

CSC411 Fall 2017

Assignment 2 Report

Tianbao Li

2017/11/11

1 Q1: Class-conditional gaussians

Given:

$$p(y = k) = a_k \quad (1)$$

$$p(\mathbf{x}|y = k, \mu, \sigma) = \left(\prod_{i=1}^d 2\pi\sigma_i^2\right)^{-1/2} \exp\left\{-\sum_{i=1}^d \frac{1}{2\sigma_i^2} (x_i - \mu_{ki})^2\right\} \quad (2)$$

1.1 Bayes' rule derivation

$$\begin{aligned} p(y = k|\mathbf{x}, \mu, \sigma) &= \frac{p(\mathbf{x}|y = k, \mu, \sigma)p(y = k)}{p(\mathbf{x}|\mu, \sigma)} \\ &= \frac{p(\mathbf{x}|y = k, \mu, \sigma)p(y = k)}{\sum_{j=1}^K p(\mathbf{x}|y = j, \mu, \sigma)} \\ &= \frac{(\prod_{i=1}^d 2\pi\sigma_i^2)^{-1/2} \exp\{-\sum_{i=1}^d \frac{1}{2\sigma_i^2} (x_i - \mu_{ki})^2\} a_k}{\sum_{j=1}^K (\prod_{i=1}^d 2\pi\sigma_i^2)^{-1/2} \exp\{-\sum_{i=1}^d \frac{1}{2\sigma_i^2} (x_i - \mu_{ji})^2\}} \end{aligned}$$

1.2 Negative likelihood function (NLL)

$$\begin{aligned} \ell(\theta; D) &= -\log p(y^{(1)}, \mathbf{x}^{(1)}, y^{(2)}, \mathbf{x}^{(2)}, \dots, y^{(N)}, \mathbf{x}^{(N)}|\theta) \\ &= -\log \prod_{n=1}^N p(y^{(n)}, x^{(n)}|\theta) \\ &= -\sum_{n=1}^N (\log p(x^{(n)}|y^{(n)}, \theta) + \log p(y^{(n)}|\theta)) \\ &= -\sum_{n=1}^N (\log((\prod_{i=1}^d 2\pi\sigma_i^2)^{-1/2} \exp\{-\sum_{i=1}^d \frac{1}{2\sigma_i^2} (x_i^{(n)} - \mu_{ki})^2\}) + \log \alpha_k) \\ &= -\sum_{n=1}^N (-\frac{1}{2} \sum_{i=1}^d 2\pi\sigma_i^2 - \sum_{i=1}^d \frac{1}{2\sigma_i^2} (x_i^{(n)} - \mu_{ki})^2 + \log \alpha_k) \\ &= \sum_{n=1}^N (\sum_{i=1}^d \pi\sigma_i^2 + \sum_{i=1}^d \frac{1}{2\sigma_i^2} (x_i^{(n)} - \mu_{ki})^2 - \log \alpha_k) \end{aligned}$$

1.3 Partial derivatives

$$\begin{aligned}
\frac{\partial \ell}{\partial \mu_{ki}} &= \frac{\partial (\sum_{n=1}^N \sum_{i=1}^d \frac{1}{2\sigma_i^2} (x_i^{(n)} - \mu_{ki})^2)}{\partial \mu_{ki}} \\
&= - \sum_{n=1}^N \sum_{i=1}^d \frac{1}{\sigma_i^2} \mathbb{1}(y^{(n)} = k) (x_i^{(n)} - \mu_{ki}) \\
\frac{\partial \ell}{\partial \sigma_i^2} &= \frac{\partial (\sum_{n=1}^N (\sum_{i=1}^d \pi \sigma_i^2 + \sum_{i=1}^d \frac{1}{2\sigma_i^2} (x_i^{(n)} - \mu_{ki})^2))}{\partial \sigma_i^2} \\
&= \sum_{n=1}^N \sum_{i=1}^d \mathbb{1}(y^{(n)} = k) (2\pi - \frac{1}{2\sigma_i^4} (x_i^{(n)} - \mu_{ki})^2)
\end{aligned}$$

1.4 Estimation

$$\begin{aligned}
\frac{\partial \ell}{\partial \mu_{ki}} &= 0 \\
\Rightarrow \mu_{ki} &= \frac{\sum_{n=1}^N \sum_{i=1}^d \mathbb{1}(y^{(n)} = k) x_i^{(n)}}{\sum_{n=1}^N \sum_{i=1}^d \mathbb{1}(y^{(n)} = k)} \\
\frac{\partial \ell}{\partial \sigma_i^2} &= 0 \\
\Rightarrow \sigma_i &= \sqrt[4]{\frac{\sum_{n=1}^N \sum_{i=1}^d \mathbb{1}(y^{(n)} = k) * (x_i^{(n)} - \mu_{ki})^2}{\sum_{n=1}^N \sum_{i=1}^d \mathbb{1}(y^{(n)} = k) * 4\pi}}
\end{aligned}$$

2 Handwritten digit classification

In this assignment, we use the dataset of handwritten digit. To get a macro view of the data, here we plot the means for each data classes in the training set, shown in Figure 1.

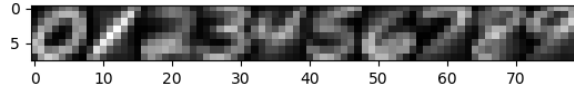


Figure 1: Mean for each class

2.1 K-NN classifier

First, we build a K nearest neighbor classifier using Euclidean distance on the data.

2.1.1 Accuracy for certain K values

- for K = 1
 - train classification accuracy: 1.0
 - test classification accuracy: 0.96875
- for K = 15

- train classification accuracy: 0.961
- test classification accuracy: 0.959

2.1.2 Ties solution

During the implementation, one important problem is that ties usually happens and need to be broken. The solution is shown as follows.

```
def query_knn(self, test_point, k):
    digit = None
    dist = self.l2_distance(test_point)
    noTies = False
    while noTies == False:
        mink = np.argpartition(dist, k)[:k]
        counts = np.bincount(self.train_labels[mink].astype(int))
        digit = np.argmax(counts == np.amax(counts))
        if len(digit) > 1:
            k = k + 1
        else:
            noTies = True
    return digit
```

While running KNN, if ties happen (which means two classes has the same maximum count), we try to decrease k by 1 and run the KNN again until no ties. As the class label should focus on one or a few classes, ties are unlikely to happen when $K = 1$, so this should be a solution to get an answer without ties.

2.1.3 Best k

Here we implement 10 fold cross validation in range 1-15 to find the optimal K. By finding minimum validation error, optimal K for this problem is that $K = 3$ and accuracies are shown as follows.

- train classification accuracy: 1.0
- validation classification accuracy: 0.96514286
- test classification accuracy: 0.96875

2.2 Conditional gaussian classifier training

2.2.1 Diagonal covariance

The diagonal elements of each covariance matrix Σ_k can be plotted as Figure 2.

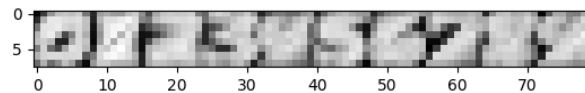


Figure 2: Diagonal elements of covariance for each class

2.2.2 Average conditional log-likelihood

The average conditional log-likelihood for each class is shown as follows.

- training data avg conditional likelihood: [-2.78902836 -2.48009327 -2.46864922 -2.53467722 -2.49894322 -2.93938866 -2.4646036 -3.70354979 -2.47281232 -2.74221317]
- test data avg conditional likelihood: [-3.22599621 -4.13098548 -2.44591828 -3.1675836 -2.65604537 -3.69867522 -3.67902811 -7.62636215 -2.81381993 -3.95651773]

2.2.3 Accuracy

By selecting the most likely posterior class for each datapoint as prediction, the accuracy is shown as follows.

- training accuracy: 0.981285714286
- test accuracy: 0.9605

2.2.4 Leading eigenvectors

The leading eigenvectors of each covariance matrix Σ_k can be plotted as Figure 3.

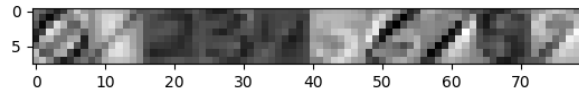


Figure 3: Leading eigenvectors of covariance for each class

2.3 Naive bayes classifier training

2.3.1 Convert to binart features

```
def binarize_data(pixel_values):  
    return np.where(pixel_values > 0.5, 1.0, 0.0)
```

2.3.2 Model fitting

For a Bernoulli Naive Bayes classifier using MAP, we need to calculate η first.

```
def compute_parameters(train_data, train_labels):  
    eta = np.zeros((10, 64))  
    K = eta.shape[0]  
    d = eta.shape[1]  
    for k in range(K):  
        k_index = np.argwhere(train_labels == k)  
        k_digits = train_data[k_index].reshape(-1, d)  
        for j in range(d):  
            eta[k][j] = (np.sum(k_digits[:, j]) + 1.0) \  
                / (k_digits.shape[0] + 2.0)  
    return eta
```

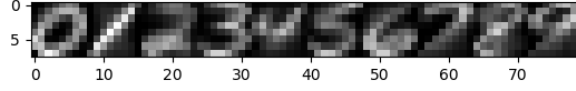


Figure 4: η for each class

2.3.3 Plot η

The parameters η is plotted in Figure 4.

2.3.4 Sample new data

Given the parameters η , we can sample new data points for each of the 10 digits. *numpy.random.binomial* method is used here. One new data point is like Figure 5.

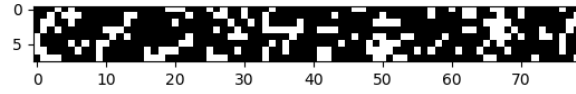


Figure 5: Sampled data point

2.3.5 Average conditional log-likelihood

The average conditional log-likelihood for each class is shown as follows.

- training data avg conditional likelihood: [-2.8909254 -3.92355812 -3.26626627 -3.10729808 -3.01401521 -3.12587265 -2.94008982 -3.17888068 -3.47393427 -3.54254905]
- test data avg conditional likelihood: [-3.10584429 -3.6500495 -3.3475608 -3.29021075 -3.04145573 -3.20550698 -3.06688846 -3.23938846 -3.44419131 -3.507459]

2.3.6 Accuracy

By selecting the most likely posterior class for each datapoint as prediction, the accuracy is shown as follows.

- training accuracy: 0.774142857143
- test accuracy: 0.76425

2.4 Model comparison

For the handwritten digit classification problem, we use models of KNN, conditional Gaussian and naive Bayes. Analyzing from the accuracy, KNN and conditional Gaussian works better, while Naive Bayes performs badly.

- KNN: Due to the relatively distinct classification among digits, decision boundaries are clear in some ways. So it is easy for KNN classifier to find the right group.
- conditional Gaussian: Because the pattern for each class seems fixed, so data points for a certain class gather around the mean. Such structure can be easily caught by Gaussian.

- naive Bayes: Naive Bayes has the assumption that all features should be i.i.d given a data point. However, as we can see from the figures that each feature corresponds to a pixel in the figure. For a digit, pixel connects with each other and form the shape of the digit. So the features here are not i.i.d. Naive Bayes could not work well upon obeying the assumption.