一、BayesianNetwork

$$X记输入的图像向量=[x_1,x_2,\dots,x_{784}],w_i记X属于第i类的概率$$

$$P(w_i|X)=\frac{P(w_i,X)}{P(X)}=\frac{P(w_i,X))}{\sum_{i=0}^{9}P(w_i,X)}$$

$$由朴素贝叶斯法,P(w_i,X)\approx P(w_i',X)(用w_i'近似w_i)$$

$$=P(w_i',x_1,\dots,x_{784})$$

$$=P(w_i')\prod_{j=1}^{n}P(x_j'|w_i')(x1,\dots,x_{784}关于w_i'条件独立)$$

$$=P(w_i')\frac{N(x_j',w_i')}{\sum_{xj'}N(x_j',w_i')}(*)$$

首先在fit函数中，利用训练数据计算$P(w_i')$和$P(x_j'|w_i')$,

self.labels_prior=$P(w_i')$，pixels[pic_n][pixel_n]=第pic_n个X的第pixel_n个维度的值=x_pixel_n,

第一个self.pixels_cond_label[pixel_n][pixels[pic_n][pixel_n]][pixel_n]=$N(x_j',w_i')$，(x_j=0/1）；

为了避免出现$P(x_j'=0,w_i')$=0或$P(x_j'=1,w_i')$=0，导致可能出现不好的影响，采用Laplace平滑；

第一个self.pixels_cond_label[pixel_n][pixels[pic_n][pixel_n]][pixel_n]=$P(x_j',w_i')=\frac{N(x_j',w_i')+1}{\sum_{xj'}N(x_j',w_i')+2}$，($x_j'$=0/1）；

```python
    # fit the model with training data
    def fit(self, pixels, labels):
        '''
        pixels: (n_samples, n_pixels, )
        labels: (n_samples, )
        '''
        n_samples = len(labels)
        # TODO: calculate prior probability and conditional probability
        #P(wi)=self.n_labels[i]/n_samples
        for cls in labels:
            self.labels_prior[cls]+=1
        self.labels_prior=self.labels_prior*1.0/n_samples
        # print(f"after fit,self.labels_prior={self.labels_prior}")
        # print(self.labels_prior)
        #P(xj=1|wi)
        for pic_n in range(n_samples):
            label_n=labels[pic_n]
            for pixel_n in range(self.n_pixels):
                self.pixels_cond_label[pixel_n][pixels[pic_n][pixel_n]][label_n]+=1
        for label_n in range(self.n_labels):
            for pixel_n in range(self.n_pixels):
                self.pixels_cond_label[pixel_n,:,label_n]=(self.pixels_cond_label[pixel_n,:,label_n]+1)*1.0/(self.pixels_cond_label[pixel_n,:,label_n].sum()+2) #Laplace平滑
        # print(f'self.pixels_cond_label是否存在0?{(self.pixels_cond_label==0).any()}')
```

在predict()中，利用公式$(*)$进行计算，可写成如下形式,

$$\begin{pmatrix} P(w_0|X) \\ \vdots \\ P(w_9|X) \end{pmatrix} = \begin{pmatrix} P(w_0') \\ \vdots \\ P(w_9') \end{pmatrix} \odot \begin{pmatrix} P(X_1|w_0') \\ \vdots \\ P(X_1|w_9') \end{pmatrix} \odot \cdots \odot \begin{pmatrix} P(X_{784}|w_0') \\ \vdots \\ P(X_{784}|w_9') \end{pmatrix}$$

一开始训练的时候分别计算P(w_i|X)，然后作归一化，得到（0,0,...,0),

可以按照上述形式，每计算一个维度，进行一次归一化(P(w_i|X)间，i=0~9)，就避免趋近于0

```
def predict(self, pixels):
    '''
    pixels: (n_samples, n_pixels, )
    return labels: (n_samples, )
    '''
    n_samples = len(pixels)
    labels = np.zeros(n_samples)
    # TODO: predict for new data
    for pic_n in range(n_samples):
        new_labels=self.labels_prior.copy()
        # print(new_labels,self.labels_prior)
        # print(f'self.pixels_cond_label是否存在0?{(self.pixels_cond_label==0).any()}')
        #直接相乘会导致new_labels趋向于0，应该先规范化
        # for label_n in range(self.n_labels):
        #     for pixel_n in range(self.n_pixels):
        #         new_labels[label_n]*=self.pixels_cond_label[pixel_n][pixels[pic_n][pixel_n]][label_n]
        for pixel_n in range(self.n_pixels):
            for label_n in range(self.n_labels):
                new_labels[label_n]*=self.pixels_cond_label[pixel_n][pixels[pic_n][pixel_n]][label_n]
                new_labels=new_labels/new_labels.sum()
        # print(new_labels,self.labels_prior)
        # pdb.set_trace()
        labels[pic_n] = np.argmax(new_labels)
        if(pic_n%1000==999):
            print("finish 1000 samples")
    return labels
```

训练结果如下:

```
PS C:\Users\Administrator\Desktop\exp2\part_1\src> python Bayesian-network.py
(60000, 784) (10000, 784)
[0.09871667 0.11236667 0.0993     0.10218333 0.09736667 0.09035
 0.09863333 0.10441667 0.09751667 0.09915    ]
finish 1000 samples
finish 1000 samples
finish 1000 samples
finish 1000 samples
finish 1000 samples
finish 1000 samples
finish 1000 samples
finish 1000 samples
finish 1000 samples
finish 1000 samples
test score: 0.843300
PS C:\Users\Administrator\Desktop\exp2\part_1\src>
```

二、K-means

1.assign_points

对每个points中的点，选择离它们最近的中心点center[center_n]，将这个点划分给这个中心点——labels[node_n]=center_n

```
    def assign_points(self, centers, points):
        '''
        centers: (n_clusters, n_dims,)
        points: (n_samples, n_dims,)
        return labels: (n_samples, )
        '''
        n_samples, n_dims = points.shape
        labels = np.zeros(n_samples,dtype=np.int8)
        # TODO: Compute the distance between each point and each center
        # and Assign each point to the closest center
        for node_n in range(n_samples):
            for center_n in range(1,self.k):
                if ((points[node_n]-centers[center_n])**2).sum() < ((points[node_n]-centers[labels[node_n]])**2).sum():
                    labels[node_n]=center_n

        return labels
```

2.把各个类的中心点更新为类中所有点的均值

```
    # Update the centers based on the new assignment of points
    def update_centers(self, centers, labels, points):
        '''
        centers: (n_clusters, n_dims,)
        labels: (n_samples, )
        points: (n_samples, n_dims,)
        return centers: (n_clusters, n_dims,)
        '''
        # TODO: Update the centers based on the new assignment of points
        new_centers=np.zeros((self.k,points[0].shape[0]))
        num_centers=[0]*self.k
        for i in range(len(points)):
            new_centers[labels[i]]+=points[i]
            num_centers[labels[i]]+=1
        print(f'num_centers={num_centers}')
        for i in range(self.k):
```

```
            centers[i]=(new_centers[i]/num_centers[i]).astype(np.uint8)
```

3. fit

首先随机初始化中心点，每次迭代，先把每个点分配给最近的中心点，

形成关于中心点的聚类，再把每个类的中心点更新为类中所有点的中心

```
    # k-means clustering
    def fit(self, points):
        '''
        points: (n_samples, n_dims,)
        return centers: (n_clusters, n_dims,)
        '''
        # TODO: Implement k-means clustering
        centers=self.initialize_centers(points)

        for i in range(self.max_iter):
            new_labels=self.assign_points(centers,points)
            print(f"epoch{i}:new_labels={new_labels}")
            self.update_centers(centers,new_labels,points)
            for i in range(self.k):
                print(centers[i])

        return centers,new_labels
```

4. compress

points为原图展开的点列，按k-means方法得到labels和center后，

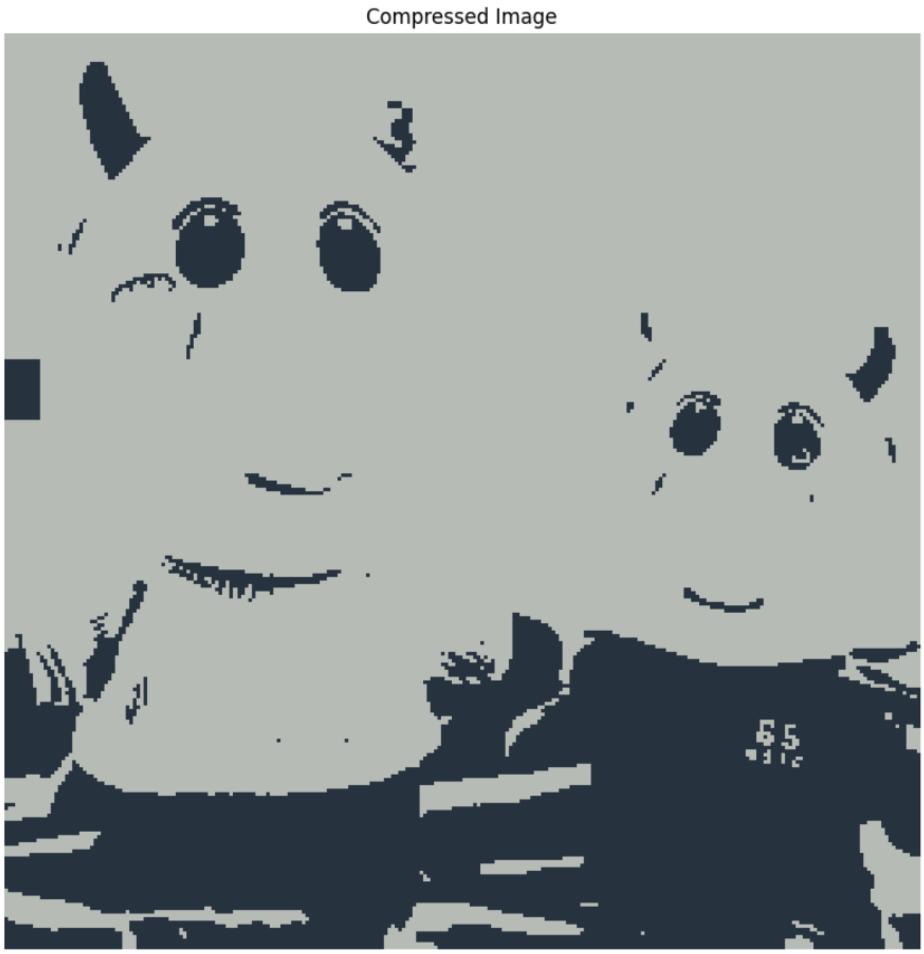
被每个点node_n（属于第lablesl[i]个类）的坐标，换成它的中心centers[lables[i]]的坐标，

要记得把点列还原成原图的形状！

```
    def compress(self, img):
        '''
        img: (width, height, 3)
        return compressed img: (width, height, 3)
        '''
        # flatten the image pixels
        shape=img.shape
        points = img.reshape((-1, img.shape[-1]))
        # TODO: fit the points and
        # Replace each pixel value with its nearby center value
        centers,labels=self.fit(points)
        for i in range(len(points)):
            points[i]=centers[labels[i]]
        points=points.reshape(shape)
        return points
```

Result(保留在一个src下)

k=2



k=4

k=8

Compressed Image

---

k=16

Compressed Image

k=32

Compressed Image