mle_for_quadratic_discriminant _analysis

May 17, 2020

1 Maximum Likelihood Estimation For Quadratic Discriminant Analysis

Assume that there is a discrete generative model. A classification is to be made using this model. The probability density function of the output to be in the class c is given as follows:

$$p\left(y=c|\mathbf{x},\boldsymbol{\theta}\right)=\frac{p\left(\mathbf{x}|y=c,\boldsymbol{\theta}\right)p\left(y=c|\boldsymbol{\theta}\right)}{p\left(\mathbf{x}|\boldsymbol{\theta}\right)}$$

If $p\left(\mathbf{x}|y=c,\boldsymbol{\theta}\right)$ is given by a multivariate Gaussian distribution, then this classification is called the quadratic discriminant classification. The maximum likelihood estimator for the quadratic discriminant classifier is to be derived. Hence, the likelihood of the data should be written first. The likelihood of a single data sample $\left(\mathbf{x}_i, y_i\right)$ is as follows:

$$p\left(\mathbf{x}_i,y_i|\boldsymbol{\theta}\right)=p\left(\mathbf{x}_i|y_i,\boldsymbol{\theta}\right)p\left(y_i|\boldsymbol{\theta}\right)=\prod_{c=1}^{C}p\left(y_i|\boldsymbol{\theta}_c\right)^{I(y_i=c)}\prod_{c=1}^{C}p\left(\mathbf{x}_i|y_i,\boldsymbol{\theta}_c\right)^{I(y_i=c)}$$

Assuming that the samples are independent and there are N training samples, the likelihood of the data can now be written as follows:

$$p\left(D|\boldsymbol{\theta}\right)=\prod_{i=1}^{N}p\left(\mathbf{x}_i,y_i|\boldsymbol{\theta}\right)$$

For the sake of avoiding any numerical underflow and computational simplicity, the logarithm of the above expression is taken to obtain the log-likelihood:

$$\log p\left(D|\boldsymbol{\theta}\right)=\sum_{i=1}^{N}\log p\left(\mathbf{x}_i,y_i|\boldsymbol{\theta}\right)=\sum_{i=1}^{N}\log\left[\prod_{c=1}^{C}p\left(y_i|\boldsymbol{\theta}_c\right)^{I(y_i=c)}\prod_{c=1}^{C}p\left(\mathbf{x}_i|y_i,\boldsymbol{\theta}_c\right)^{I(y_i=c)}\right]=$$

$$\sum_{i=1}^{N}\left[\sum_{c=1}^{C}\log\left[p\left(y_i|\boldsymbol{\theta}_c\right)^{I(y_i=c)}\right]+\sum_{c=1}^{C}\log\left[p\left(\mathbf{x}_i|y_i,\boldsymbol{\theta}_c\right)^{I(y_i=c)}\right]\right]=\sum_{i=1}^{N}\left[\sum_{c=1}^{C}I\left(y_i=c\right)\log p\left(y_i|\boldsymbol{\theta}_c\right)+\sum_{c=1}^{C}I\left(y_i=c\right)\log p\left(\right.$$

$$\log p\left(D|\boldsymbol{\theta}\right)=\sum_{c=1}^{C}N_c\log p\left(y_i=c|\boldsymbol{\theta}_c\right)+\sum_{c=1}^{C}\sum_{i:y_i=c}\log p\left(\mathbf{x}_i|y_i,\boldsymbol{\theta}_c\right)$$

The first term is maximized if the maximum likelihood estimator (MLE) for the class occurrences which is

$$\pi_{c_{MLE}}=\frac{N_c}{N}$$

is plugged in place of $p(y_i = c|\boldsymbol{\theta}_c)$. N_c is the number of occurrences of the class c. The second term is maximized if the maximum likelihood estimators for the means and covariances of the multivariate Gaussian distributions for the probability densities of the input features conditioned on the class label are used. The MLE's for the mean and covariances are as follows:

$$\boldsymbol{\mu}_{c_{MLE}} = \frac{1}{N_c} \sum_{i:y_i=c} x_i$$

$$\boldsymbol{\Sigma}_{c_{MLE}} = \frac{1}{N_c} \sum_{i:y_i=c} \left(\mathbf{x}_i - \boldsymbol{\mu}_{c_{MLE}}\right) \left(\mathbf{x}_i - \boldsymbol{\mu}_{c_{MLE}}\right)^T$$

The maximum likelihood estimation measure for $p(y = c|\mathbf{x}, \boldsymbol{\theta})$ is calculated without computing $p(\mathbf{x}|\boldsymbol{\theta})$ since it is the same for all of the classes.

[10]:
```
import numpy as np
from scipy.stats import multivariate_normal
```

[11]:
```
class CategoryGaussianData:
    def __init__(self, dist_mean, nonsingular_matrix, dist_cov_eigvalues,␣
↪num_of_samples):
        # 1-d numpy array for the mean of the distribution
        self.dist_mean = dist_mean
        # a nonsingular matrix to create the eigenvector matrix of the␣
↪covariance matrix for the distribution
        self.nonsingular_matrix = nonsingular_matrix
        # 1-d array with eigenvalues to be used for the generation of the␣
↪covariance matrix for the distribution
        self.dist_cov_eigvalues = dist_cov_eigvalues
        # number of samples from the distribution
        self.num_of_samples = num_of_samples

    def compute_cov_matrix(self):
        nonsingular_matrix = self.nonsingular_matrix
        q, r = np.linalg.qr(nonsingular_matrix)
        # normalize the eigenvector matrix of the covariance matrix for the␣
↪distribution
        for i in range(0, q.ndim):
            q[i, :] = q[i, :] / np.linalg.norm(q[i, :])
        eigenvalue_matrix = np.diag(self.dist_cov_eigvalues)
        dist_cov_matrix = np.matmul(np.matmul(q, eigenvalue_matrix), q.T)

        return dist_cov_matrix


    def sample_data(self):
        dist_cov_matrix = self.compute_cov_matrix()

        return np.random.default_rng().multivariate_normal(mean=self.dist_mean,␣
↪cov=dist_cov_matrix, size=self.num_of_samples)
```

```
class MLEComputation:
    def __init__(self, train_input, pi_class_mle):
        # input data for training
        self.train_input = train_input
        # mle for class density
        self.pi_class_mle = pi_class_mle

    def gaussian_mle_estimates(self):
        mean_mle = np.mean(self.train_input, axis=0)
        num_of_samples = self.train_input.shape[0]
        num_of_features = self.train_input.shape[1]
        cov_mle = np.zeros((num_of_features, num_of_features))
        for i in range(0, num_of_samples):
            centered_sample = self.train_input[i, :]-mean_mle
            cov_mle = cov_mle + np.outer(centered_sample, centered_sample)
        cov_mle = cov_mle/num_of_samples

        return mean_mle, cov_mle

    def compute_class_density_measure(self, x):
        mean_mle, cov_mle = self.gaussian_mle_estimates()
        return (multivariate_normal.pdf(x, mean=mean_mle, cov=cov_mle))*self.
 ↪pi_class_mle
```

[12]:
```
mean_0 = np.array([1, 2, 3, 4])
print('the mean of the category 0 is:')
print(mean_0)
non_singular_0 = np.array([[1, 2, 0, 0], [2, -1, 0, 0], [0, 4, 3, 0], [0, 0, 2.
 ↪5, 5]])
print('nonsingular matrix for generating the eigenvector matrix of the␣
 ↪covariance of the category 0 is:')
print(non_singular_0)
Lambda_0 = np.array([1, 2.4, 3, 3.8])
print('eigenvalues of the covariance matrix of the distribution for the␣
 ↪category 0:')
print(Lambda_0)
num_of_samples_0 = 1000
category_0 = CategoryGaussianData(mean_0, non_singular_0, Lambda_0,␣
 ↪num_of_samples_0)
cov_0 = category_0.compute_cov_matrix()
print('covariance matrix of the category 0')
print(cov_0)
print('verification for the eigenvalues of the covariance matrix of the␣
 ↪category 0')
print(np.linalg.eigvalsh(cov_0))
```

```
the mean of the category 0 is:
[1 2 3 4]
nonsingular matrix for generating the eigenvector matrix of the covariance of
the category 0 is:
[[ 1.   2.   0.   0. ]
 [ 2.  -1.   0.   0. ]
 [ 0.   4.   3.   0. ]
 [ 0.   0.   2.5  5. ]]
eigenvalues of the covariance matrix of the distribution for the category 0:
[1.  2.4 3.  3.8]
covariance matrix of the category 0
[[ 2.84883485 -0.92441743 -0.45552178  0.27234043]
 [-0.92441743  1.46220871  0.22776089 -0.13617021]
 [-0.45552178  0.22776089  2.68470111 -0.17021277]
 [ 0.27234043 -0.13617021 -0.17021277  3.20425532]]
verification for the eigenvalues of the covariance matrix of the category 0
[1.  2.4 3.  3.8]
```

[13]:
```
mean_1 = np.array([5, 6, 7, 8])
print('the mean of the category 1 is:')
print(mean_1)
non_singular_1 = np.array([[1, 2, 0, 0], [2, -1, 0, 0], [0, 4, 3, 0], [0, 0, 2.
 ↪5, 5]])
print('nonsingular matrix for generating the eigenvector matrix of the␣
 ↪covariance of the category 1 is:')
print(non_singular_1)
Lambda_1 = np.array([1.5, 2.8, 3.3, 4.6])
print('eigenvalues of the covariance matrix of the distribution for the␣
 ↪category 1:')
print(Lambda_1)
num_of_samples_1 = 1000
category_1 = CategoryGaussianData(mean_1, non_singular_1, Lambda_1,␣
 ↪num_of_samples_1)
cov_1 = category_1.compute_cov_matrix()
print('covariance matrix of the category 1:')
print(cov_1)
print('verification for the eigenvalues of the covariance matrix of the␣
 ↪category 1:')
print(np.linalg.eigvalsh(cov_1))
```

```
the mean of the category 1 is:
[5 6 7 8]
nonsingular matrix for generating the eigenvector matrix of the covariance of
the category 1 is:
[[ 1.   2.   0.   0. ]
 [ 2.  -1.   0.   0. ]
 [ 0.   4.   3.   0. ]
 [ 0.   0.   2.5  5. ]]
eigenvalues of the covariance matrix of the distribution for the category 1:
[1.5 2.8 3.3 4.6]
covariance matrix of the category 1:
[[ 3.43483283 -0.96741641 -0.55927052  0.44255319]
 [-0.96741641  1.98370821  0.27963526 -0.2212766 ]
 [-0.55927052  0.27963526  3.14954407 -0.27659574]
 [ 0.44255319 -0.2212766  -0.27659574  3.63191489]]
verification for the eigenvalues of the covariance matrix of the category 1:
[1.5 2.8 3.3 4.6]
```

[14]:
```
train_data_0 = category_0.sample_data()
pi_c_0 = 1000 / 2000
mle_0 = MLEComputation(train_data_0, pi_c_0)
mean_mle, cov_mle = mle_0.gaussian_mle_estimates()
print('mle estimate of the mean for the distribution of category 0:')
print(mean_mle)
print('mle estimate of the covariance for the distribution of category 0:')
print(cov_mle)
print('eigenvalues of the mle of the covariance matrix for the distribution of␣
 ↪category 0:')
print(np.linalg.eigvalsh(cov_mle))
```

```
mle estimate of the mean for the distribution of category 0:
[0.93586769 1.99695486 3.00596426 3.88525981]
mle estimate of the covariance for the distribution of category 0:
[[ 2.5987325  -0.83071122 -0.40746526  0.19787551]
 [-0.83071122  1.49596924  0.1671663  -0.13427102]
 [-0.40746526  0.1671663   2.72224439 -0.12514498]
 [ 0.19787551 -0.13427102 -0.12514498  3.16768647]]
eigenvalues of the mle of the covariance matrix for the distribution of category
0:
[1.04865571 2.41582932 2.97908351 3.54106406]
```

[15]:
```
train_data_1 = category_1.sample_data()
pi_c_1 = 1000 / 2000
mle_1 = MLEComputation(train_data_1, pi_c_1)
mean_mle, cov_mle = mle_1.gaussian_mle_estimates()
print('mle estimate of the mean for the distribution of category 1:')
print(mean_mle)
print('mle estimate of the covariance for the distribution of category 1:')
print(cov_mle)
print('eigenvalues of the mle of the covariance matrix for the distribution of␣
 ↪category 1:')
print(np.linalg.eigvalsh(cov_mle))
```

```
mle estimate of the mean for the distribution of category 1:
[4.98037281 6.00810522 7.06844913 7.88405738]
```

```
mle estimate of the covariance for the distribution of category 1:
[[ 3.26917221 -1.01059087 -0.49481094  0.34243099]
 [-1.01059087  2.00005137  0.27328701 -0.15942071]
 [-0.49481094  0.27328701  3.1465712  -0.28339761]
 [ 0.34243099 -0.15942071 -0.28339761  3.51020651]]
eigenvalues of the mle of the covariance matrix for the distribution of category
1:
[1.44096938 2.82008397 3.2699866  4.39496134]
```

[16]:
```
test_input = np.array([3, 4, 5, 6])
print('mle of the class density measure for the class 0:', mle_0.
 ↪compute_class_density_measure(test_input))
print('mle of the class density measure for the class 1:', mle_1.
 ↪compute_class_density_measure(test_input))
```

```
mle of the class density measure for the class 0: 1.1009810712693366e-05
mle of the class density measure for the class 1: 3.206432917902719e-05
```

From the above two results, it is observed that the mle's for the probability density measures of the two classes are close to each other for the test input $[3, 4, 5, 6]$. This test input is the half-way between the means of the distributions for the two classes.

[17]:
```
mle_0 = MLEComputation(train_data_0, pi_c_0)
mle_1 = MLEComputation(train_data_1, pi_c_1)
test_input = np.array([1, 2, 3, 4])
print('mle of the class density measure for the class 0:', mle_0.
 ↪compute_class_density_measure(test_input))
print('mle of the class density measure for the class 1:', mle_1.
 ↪compute_class_density_measure(test_input))
```

```
mle of the class density measure for the class 0: 0.002442610780499476
mle of the class density measure for the class 1: 2.2131149491416957e-10
```

For the test input being equal to the mean of the distribution for the class 0, the above results are in compliance with the expectation that category 0 should be chosen by the mle.

[18]:
```
mle_0 = MLEComputation(train_data_0, pi_c_0)
mle_1 = MLEComputation(train_data_1, pi_c_1)
test_input = np.array([5, 6, 7, 8])
print('mle of the class density measure for the class 0:', mle_0.
 ↪compute_class_density_measure(test_input))
print('mle of the class density measure for the class 1:', mle_1.
 ↪compute_class_density_measure(test_input))
```

```
mle of the class density measure for the class 0: 1.4342942088844171e-12
mle of the class density measure for the class 1: 0.0016532044136216772
```

For the test input being equal to the mean of the distribution for the class 1, the above results are in compliance with the expectation that category 1 should be chosen by the mle.

2 References

Machine Learning A Probabilistic Perspective, Kevin P. Murphy