# Dokumentacja
# Mean Shift Algorytm

Autor: Mateusz Salamon

Kierunek: Elektronika 4

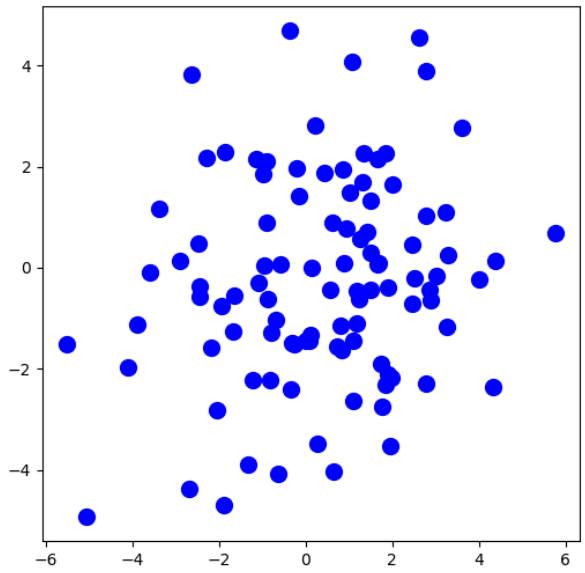
Przedmiot: Programowanie w językach skryptowych – Python i Linux Bash

1. Cluster Danych do przetwarzania
2. Funkcje matematyczne używane w Algorytmic Mean Shift
3. Algorytm Mean Shift
4. Tworzenie wykresów
5. Algorytm Mean Shift z użyciem sklearn
6. Testowanie błędów pomiędzy metodami

1. Cluster Danych

```
centers = [[1, 1], [-1, -1], [1, -1]]
# centers = 3
X_tab, Y_tab = make_blobs(n_samples=100, n_features=2, centers=centers,
cluster_std=1.9)
print(X_tab.shape)
plt.plot(X_tab[:, 0], X_tab[:, 1], 'bo', markersize=10)
# plt.show() # shows graph of original data cluster
```

Tworzenie clustra danych  o 100 punktach i odchyleniu standardowym równym 1.9.
Centra są przypisane w trzech punktach.
Odkomentowanie plt.show pokazuje wykres punktów.




Ten zbiór (cluster) poddamy działaniu algorytmu Mean Shift.

```
def eucl_dist(qi,pi):    # Euclidean Distance
    eudist = np.sqrt(np.sum((qi-pi)**2))
    return eudist
```

Funkcja zwracająca odległość euklidesową pomiędzy punktami q i p.

```
def neigh_points(X, x, dist = 5):  # Distance from neighbouring points
    Y = []
    for i in X:
        d = eucl_dist(i, x)
```

```
        if d <= dist:
            Y.append(i)
    return Y
```

Funkcja zwracająca tablicę punktów w odległości określonej przez parametr dist.

2. Funkcje matematyczne

```
def gaussian_krnl(x, sigma):     # Gaussian Kernel
    ret = (1/(sigma*math.sqrt(2*math.pi))) * np.exp(-0.5*(x / sigma)**2)
    return ret
```

Zwraca parametr definiujący jądro gaussowskie potrzebne do algorytmu Mean Shift.

```
search_distance = 6
sigma = estimate_bandwidth(X_tab, quantile=0.2, n_samples=500)
```

Search distance określa wielkość okręgu szukającego.
Sigma określa odchylenie standardowe.

```
X = np.copy(X_tab)  # X is an array copy of original points
Y = []
n = 5    # Number of iterations
```

3. Algorytm Mean shift

Tablice potrzebne do dalszej części algorytmu.
Parametr n określa ilość iteracji wykonania algorytmu.

```
for k in range(n):
    for i, x in enumerate(X):
        # Find closest points for each point
        points = neigh_points(X, x, search_distance)

        # Calculate Mean Shift / m_of_x
        numr = 0    # numerator
        denomr = 0   # denominator
        for xi in points:
            distance = eucl_dist(xi, x)
            weight = gaussian_krnl(distance, sigma)
            numr += (weight * xi)
            denomr += weight
        m_of_x = numr / denomr
        # update table
        X[i] = m_of_x
    # append X to new table
    Y.append(np.copy(X))
```

Pętla wykonująca algorytm Mean Shift.

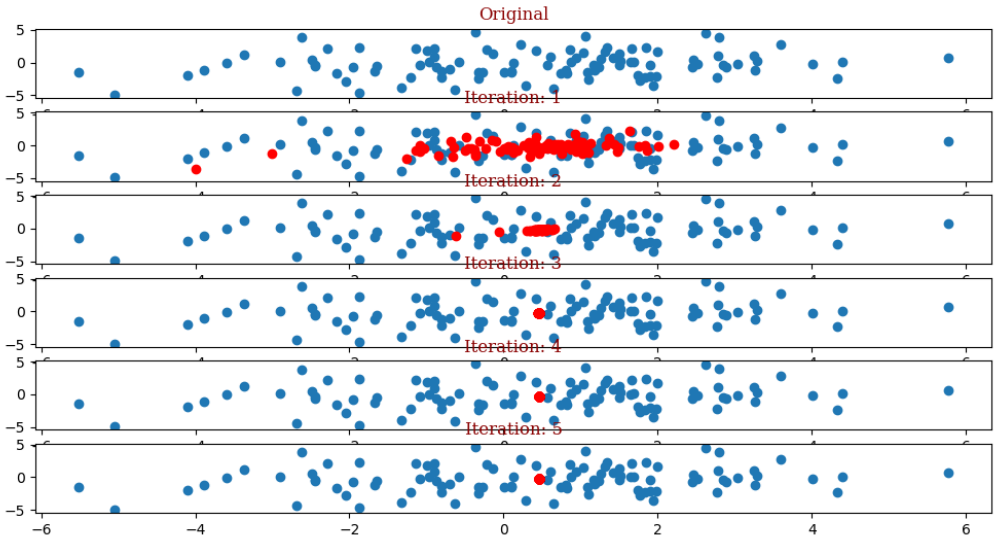
$$m(x) = \frac{\sum_{x_i \in N(x)} K(x_i - x) x_i}{\sum_{x_i \in N(x)} K(x_i - x)}$$

X[i] = m_of_x Dopisuje wartości do tablicy używanej podczas plotowania.

Parametr font zmienia wygląd czcionki pokazywanej na wykresie.

```
font = {'family': 'serif',
        'color':  'darkred',
        'weight': 'normal',
        'size': 12,
        }
```

4. Tworzenie wykresów iteracyjnych



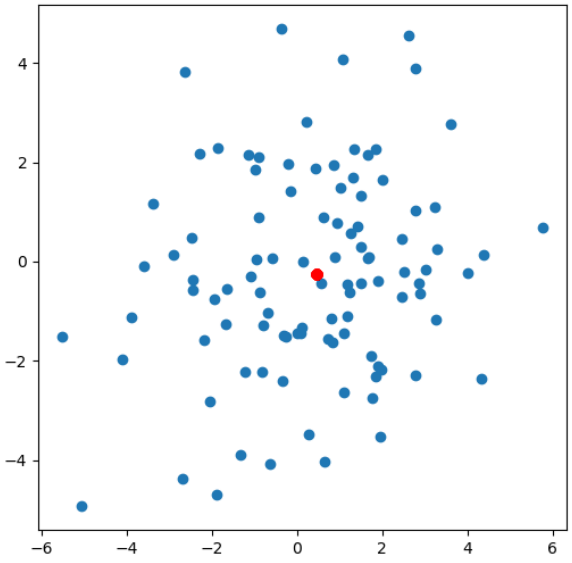

Tworzy subplot z wykresem oryginalnych próbek, które poddawane są operacjom algorytmu.

```
figure = plt.figure(1)
figure.set_size_inches((12, 9))
plt.subplot(n + 2, 1, 1)
plt.title(label='Original', fontdict=font)
plt.plot(X_tab[:,0], X_tab[:,1], 'o')
```

Tworzy pozostałe subploty, których ilość bazuje na ilości iteracji

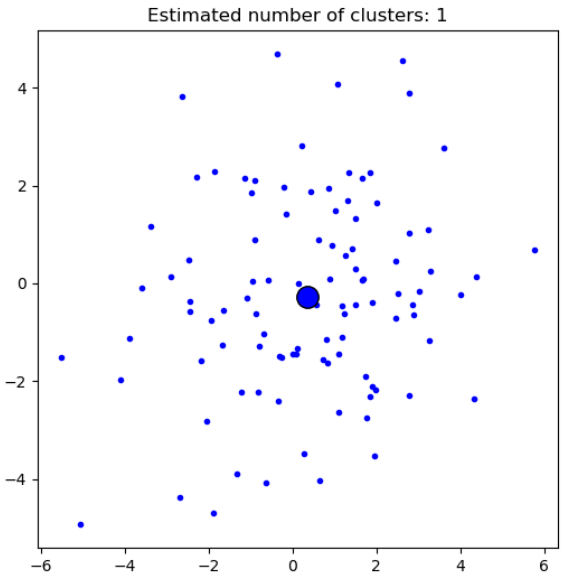
```
for i in range(n):
    index = i + 2
    plt.subplot(n + 2, 1, index)
    plt.title(label='Iteration: %d' % (index - 1), fontdict=font)
    plt.plot(X_tab[:,0], X_tab[:,1], 'o')
    plt.plot(Y[i][:,0], Y[i][:,1], 'ro')
plt.show()
```

Plot pokazujujący ostateczną wersję iterowanego clustera



```
plt.plot(X_tab[:,0], X_tab[:,1], 'o')
plt.plot(Y[i][:,0], Y[i][:,1], 'ro')
print("centroid iteration",Y[n-1][0,0],Y[n-1][0,1])
plt.show()
```

## 5. Algorytm Mean Shift z użyciem sklearn




```
def sklearn_mean_shift(X_tab):   #sklearn faster method using developed functions
    import numpy as np
    from sklearn.cluster import MeanShift, estimate_bandwidth
    from sklearn.datasets.samples_generator import make_blobs
```

```
    # Generate sample data
    # centers = [[1, 1], [-1, -1], [1, -1]]
    # X, _ = make_blobs(n_samples=100, n_features= 2 , centers=centers,
cluster_std=1.9)

    # Compute clustering with MeanShift

    # The following bandwidth can be automatically detected using
    # bandwidth = estimate_bandwidth(X_tab, quantile=0.2, n_samples=500)

    ms = MeanShift(bandwidth=100, bin_seeding=True)
    ms.fit(X_tab)
    labels = ms.labels_
    cluster_centers = ms.cluster_centers_

    labels_unique = np.unique(labels)
    n_clusters_ = len(labels_unique)
```

Tworzenie wykresów do metody z wykorzystaniem biblioteki sklearn.

```
     plt.figure(1)
    plt.clf()

    colors = cycle('bgrcmykbgrcmykbgrcmykbgrcmyk')
    for k, col in zip(range(n_clusters_), colors):
        my_members = labels == k
        cluster_center = cluster_centers[k]
        plt.plot(X_tab[my_members, 0], X_tab[my_members, 1], col + '.')
        plt.plot(cluster_center[0], cluster_center[1], 'o', markerfacecolor=col,
                 markeredgecolor='k', markersize=14)
    plt.title('Estimated number of clusters: %d' % n_clusters_)
    plt.show()
    print("centroid sklearn", cluster_center[0], cluster_center[1])
    a = cluster_center[0]
    b = cluster_center[1]
    return a, b     # returns final coordinates of sklearn cluster center
```

## 6. Testowanie błędów pomiędzy metodami.

```
# Errors between two methods
c = Y[n-1][0, 0]               # x coordinate of iteration cluster center
d = Y[n-1][0, 1]               # y coordinate of iteration cluster center
error = [abs(a-c),abs(b-d)]             # error table
print(" x_err: ",error[0],"\n", "y_err: ", error[1])
```

```
centroid iteration 0.33980269297602267 -0.21103091484020978
centroid sklearn 0.2869611986056376 -0.187245361590433
 x_err:  0.052841494370385045
 y_err:  0.02378555324977677
done

Process finished with exit code 0
```