en-ch4.5_Naive_Bayes_Classiication

2019년 10월 8일

1 Chapter 4 - THE PRELIMINARIES: A CRASHCOURSE

1.1 4.5 Naive Bayes Classiication

```
In [1]: %matplotlib inline
        import d2l
        import math
        from mxnet import nd, gluon
        d2l.use_svg_display()
```

4.5.1 Optical Character Recognition

```
In [2]: def transform(data, label):
            return nd.floor(data/128).astype('float32').squeeze(axis=-1), label

        mnist_train = gluon.data.vision.MNIST(train=True, transform=transform)
        mnist_test = gluon.data.vision.MNIST(train=False, transform=transform)
```

```
In [3]: image, label = mnist_train[2]
        image.shape, label
```

```
Out[3]: ((28, 28), 4)
```

```
In [4]: image.shape, image.dtype
```

```
Out[4]: ((28, 28), numpy.float32)
```

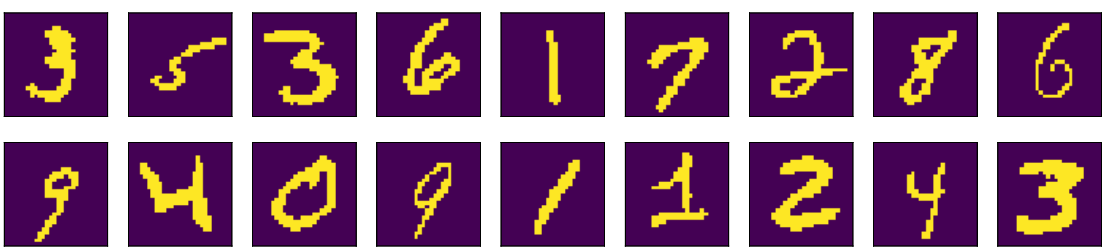
```
In [5]: label, type(label), label.dtype
```

```
Out[5]: (4, numpy.int32, dtype('int32'))
```

```
In [6]: images, labels = mnist_train[10:38]
        images.shape, labels.shape
```

```
Out[6]: ((28, 28, 28), (28,))
```

```
In [7]: # Save to the d2l package.
        def show_images(imgs, num_rows, num_cols, titles=None, scale=1.5):
            """Plot a list of images."""
            figsize = (num_cols * scale, num_rows * scale)
            _, axes = d2l.plt.subplots(num_rows, num_cols, figsize=figsize)
            axes = axes.flatten()
            for i, (ax, img) in enumerate(zip(axes, imgs)):
                ax.imshow(img.asnumpy())
                ax.axes.get_xaxis().set_visible(False)
                ax.axes.get_yaxis().set_visible(False)

                if titles:
                    ax.set_title(titles[i])
            return axes

        show_images(images, 2, 9);
```



4.5.2 The Probabilistic Model for Classiication If we are able to compute these probabilities, which are $p(y|\mathbf{x})$ for y = 0, . . . , 9 in our example, then the classifier will output the prediction y given by the expression:

$$\hat{y} = \operatorname{argmax}_y p(y|\mathbf{x})$$

4.5.3 The Naive Bayes Classiier

To begin, let's use Bayes Theorem, to express the classifier as

$$\hat{y} = \text{argmax}_y \frac{p(\mathbf{x}|y)p(y)}{p(\mathbf{x})}$$

argmax 이므로 $p(\mathbf{x})$ 는 모두 같음. 즉 $p(\mathbf{x}|y)p(y)$ 만 구하고 가장 큰 값을 취하면 됨. 그러므로 $p(\mathbf{x})$ 는 생략 가능

Now, using the chain rule of probability, we can express the term $p(\mathbf{x}|y)$ as

$$p(x_1|y) \cdot p(x_2|x_1, y) \cdot ... \cdot p(x_d|x_1, ..., x_{d-1}y)$$

$$\hat{y} = \text{argmax}_y \ = \prod_i p(x_i|y)p(y)$$

4.5.4 Training

```
In [8]: X, Y = mnist_train[:] # all training examples

        print(X.shape)
        print(Y.shape)
        print(Y)

        n_y = nd.zeros((10))

        for y in range(10):
            # print((Y==y).astype(float))
            # 0, 1, 2, 3 ... 9 각 class 의 total
            n_y[y] = (Y==y).sum()

        # y 의 total 로 y 의 각 class 를 나눔. 즉 각 y class 별 확률
        P_y = n_y / n_y.sum()
        P_y
```

```
(60000, 28, 28)
(60000,)
[5 0 4 ⋯ 5 6 8]
```

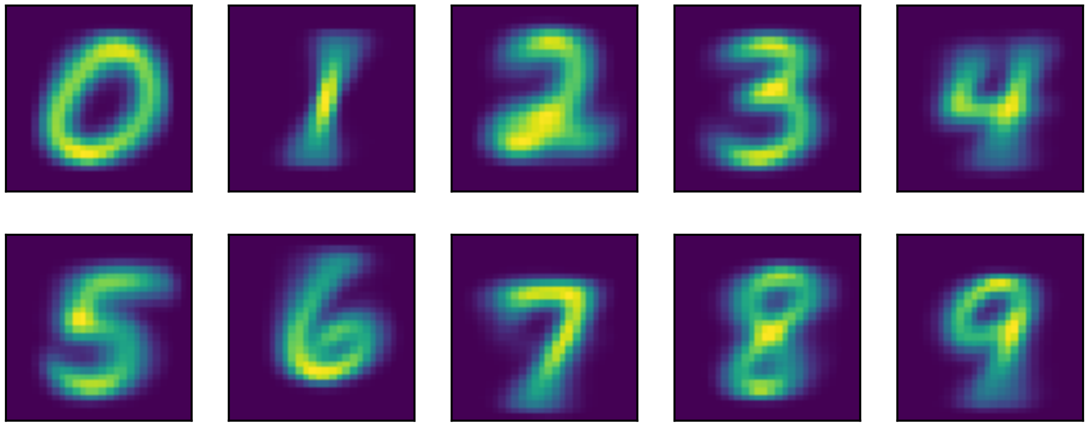
Out[8]:

```
[0.09871667 0.11236667 0.0993     0.10218333 0.09736667 0.09035
 0.09863333 0.10441667 0.09751666 0.09915   ]
<NDArray 10 @cpu(0)>
```

In [9]:

```python
n_x = nd.zeros((10, 28, 28))


a = nd.array([[1, 2, 3, 4], [5,6,7,8]])
b = nd.array([[1, 5, 6, 4], [9,10,11,8]])
print(a[a==b].sum(axis=0))

print((Y==y))
print(X.asnumpy()[2][1])

for y in range(10):
    #print(X.asnumpy()[Y==y][1])
    n_x[y] = nd.array(X.asnumpy()[Y==y].sum(axis=0))
    #print(n_x[y])

#print(n_x[9])
# class y 일때 각 픽셀 값들의 확률.
P_xy = (n_x+1) / (n_y+1).reshape((10,1,1))
print(P_xy.shape)
show_images(P_xy, 2, 5);
```

```
[[ 6.  8. 10. 12.]
 [ 2.  4.  6.  8.]
 [ 2.  4.  6.  8.]
 [10. 12. 14. 16.]]
<NDArray 4x4 @cpu(0)>
[False False False ... False False False]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0.]
(10, 28, 28)
```



In [ ]:

In [10]:
```
def bayes_pred(x):
    x = x.expand_dims(axis=0) # (28, 28) -> (1, 28, 28)
    print(P_xy[0][0])
    p_xy = P_xy * x #+ (1-P_xy)*(1-x) # ???

    p_xy = p_xy.reshape((10,-1)).prod(axis=1) # p(x/y)
    print(p_xy.shape)
    return p_xy * P_y

image, label = mnist_test[0]
bayes_pred(image)
```

```
[0.0001688 0.0001688 0.0001688 0.0001688 0.0001688 0.0001688 0.0001688
 0.0001688 0.0001688 0.0001688 0.0001688 0.0001688 0.0001688 0.0001688
 0.0001688 0.0001688 0.0001688 0.0001688 0.0001688 0.0001688 0.0001688
 0.0001688 0.0001688 0.0001688 0.0001688 0.0001688 0.0001688 0.0001688]
<NDArray 28 @cpu(0)>
(10,)
```

Out[10]:

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
<NDArray 10 @cpu(0)>
```

In [11]:

```python
a = 0.1
print('underflow:', a**784)
print('logrithm is normal:', 784*math.log(a))
```

```
underflow: 0.0
logrithm is normal: -1805.2267129073316
```

In [12]:

```python
log_P_xy = nd.log(P_xy)
log_P_xy_neg = nd.log(1-P_xy)
log_P_y = nd.log(P_y)

def bayes_pred_stable(x):
    x = x.expand_dims(axis=0) # (28, 28) -> (1, 28, 28)
    p_xy = log_P_xy * x + log_P_xy_neg * (1-x)
    p_xy = p_xy.reshape((10,-1)).sum(axis=1) # p(x|y)
    return p_xy + log_P_y

py = bayes_pred_stable(image)
py
```

Out[12]:

```
[-269.00424 -301.73447 -245.21458 -218.8941  -193.46907 -206.10315
 -292.54315 -114.62834 -220.35619 -163.18881]
<NDArray 10 @cpu(0)>
```

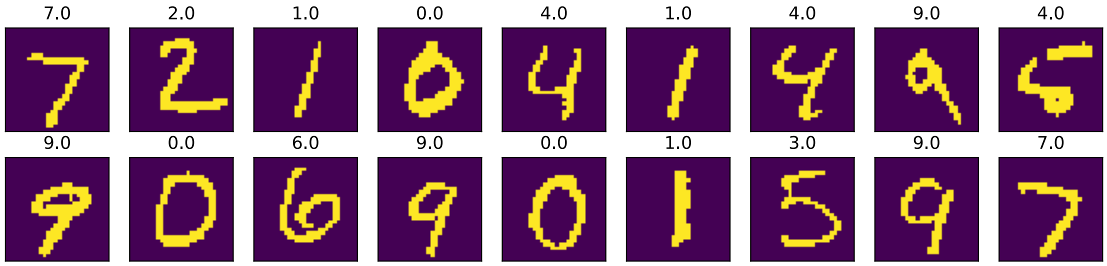
In [13]:

```python
py.argmax(axis=0).asscalar() == label
```

Out[13]: True

In [14]:

```python
def predict(X):
    return [bayes_pred_stable(x).argmax(axis=0).asscalar() for x in X]

X, y = mnist_test[:18]
show_images(X, 2, 9, titles=predict(X));
```



In [15]:
```
X, y = mnist_test[:]
py = predict(X)

'Validation accuracy', (nd.array(py).asnumpy() == y).sum() / len(y)
```

Out[15]:
```
('Validation accuracy', 0.8426)
```
