Machine Learning, Spring 2020

Project Three – Clustering

Python tutorial: http://learnpython.org/

TensorFlow tutorial: https://www.tensorflow.org/tutorials/

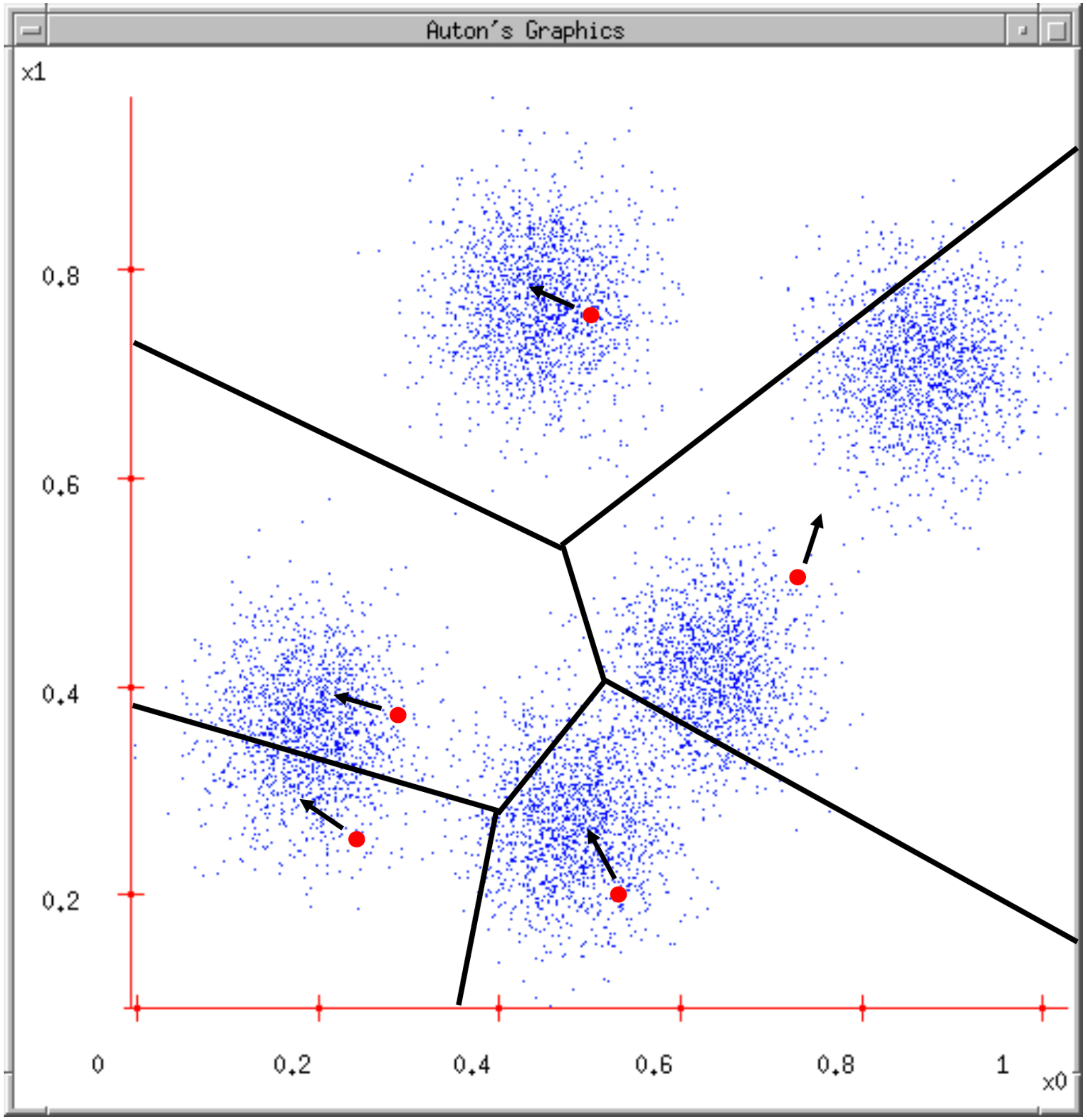
PyTorch tutorial: https://pytorch.org/tutorials/

Prof. Yi Fang (yfang@nyu.edu)

K-means

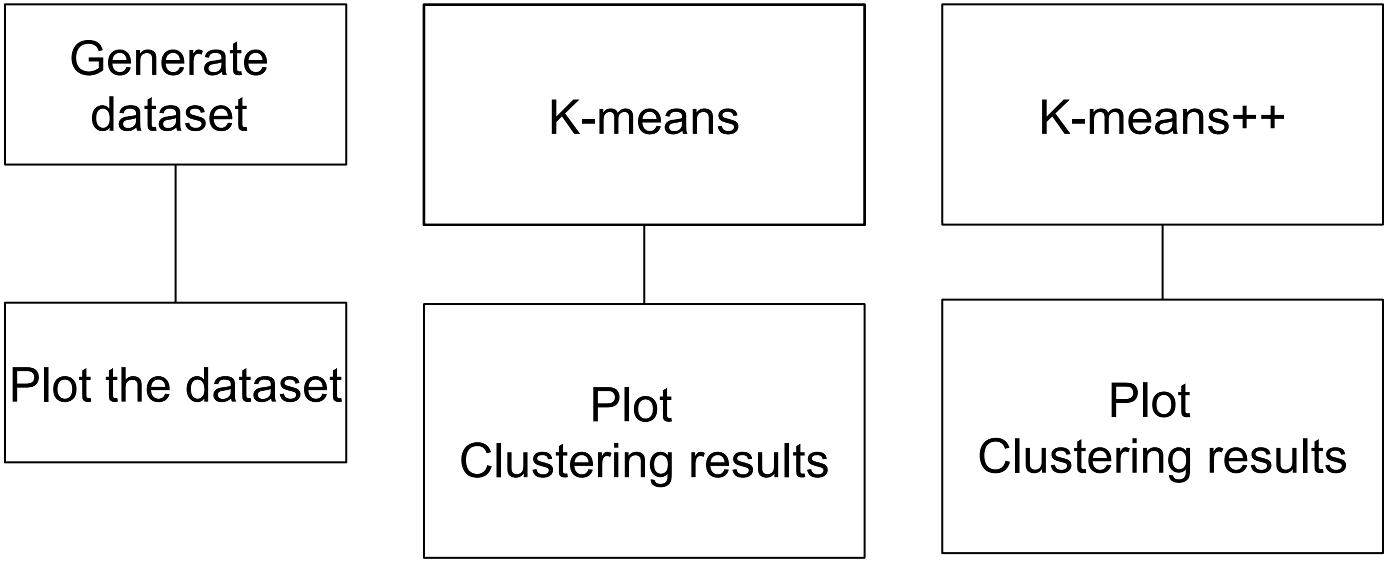
1. Ask user how many clusters they'd like. (e.g. k = 5)
2. Randomly guess k cluster center locations
3. Each datapoint finds out which center it's closest to.
4. Each center finds the centroid of the points it owns, and moves there.

Repeat steps 3-4 until convergence!

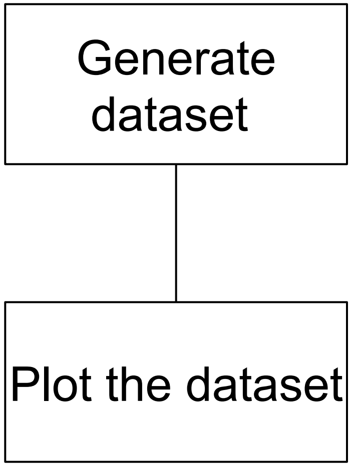


Thanks to Andrew Moore for providing this example.

Main Modules for Clustering



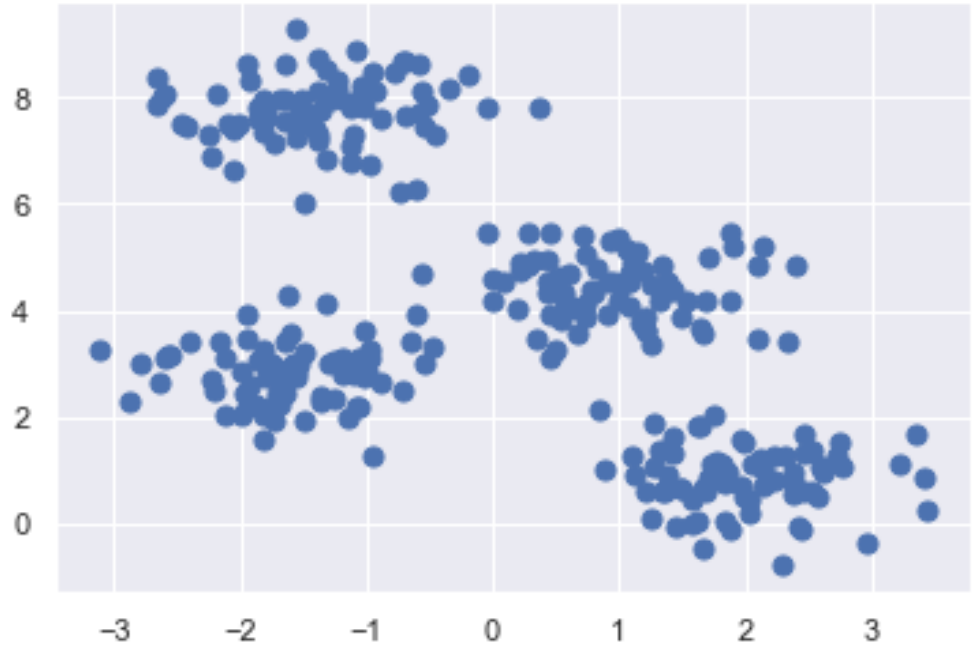
Main Modules for Clustering



In [2]:
```
# manually generate dataset

# use the make_blobs() function with n_samples=300,centers=4,cluster_std=0.6 and random_state=0
# store the return value to X and y_true
# plot the dataset using plt.scatter()
```

Example result:



Main Modules for Clustering

K-means

Write the function k_means(X, k, rseed) that:

- Randomly select *k* points from X with rseed as initial centers
- Assign points in X to closest center and update the centers until convergence
- return final centers and cluster label for each point in X

In [5]:
```
# def k_means(X, n_clusters, rseed=2):
    # 1. Randomly choose clusters
#     using np.random.RandomState first to set the seed and store it to a variable r
#     using r.permutation(data shape) to choose first k data point index as initial center.
#     store the center to a list.
#     repeat until convergence:
#         Assign labels based on closest center using pairwise_distances_argmin()
#         Find new centers from means of points:
#             Update centroid of each cluster to be the average(mean) of examples assigned to cluster k
#         check for convergence:
#             convergence if old center is new center
#     return the centers and labels
```

Main Modules for Clustering

K-means

Plot
Clustering results

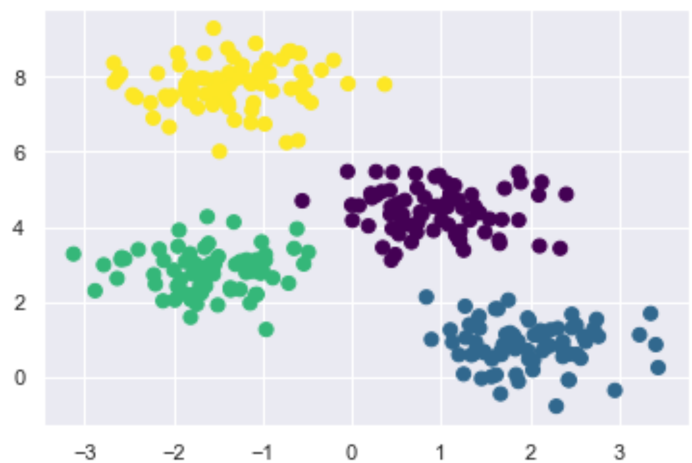
Write the function k_means(X, k, rseed) that:

- Randomly select *k* points from X with rseed as initial center
- Assign points in X to closest center and update the centers until convergence
- return final centers and cluster label for each point in X

```
In [ ]: # fit our function to the data set with the starting point rseed=0.
        # plot the figures
```

```
In [6]: # fit our function to the data set with the starting point rseed=2.
        # plot the figure
```

Example result:



Main Modules for Clustering

K-means++

Write the function k_meanspp(X, k, rseed) that:

- Randomly select ***a*** point from X with rseed as initial center
- Repeat until all ***k*** centers have been found
  - For each point in X, calculate the distance to closest center we found so far, then calculate the probability
  - Randomly choose a new center based on probability
- Run k-means with selected centers as initialization

❑Algorithm k-means++

$\mu_1 = \mathbf{x}^{(j)}$ for j chosen uniformly at random *// randomly initialize first point*

for k''=2 **to** k **do**

$$d_j = \min_{k'<k''} \left\| \boldsymbol{x}^{(j)} - \boldsymbol{\mu}_{k'} \right\|, \forall j$$ *// compute distances*

$$p_j = \frac{d_j^2}{\sum_{i=1}^{m} d_i^2}, \forall j$$ *// normalize to probability distribution*

j = random chosen with probability p_j

$\mu_{k''} = \mathbf{x}^{(j)}$

run k-means using μ as initial centers

Try to find a point far away from all the other centers as a new center

Main Modules for Clustering

K-means++

Write the function k_meanspp(X, k, rseed) that:

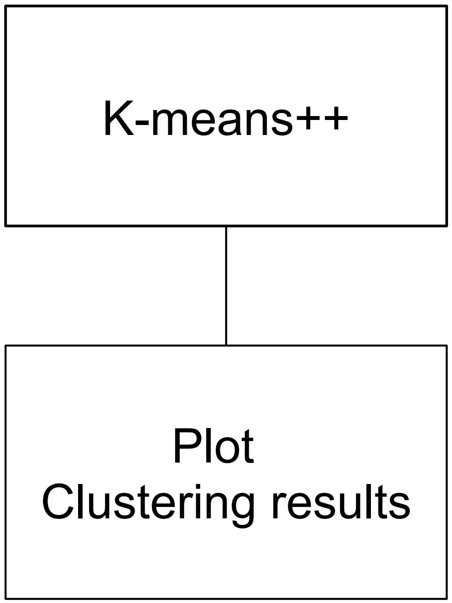
- Randomly select ***a*** point from X with rseed as initial center
- Repeat until all ***k*** centers have been found
    - For each point in X, calculate the distance to closest center we found so far, then calculate the probability
    - Randomly choose a new center based on probability
- Run k-means with selected centers as initialization

In [8]:
```
# def eucl_dist(a, b, axis=1):
#     def the function that calculate the L2 distance

# def the init function for kmean++:
#
# def init_center(k,X,rseed):
#     create a empty list store centers
#     random choose a center:
#         random choose a index:
#         using np.random.RandomState first to set the seed and store it to a variable r
#         using r.permutation(data shape) to choose first data point index as initial center.
#     append this center to the center list
#     while the length of the list less than k:
#         calculate dj for all data point: dj=min(||x^j-c_k||) whiere dj store the distance to the cloest center
#         calculate pj: pj=dj^2/sum_all(d^2) for all data point
#         random choose j using the probability:
#             using np.random.choice()
#         set the new center to be x^j
#         append the new center to center list
#     return all centers
```

In [9]:
```
# def the kmean++:
# def k_meanspp(X, n_clusters):
#     first init centers
#     then, run the k-means with the initialized centers.
```

Main Modules for Clustering



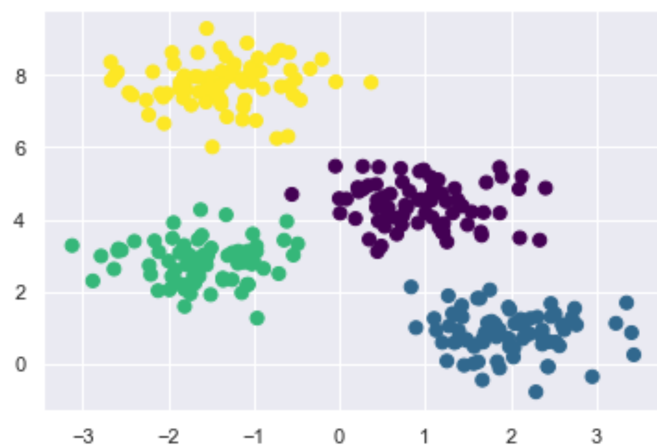
Write the function k_meanspp(X, k, rseed) that:

- Randomly select ***a*** point from X with rseed as initial center
- Repeat until all ***k*** centers have been found
  - For each point in X, calculate the distance to closest center we found so far, then calculate the probability
  - Randomly choose a new center based on probability
- Run k-means with selected centers as initialization

```
In [11]: # fit our kmean++ function to the data set with rseed=0.
         # plot the figure
```

```
In [10]: # fit our kmean++ function to the data set with rseed=2.
         # plot the figure
```

Example result:



Pipeline for Clustering