CS 412 Introduction to Machine Learning

KMeans– Code Tutorial

Instructor: Wei Tang

Department of Computer Science
University of Illinois at Chicago
Chicago IL 60607

https://tangw.people.uic.edu
tangw@uic.edu

Data

```
X1 = np.random.multivariate_normal([1,1], np.eye(2)*0.02, 100)
X2 = np.random.multivariate_normal([1,2], np.eye(2)*0.02, 100)
X3 = np.random.multivariate_normal([2,1], np.eye(2)*0.02, 100)
X = np.concatenate((X1,X2,X3), axis=0)
```

numpy.random.multivariate_normal

random.multivariate_normal(*mean, cov, size=None, check_valid='warn', tol=1e-8*)

Parameters: **mean** : *1-D array_like, of length N*

Mean of the N-dimensional distribution.

cov : *2-D array_like, of shape (N, N)*

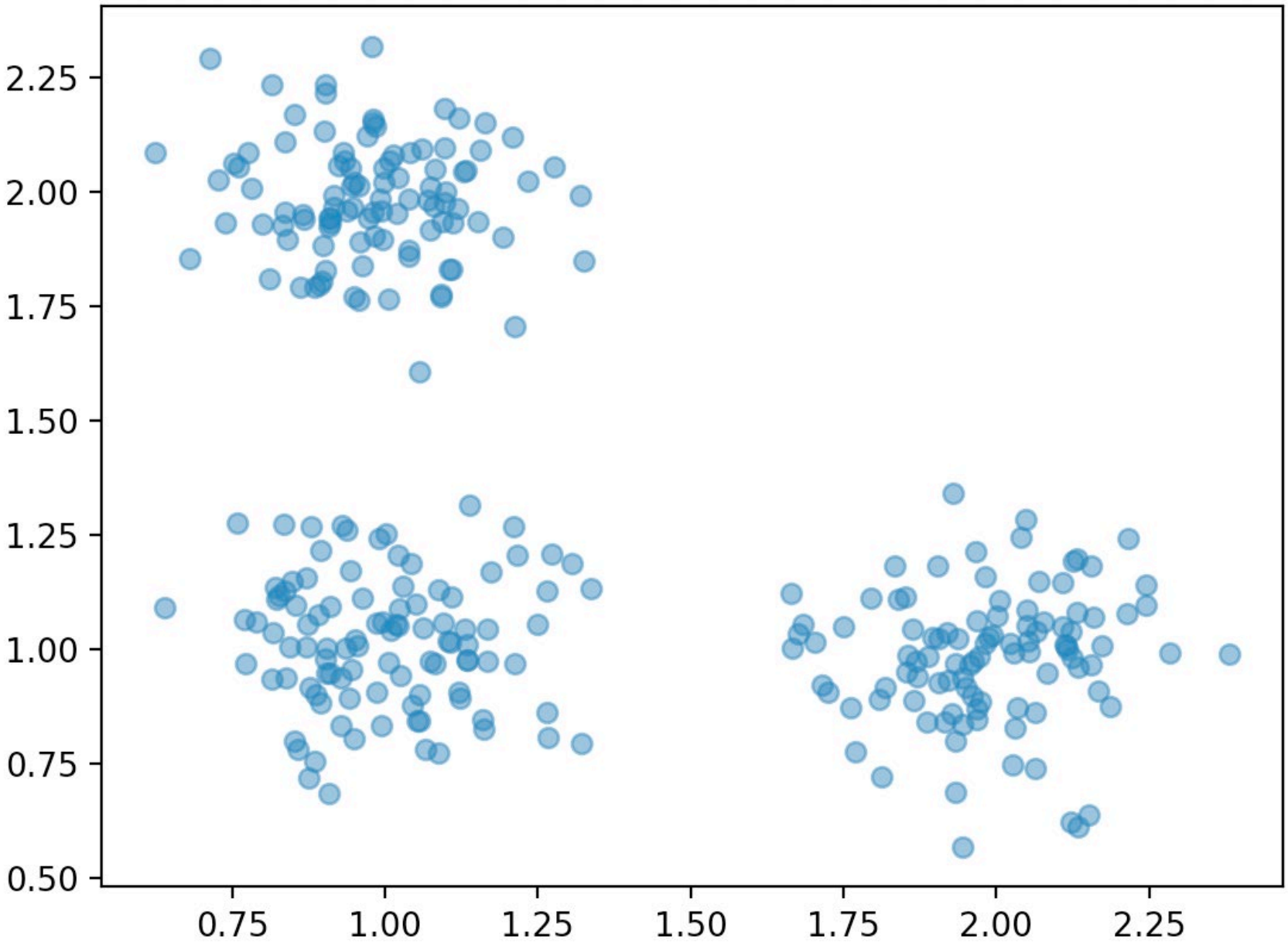
Covariance matrix of the distribution. It must be symmetric and positive-semidefinite for proper sampling.

size : *int or tuple of ints, optional*

Given a shape of, for example, `(m,n,k)`, `m*n*k` samples are generated, and packed in an *m*-by-*n*-by-*k* arrangement. Because each sample is *N*-dimensional, the output shape is `(m,n,k,N)`. If no shape is specified, a single (*N*-D) sample is returned.

Returns: **out** : *ndarray*

The drawn samples, of shape *size*, if that was provided. If not, the shape is `(N,)`. In other words, each entry `out[i,j,...,:]` is an N-dimensional value drawn from the distribution.


2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.75
1.00
1.25
1.50
1.75
2.00
2.25

Useful functions

```
def initialize_clusters(points, k):
    res = points[np.random.randint(points.shape[0], size=k)]
    return res

def get_distances(centroid, points):
    res = np.linalg.norm(points - centroid, axis=1)
    return res
```

numpy.random.**randint**(*low*, *high=None*, *size=None*, *dtype='l'*)¶

Return random integers from *low* (inclusive) to *high* (exclusive).

Return random integers from the “discrete uniform” distribution of the specified dtype in the “half-open” interval [*low*, *high*). If *high* is None (the default), then results are from [0, *low*).

Learning

```
def learning(X, k=3, maxiter=50):
    centroids = initialize_clusters(X, k)

    classes = np.zeros(X.shape[0], dtype=np.float64)
    distances = np.zeros([X.shape[0], k], dtype=np.float64)

    for i in range(maxiter):
        for i, c in enumerate(centroids):
            distances[:, i] = get_distances(c, X)

        classes = np.argmin(distances, axis=1)

        for c in range(k):
            centroids[c] = np.mean(X[classes == c], 0)
    return classes, centroids
```

Entire pipeline

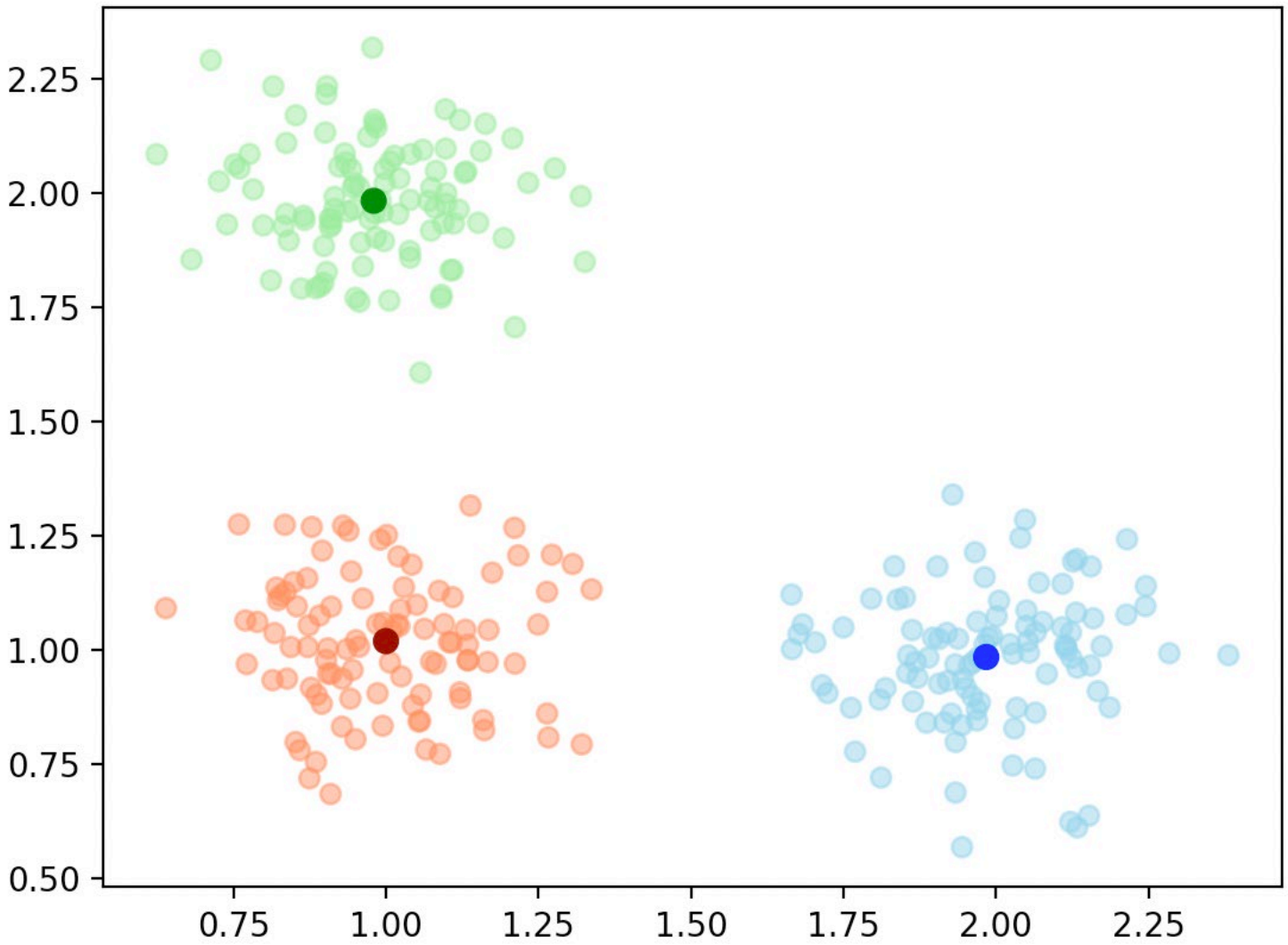
```
np.random.seed(0)
X1 = np.random.multivariate_normal([1,1], np.eye(2)*0.02, 100)
X2 = np.random.multivariate_normal([1,2], np.eye(2)*0.02, 100)
X3 = np.random.multivariate_normal([2,1], np.eye(2)*0.02, 100)
X = np.concatenate((X1,X2,X3), axis=0)

plt.scatter(X[:,0], X[:,1], alpha=0.5)
plt.show()

classes, centroids = learning(X)

group_colors = ['skyblue', 'coral', 'lightgreen']
colors = [group_colors[j] for j in classes]

plt.scatter(X[:,0], X[:,1], color=colors, alpha=0.5)
plt.scatter(centroids[:,0], centroids[:,1], color=['blue', 'darkred', 'green'], marker='o', lw=2)
plt.show()
```


2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.75
1.00
1.25
1.50
1.75
2.00
2.25