Density estimation for a Gaussian mixture

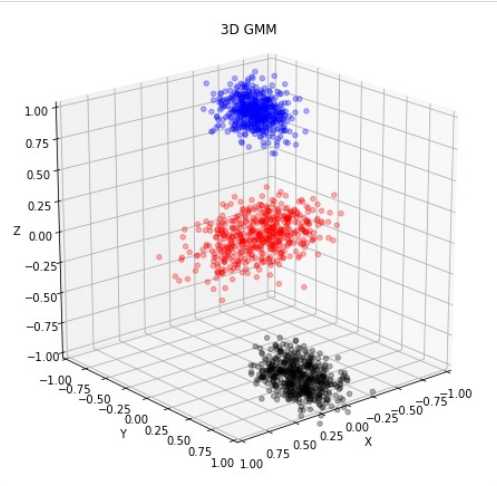
- Antoniadis Pablo
- Llobregat Thomas
- Ye Sebastien

Imports and visualisation tools

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.colors as colors
from matplotlib.patches import Ellipse
import imageio
from PIL import Image


def visualize_3d_points(points,n_gaussians, export=True):
    '''
    plots points and their corresponding gmm model in 3D
    Input:
        points: N X 3, sampled points
    Output:
        None
    '''
    N = int(np.round(points.shape[0] / n_gaussians))
    # Visualize data
    fig = plt.figure(figsize=(8, 8))
    axes = fig.add_subplot(111, projection='3d')
    axes.set_xlim([-1, 1])
    axes.set_ylim([-1, 1])
    axes.set_zlim([-1, 1])
    plt.set_cmap('Set1')
    colors = ['blue','red','black','purple','slategrey','limegreen','peru','pink']
    for i in range(n_gaussians):
        idx = range(i * N, (i + 1) * N)
        axes.scatter(points[idx, 0], points[idx, 1], points[idx, 2], alpha=0.3, c=colors[i])

    plt.title('3D GMM')
    axes.set_xlabel('X')
    axes.set_ylabel('Y')
    axes.set_zlabel('Z')
    axes.view_init(20, 50)

    plt.show()
```

Generating 500 random points in a 3d space

In [ ]:
```
## Generate synthetic data
N,D = 500, 3 # number of points and dimenstinality

# set gaussian centers and covariances in 3D
mu = np.array([[0.0, 0.0, 1.0],
                  [0.0, 0.0, 0.0],
                  [0.0, 0.5, -1.0]])

sigma = np.array([np.diag([0.01, 0.03, 0.01]),
                  np.diag([0.08, 0.01, 0.02]),
                  np.diag([0.01, 0.05, 0.01])])

n_gaussians = mu.shape[0]

points = []
for i in range(len(mu)):
    x = np.random.multivariate_normal(mu[i], sigma[i], N )
    points.append(x)
points = np.concatenate(points)

#visualize
visualize_3d_points(points,n_gaussians)
```



In [ ]:
```
class GMM:
  def __init__(self, X, n_clusters, n_epochs):
    self.clusters = []
    self.X = X
    self.n_clusters = n_clusters
    self.j = self.X.shape[0]
    self.likelihoods = np.zeros((n_epochs, ))
    self.scores = np.zeros((self.X.shape[0], self.n_clusters))
    self.n_epochs = n_epochs

    for i in range(n_clusters):
        self.clusters.append({
            'phi_j': 1.0 / self.n_clusters,
```

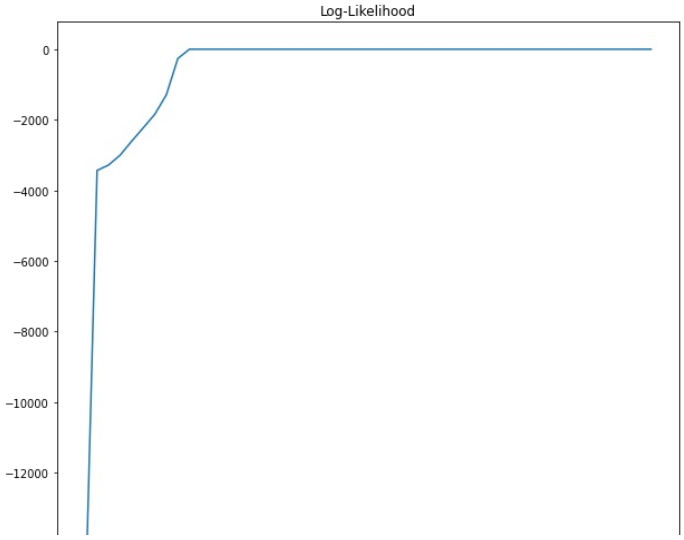
```
            'mu_j': np.random.rand(3),
            'sigma_j': np.identity(X.shape[1], dtype=np.float64)
            })

  @staticmethod
  def gaussian(X, mu, sigma):
    n = X.shape[1]
    diff = (X - mu).T
    return np.diagonal(1 / ((2 * np.pi) ** (n / 2) * np.linalg.det(sigma) ** 0.5) * np.exp(-0.5 * np.dot(np.dot(diff.T, np.linalg.inv(sigma)), diff))).reshape(-1,

  def _e_step(self):
    totals = np.zeros((self.X.shape[0], 1), dtype=np.float64)

    for cluster in self.clusters:
        phi_j = cluster['phi_j']
        mu_j = cluster['mu_j']
        sigma_j = cluster['sigma_j']

        # gamma = P(x(i)|z(i)=j)P(z(i)=j)
        w_j_numerator = (phi_j * GMM.gaussian(self.X, mu_j, sigma_j)).astype(np.float64)

        for i in range(self.j):
            totals[i] += w_j_numerator[i]

        cluster['w_j'] = w_j_numerator
        cluster['totals'] = totals

    for cluster in self.clusters:
        cluster['w_j'] /= cluster['totals']

  def _m_step(self):
    for cluster in self.clusters:
        w_j = cluster['w_j']
        sigma_j = np.zeros((self.X.shape[1], self.X.shape[1]))

        sum_w_j = np.sum(w_j, axis=0)

        phi_j = sum_w_j / self.j
        mu_j = np.sum(w_j * self.X, axis=0) / sum_w_j

        for j in range(self.j):
          diff = (self.X[j] - mu_j).reshape(-1, 1)
          sigma_j += w_j[j] * np.dot(diff, diff.T)

        sigma_j /= sum_w_j

        cluster['phi_j'] = phi_j
        cluster['mu_j'] = mu_j
        cluster['sigma_j'] = sigma_j


  def _get_likelihood(self):
      likelihood = []
      sample_likelihoods = np.log(np.array([cluster['totals'] for cluster in self.clusters]))
      return np.sum(sample_likelihoods), sample_likelihoods

  def train(self, stop=0):
    for i in range(self.n_epochs):
        self._e_step()
        self._m_step()
        likelihood, sample_likelihoods = self._get_likelihood()
        self.likelihoods[i] = likelihood
        print('Epoch: ', i + 1, 'Likelihood: ', likelihood)

        if abs(likelihood) < stop:
          break

    for i, cluster in enumerate(self.clusters):
      self.scores[:, i] = np.log(cluster['w_j']).reshape(-1)

  def plot_likelihood_history(self):
    plt.figure(figsize=(10, 10))
    plt.title('Log-Likelihood')
    plt.plot(np.arange(1, self.n_epochs + 1), self.likelihoods)
    plt.show()
```

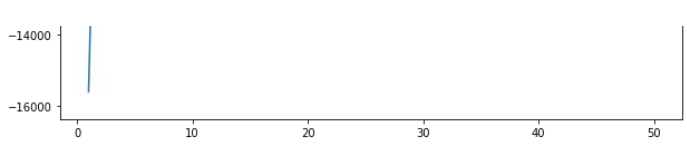
In [ ]:
```
clf = GMM(points, 3, 50)
```

In [ ]:
```
clf.train(500)
```

```
Epoch:  1 Likelihood:  -15599.12257184734
Epoch:  2 Likelihood:  -3436.109669304922
Epoch:  3 Likelihood:  -3281.410710804132
Epoch:  4 Likelihood:  -3001.2021137875345
Epoch:  5 Likelihood:  -2605.664878123571
Epoch:  6 Likelihood:  -2228.2901436499083
Epoch:  7 Likelihood:  -1843.9102043363464
Epoch:  8 Likelihood:  -1294.301441237485
Epoch:  9 Likelihood:  -262.4212101096116
```

In [ ]:
```
clf.plot_likelihood_history()
```




−14000
−16000
0
10
20
30
40
50