Score
Reference KMeans from Sklearn vs Ground Truth	0.862091
Our KMeans vs Ground Truth	0.781036
Our KMedians vs Ground Truth	0.904241

Question 3: After the above experiments, (1)Can you summarize when is better to use Euclidean distance for K-Means, and when is better to use Manhattan distance for K-Medians? (2)If a model performs better on K-Medians than the most popular K-Means, what does that mean for the dataset? (3)Are there ways you can manipulate the dataset a little bit to make the model achieve a better performance on K-Means? (4)You may noticed that Sklearn's KMeans algorithms performs better than our KMeans algorithm. What could be the cause here?

Answer: (1) it is better to use euclidean distance for K-Means when dealing with numeric data and when the clusters are expected to have a spherical shape. This is because euclidean distance is measuring the distance as a straignt line which is more useful for classifying clusters with a circle shape. The manhatten distance is good for

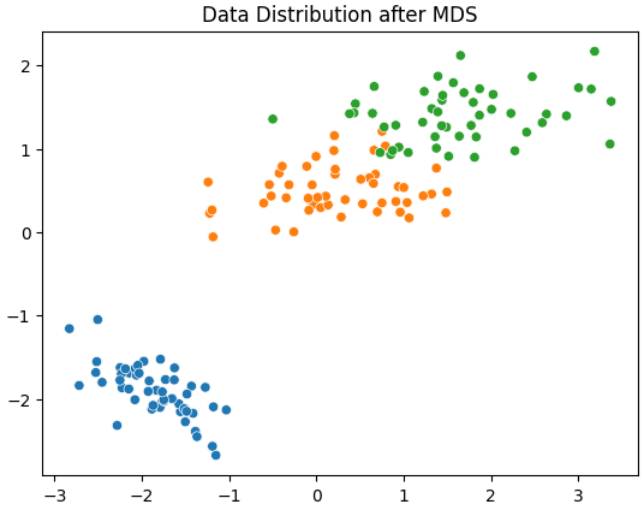
k-medians clustering due to finding the distance by counting the steps it takes to get to the point. There are less outliers when finding distance this way and can work with a wide variety of distribtution of the data. (2) It would suggest that the data contains outliers within the dataset. (3) You can normalize the data using any method such as min-max or log depending on the data as well as running PCA on the data to reduce the dimensions when calculating clusters. (4) The initialization may be different when comparing the centroids. The hyperparameters could be different and additional methods may be used to optimize the convergence speed.

Experiment: Real-World Data

Now, after that we have dealt with some synthetic data, which come from a normal distribution at different centers, we will evaluate our model's performance on real-world data. Here, we will be using the iris dataset. Let's first visualize our data using Multi-Dimensional Scaling, which is another way to visualize multi-dimensional data into a 2D space.

In [14]:
```
data = datasets.load_iris()
X, y = data['data'], data['target']

dims = MDS(random_state=42).fit_transform(X)
dim1, dim2 = dims[:, 0], dims[:, 1]
sns.scatterplot(x=dim1, y=dim2, hue=y, palette='tab10', legend=False)
plt.title('Data Distribution after MDS');
```

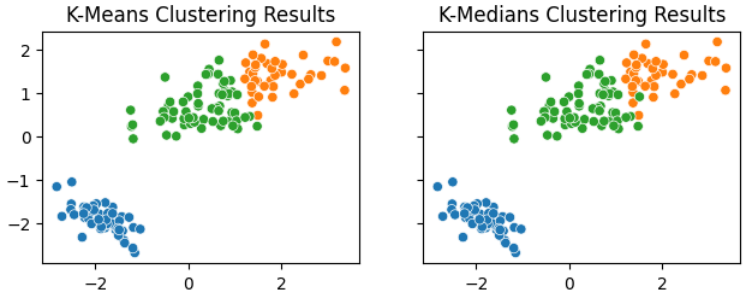
```
c:\Users\mwood\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\manifold\_mds.py:299: FutureWarning: The default value of `normalized
_stress` will change to ``'auto'`` in version 1.4. To suppress this warning, manually set the value of `normalized_stress`.
  warnings.warn(
```



Now, let's see how our algorithm performs compared to the ground truth.

In [15]:
```
fig, axes = plt.subplots(1, 2, figsize=(7.5, 2.5), sharey=True)

# This is a reference of KMeans from sklearn's implementation, which we will be using later to evaluate our model
ref_kmeans = Ref(3, init='random').fit(X).predict(X)

# This is to evaluate our KMeans model predictions
y_pred_kmeans = KMeans(3, order=2).fit(X).predict(X)
sns.scatterplot(x=dim1, y=dim2, hue=y_pred_kmeans, palette='tab10', ax=axes[0], legend=False)
axes[0].set_title('K-Means Clustering Results')

# This is to evaluate our KMedians model predictions
y_pred_kmedians = KMeans(3, order=1).fit(X).predict(X)
sns.scatterplot(x=dim1, y=dim2, hue=y_pred_kmedians, palette='tab10', ax=axes[1], legend=False)
axes[1].set_title('K-Medians Clustering Results');
```

```
Early Stopped at Iteration 5
Early Stopped at Iteration 4
c:\Users\mwood\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init`
will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```



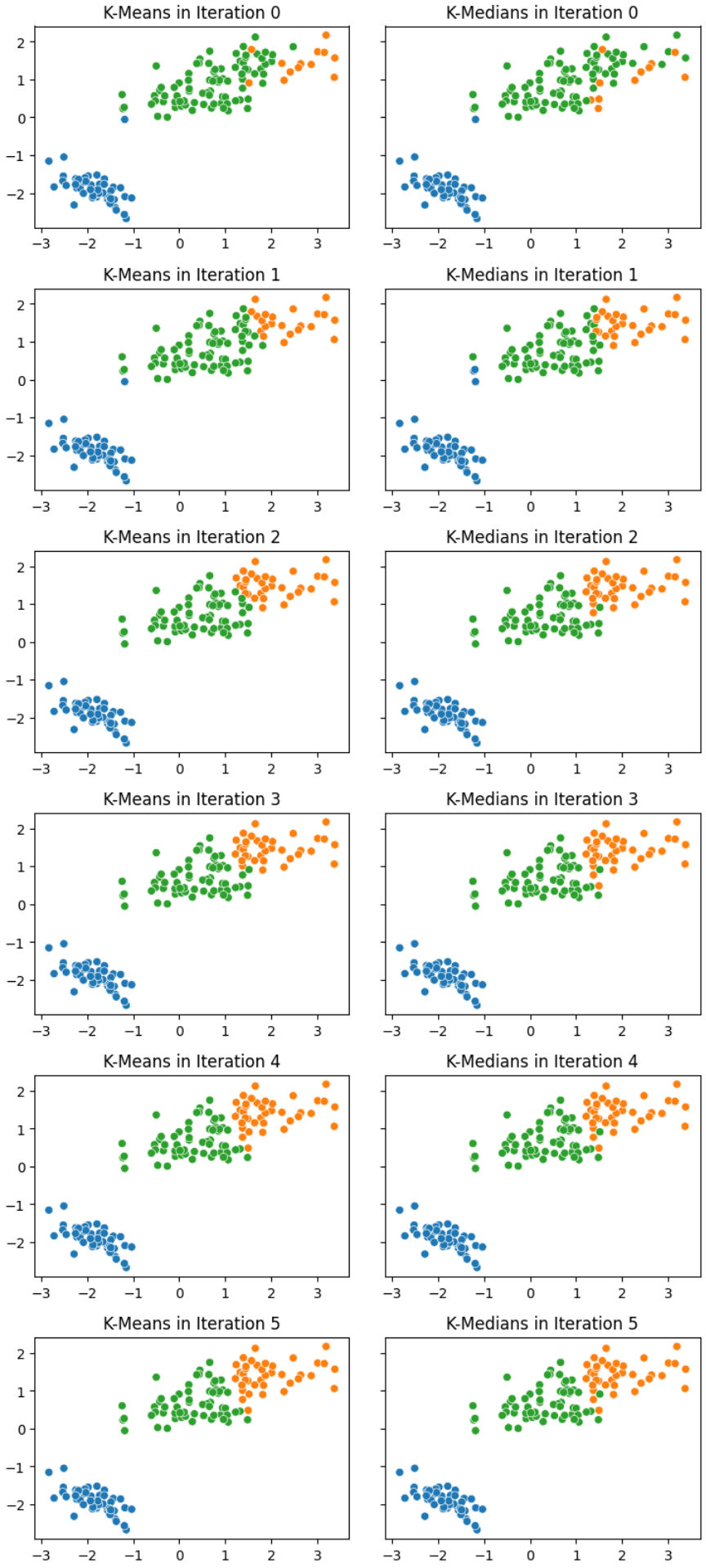
Question 4: From the above results, do our models still perform that well compared to the expereiment where we used the synthetic data? What makes the difference in real-life data?

Answer: The blue cluster looks to be exactly correct, but there is some differences between the top two clusters when comparing to the original dataset. The green cluster has additional points counted for its cluster as well. Both the K-means and medians clustering results show this distinction. Synthetic data also is generated based off assumptions and controlled patterns and may not be indicative of the actual spread of the data. Real data aso has noise and can be incomplete and is subject to many different factors that the model can't immediately capture.

Let's see how the clustering goes over each iteration

In [16]:
```
fig, axes = plt.subplots(6, 2, figsize=(7.5, 16), sharey=True)
fig.tight_layout()
plt.subplots_adjust(hspace=0.3)

for i in range(6):
    y_pred = KMeans(3, num_iter=i, order=2).fit(X).predict(X)
    ax = axes[i][0]
    ax.title.set_text(f'K-Means in Iteration {i}')
    sns.scatterplot(x=dim1, y=dim2, hue=y_pred, palette='tab10', ax=ax, legend=False)

    y_pred = KMeans(3, num_iter=i, order=1).fit(X).predict(X)
    ax = axes[i][1]
    ax.title.set_text(f'K-Medians in Iteration {i}')
    sns.scatterplot(x=dim1, y=dim2, hue=y_pred, palette='tab10', ax=ax, legend=False);
```

```
Early Stopped at Iteration 4
```


K-Means in Iteration 0
K-Medians in Iteration 0
K-Means in Iteration 1
K-Medians in Iteration 1
K-Means in Iteration 2
K-Medians in Iteration 2
K-Means in Iteration 3
K-Medians in Iteration 3
K-Means in Iteration 4
K-Medians in Iteration 4
K-Means in Iteration 5
K-Medians in Iteration 5

Let's now evaluate our models with respect to sklearn's model. Here, we will be using adjusted mutual information score as our metric to evaluate the performance of clustering.

Hint: If your model is correctly implemented, you should have one of the models (K-Means, K-Medians) to have the same mutual info score as sklearn's implementation.

In [17]:
```
pd.DataFrame({'Reference KMeans from Sklearn vs Ground Truth': adjusted_mutual_info_score(ref_kmeans, y),
              'Our KMeans vs Ground Truth': adjusted_mutual_info_score(y_pred_kmeans, y),
              'Our KMedians vs Ground Truth': adjusted_mutual_info_score(y_pred_kmedians, y)},
             index=['Mutual Info Score']).T
```

Out[17]:

	Mutual Info Score
Reference KMeans from Sklearn vs Ground Truth	0.755119
Our KMeans vs Ground Truth	0.755119
Our KMedians vs Ground Truth	0.747583

But most often time... we don't know how many clusters are there beforehand

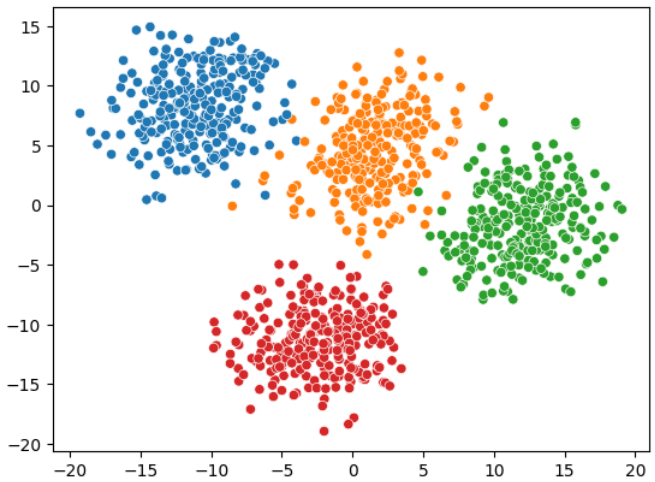
Let us turn back to the synthetic data. For this part of the experiment, there are 1000 points in our data, and it should have 4 features and 4 centers (or 4 different types/labels).

In [18]:
```
X, y = make_blobs(n_samples=1000, n_features=4, centers=4, cluster_std=2.5, random_state = 15)
```

In [19]:
```
dims = MDS(random_state=42).fit_transform(X)
dim1, dim2 = dims[:, 0], dims[:, 1]
sns.scatterplot(x=dim1, y=dim2, hue=y, palette='tab10', legend=False)
```

```
c:\Users\mwood\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\manifold\_mds.py:299: FutureWarning: The default value of `normalized
_stress` will change to `'auto'` in version 1.4. To suppress this warning, manually set the value of `normalized_stress`.
  warnings.warn(
```

Out[19]: <AxesSubplot: >



We will train our model using k from 2 to 10, and store the Sum of Squares Error per cluster for each k.

SSE per cluster is the squared distances between points in a cluster and the cluster center. It indicates how "compact" each cluster is.

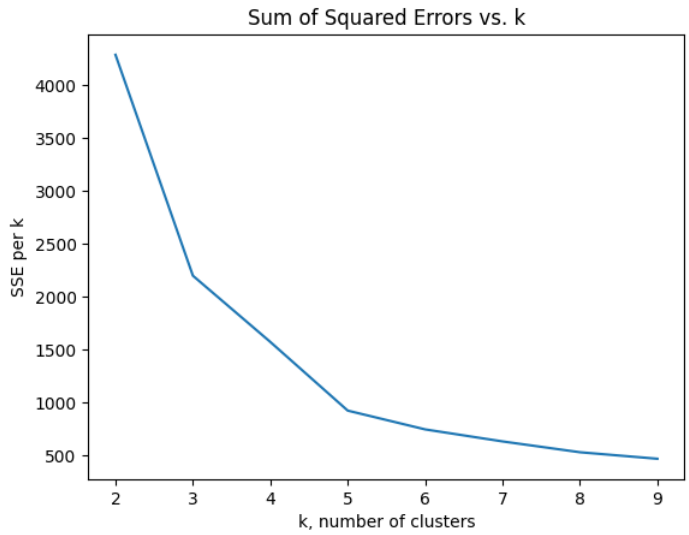
In [20]:
```
SSE = []
y_preds = []

for i in range(2, 10):
    clf = KMeans(i, order=2)
    clf.fit(X)
    y_pred = clf.predict(X)
    SSE_jlst = []
    for j in range(i):
        SSE_j = 0
        idx = np.array(y_pred == j)
        for xj in X[idx]:
            se = np.linalg.norm((xj - clf.centers[j]), ord = 2)
            SSE_j += se
        SSE_jlst.append(SSE_j)
    SSE.append(sum(SSE_jlst) / i)
    y_preds.append(y_pred)
```

```
Early Stopped at Iteration 5
Early Stopped at Iteration 9
Early Stopped at Iteration 13
Early Stopped at Iteration 18
Early Stopped at Iteration 18
Early Stopped at Iteration 17
Early Stopped at Iteration 22
Early Stopped at Iteration 22
```

After training our model, let's plot SSE against k. Pay attention to the trend and see if you can find a tipping point. A tipping point sometimes indicates a balance point for our model; increasing k furthur would lead to overfitting.

In [21]:
```
x = range(2, 10)
plt.plot(x, SSE)
plt.xlabel("k, number of clusters")
plt.ylabel("SSE per k")
plt.title('Sum of Squared Errors vs. k')
```

Out[21]:
```
Text(0.5, 1.0, 'Sum of Squared Errors vs. k')
```



Question 5: Can you find the most reasonable k based on this metric? Recall that we should have 4 clusters in this dataset. With that in mind, can you completely trust some metrics that help you determine how many clusters to use? What are the takeaways from this part of the experiment?

Answer: You can not determine a reasonable k based soley on this metric since while we want to reduce the loss of the model, we also have to take into account how many clusters should exist within the model. Having too many clusters could cause overfitting of the model which reduces the SSE which is why we shouldn't just take loss into account. We could also analyse the tipping point which occurs around 5 where K values diminish past 5. From these two pieces of information, we can assume the most reasonable k is most likely 5. While we could go ahead with this number, it is also best to think intuitvely about the number of clusters to use. We see 4 distinct clusters within the dataset while the most optimal is computed to be 5. It would be up to the user if there should be 4 or 5 clusters.

Question 6: In designing our model, we simplified some steps to make this implementation process easier. Can you list some potential improvements that can possibly make our model better when encountering data with noises or outliers, data with different types of distributions, or data where clusters have various distances of separation? (Hint: consider how we can work on initialization, assigning centers, data processing within the model, etc to make it better).

Answer: We could improve upon the initalization step when determing centroids by using more advanced algorithms that look at spread of data before creating centroids so that the chosen centroids are more applicable to the data. We could also improve upon the data processsing of the model by normalizing the data first and addressing the problem of outliers. We could also use better distancing metrics by choosing something other than Euclidean or Manhatten such as angular distance.

6 (20 points) Exponential Mixture Model

Suppose there exists a distribution called **Exponential Distribution** whose probability density function is described as:

$$\text{Exp}(\beta)\text{: } f(x;\beta) = \begin{cases} \beta e^{-\beta x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

where β is the parameter.

Suppose $X \in \mathbb{R}^+$ is sampled from a mixture of K clusters and each cluster Z_i is characterized by an Exponential Distribution with a parameter β_{Z_i}. The corresponding cluster priors are given as

$$P(Z_i) = \Pi_i \text{ for } i = 1, 2, \ldots, K$$

For example, given $K = 2$, $P(Z_1) = \Pi_1, P(Z_2) = \Pi_2$, X is sampled from a mixture of $\text{Exp}(\beta_{Z_1})$ and $\text{Exp}(\beta_{Z_2})$.

1. Given $K = 3$ and $\beta_{Z_1} = 1$, $\beta_{Z_2} = 2$, $\beta_{Z_3} = 4$, what is $P(Z_1|X = 1)$? **(5 points)**
2. Describe the E-step. Write an equation for each value that is computed. **(15 points)**

$\beta e^{-\beta x}$

1. $P(Z_1|x=1) = \dfrac{P(x=1|Z_1) \cdot P(Z_1)}{P(x=1)}$

$P(x=1|Z_1) = e^{-1}$

$P(x=2|Z_2) = 2e^{-2}$

$P(x=3|Z_3) = 4e^{-4}$

$P(x=1) = \Pi_1 e^{-1} + 2\Pi_2 e^{-2} + 4\Pi_3$

$\dfrac{e^{-1}\Pi_1}{\Pi_1 e^{-1} + 2\Pi_2 e^{-2} + 4\Pi_3 e^{-4}}$

2. The e-step is used to estimate the values that are missing from the dataset. updates the variables for each cluster for each iteration

$P(x_j|Z_i) = \beta e^{-\beta x}$

$P(Z_i|x=j) = \dfrac{\beta_i e^{-\beta_i} \cdot \Pi_i}{\sum_{i=1}^{n} P(x=j|z_i) \cdot P(z_i)}$

$P(Z_i) = \Pi_i$

$P(x_j) = \sum_{i=1}^{n} P(x=j|Z_i) \cdot P(Z_i)$