Gaussian Mixture Model

03/09/2020

VIDA GHARAVIAN

- A Gaussian mixture model (GMM) attempts to find a mixture of multi-dimensional Gaussian probability distributions that best model any input dataset.

n_components

weights_init defult 'kmeans' or random

covariance matrix

$$p(\vec{x}) = \sum_{i=1}^{K} \phi_i \mathcal{N}(\vec{x} \mid \vec{\mu}_i, \Sigma_i)$$

$$\mathcal{N}(\vec{x} \mid \vec{\mu}_i, \Sigma_i) = \frac{1}{\sqrt{(2\pi)^K |\Sigma_i|}} \exp\left(-\frac{1}{2}(\vec{x} - \vec{\mu}_i)^{\mathrm{T}} {\Sigma_i}^{-1} (\vec{x} - \vec{\mu}_i)\right)$$

$$\sum_{i=1}^{K} \phi_i = 1$$

mean

the total probability distribution normalizes to 1.

-

Parameters: **n_components*int, defaults to 1.***

The number of mixture components.

covariance_type*{'full' (default), 'tied', 'diag', 'spherical'}*

String describing the type of covariance parameters to use. Must be one of:

'full'
each component has its own general covariance matrix

'tied'
all components share the same general covariance matrix

'diag'
each component has its own diagonal covariance matrix

'spherical'
each component has its own single variance

tol*float, defaults to 1e-3.*

The convergence threshold. EM iterations will stop when the lower bound average gain is below this threshold.

reg_covar*float, defaults to 1e-6.*

Non-negative regularization added to the diagonal of covariance. Allows to assure that the covariance matrices are all positive.

max_iter*int, defaults to 100.*

The number of EM iterations to perform.

n_init*int, defaults to 1.*

The number of initializations to perform. The best results are kept.

init_params*{'kmeans', 'random'}, defaults to 'kmeans'.*

The method used to initialize the weights, the means and the precisions. Must be one of:

```
'kmeans' : responsibilities are initialized using kmeans.
'random' : responsibilities are initialized randomly.
```

weights_init*array-like, shape (n_components,), optional*

The user-provided initial weights, defaults to None. If it None, weights are initialized using the `init_params` method.

means_init*array-like, shape (n_components, n_features), optional*

The user-provided initial means, defaults to None, If it None, means are initialized using the `init_params` method.

precisions_init*array-like, optional.*

The user-provided initial precisions (inverse of the covariance matrices), defaults to None. If it None, precisions are initialized using the 'init_params' method. The shape depends on 'covariance_type':

```
(n_components,)                        if 'spherical',
(n_features, n_features)               if 'tied',
(n_components, n_features)             if 'diag',
(n_components, n_features, n_features) if 'full'
```

random_state*int, RandomState instance or None, optional (default=None)*

If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If

None, the random number generator is the RandomState instance used by `np.random`.

warm_start*bool, default to False.*
If 'warm_start' is True, the solution of the last fitting is used as initialization for the next call of fit(). This can speed up convergence when fit is called several times on similar problems. In that case, 'n_init' is ignored and only a single initialization occurs upon the first call. See the Glossary.

verbose*int, default to 0.*
Enable verbose output. If 1 then it prints the current initialization and each iteration step. If greater than 1 then it prints also the log probability and the time needed for each step.

verbose_interval*int, default to 10.*
Number of iteration done before the next print

-
    - Get input
        - ```
          def read_file(file_name: str) -> np.ndarray:
              img = Image.open(file_name, 'r')
              pix_val = list(img.getdata())
              return np.asarray(pix_val, dtype=np.float32)
          ```
    - Gaussian Mixture Model with 5 components
        - ```
          def gmm(X: np.ndarray) -> GaussianMixture:
              gmm = GaussianMixture(n_components=15)
              gmm.fit(X)
              return gmm
          ```
    - Threshold filtering
        - ```
          def threshold_filtering(gmm: GaussianMixture, X: np.ndarray, img: Image):
              threshold = np.mean(gmm.means_)
              new_image = []
              for x in range(img.size[1]):
                  new_image_row = []
                  for i in range(img.size[0]):
                      new_image_rgb = []
                      for j in range(0, X.shape[1], 1):
                          if X[x * (img.size[0]) + i][j] > threshold:
                              new_image_rgb.append(X[x * (img.size[0]) +i][j])
                          else:
          ```
          ```

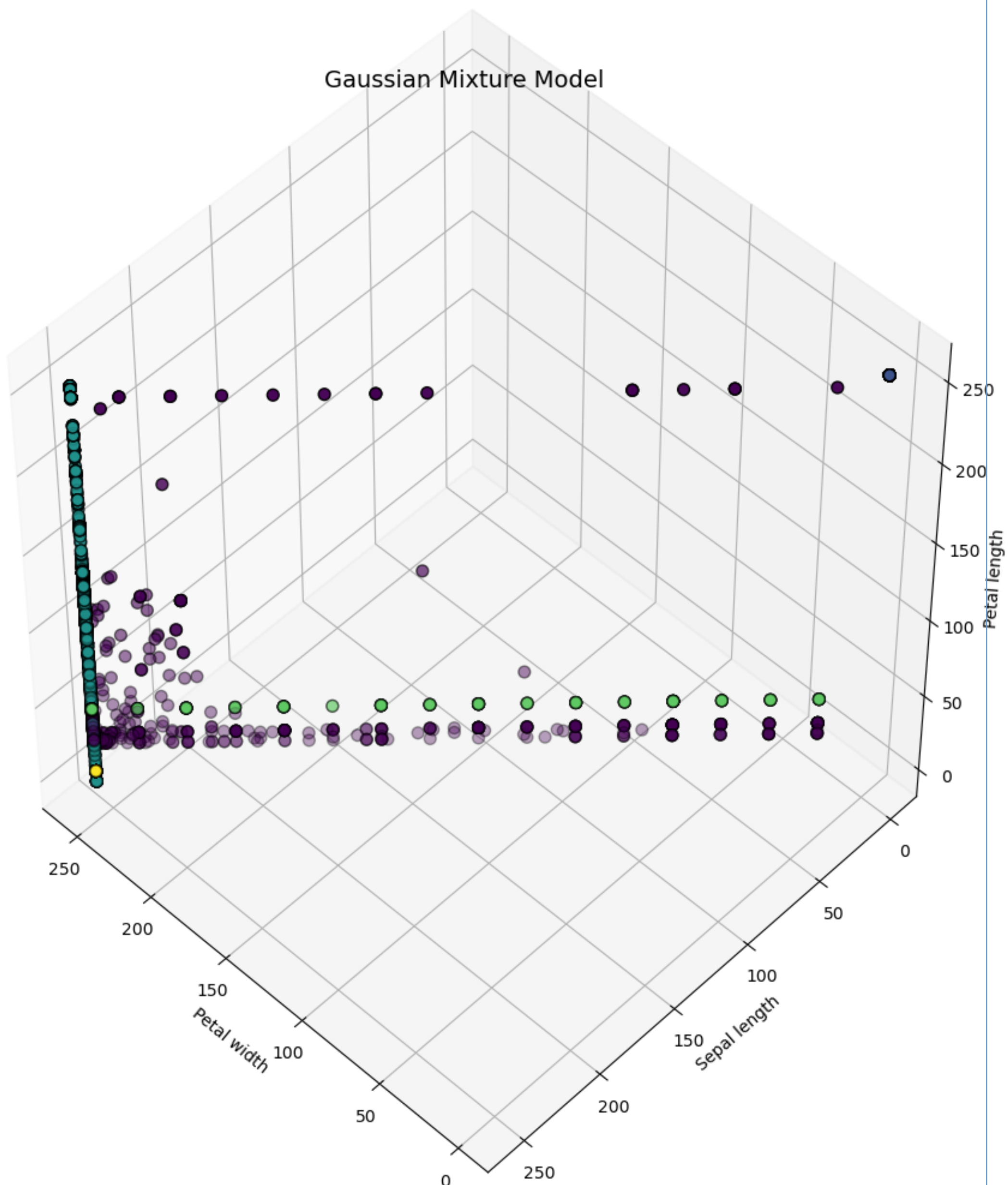
```
                    new_image_rgb.append(255)
                new_image_row.append(new_image_rgb)
            new_image.append(new_image_row)
        new_image = np.asarray(new_image, dtype=np.uint8)
        new_image = Image.fromarray(new_image, 'RGB')
        new_image.save('my.jpg')
        new_image.show()
```

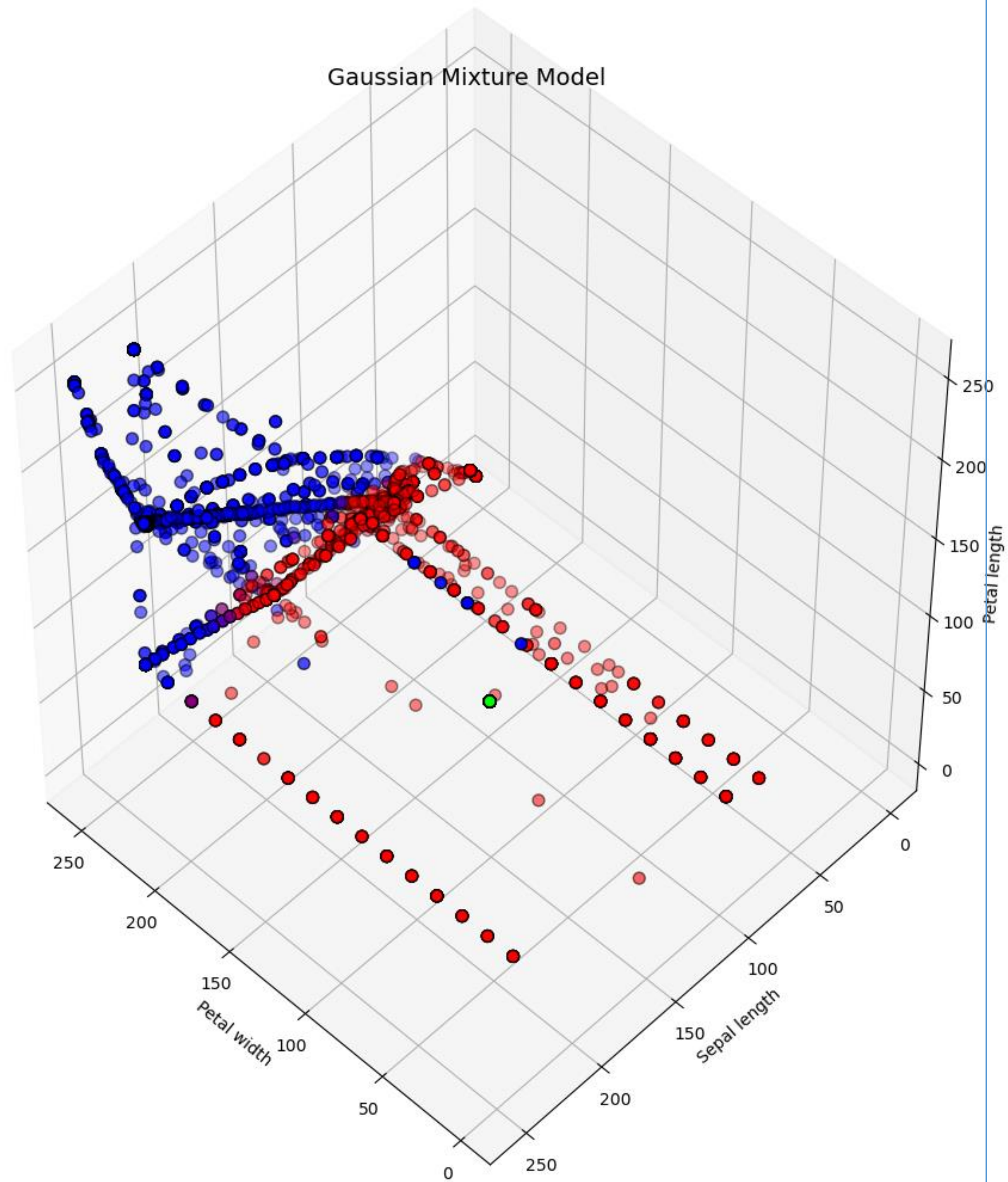
- Plotting

```
labels = gmm.predict(X)
fig = plt.figure(1, figsize=(10,10))
ax = Axes3D(fig, rect=[0, 0, 0.95, 1], elev=48, azim=134)
ax.scatter(X[:, 3], X[:, 3], X[:, 2],
           c=labels, edgecolor="k", s=50)
ax.set_xlabel("Petal width")
ax.set_ylabel("Sepal length")
ax.set_zlabel("Petal length")
plt.title("Gaussian Mixture Model", fontsize=14)
plt.show()
```

- input:

- Out put:


Gaussian Mixture Model
Petal length
Petal width
Sepal length
0
50
100
150
200
250


Gaussian Mixture Model
250
200
150
100
50
0
Petal length
250
200
150
100
50
0
Petal width
0
50
100
150
200
250
Sepal length