Assignment 1

Pattern Recognition and Machine Learning

Fudan University @ 2020 Spring

2020 年 4 月 15 日

1 数据生成

在本节中，将设计 3 个二维高斯分布，分别标记为类别 A、B、C，从这三个高斯分布中随机采样得到数据集。

1.1 二维高斯分布

对每个二维高斯分布，分别定义三个参数：均值 μ，协方差 σ，数据规模 sz，概率密度函数如下，

$$f(x) = \frac{1}{2\pi\sqrt{det(\sigma)}} exp(-\frac{1}{2}(x-\mu)^T\sigma^{-1}(x-\mu)) \tag{1}$$

1.2 数据集

数据集（data.data）中，三个高斯分布的均值分别为 (4,3)，(0,0)，(-3,4)，协方差矩阵均为 [(1, 0), (0, 1)]，样本数量均为 100。在采样完成后，进行随机排序，使不同类别的样本在数据集中均匀分布。

可通过调用函数 gendata()，调整参数（均值、协方差、样本数量），产生不同的数据集，详见第 4 节代码运行说明。

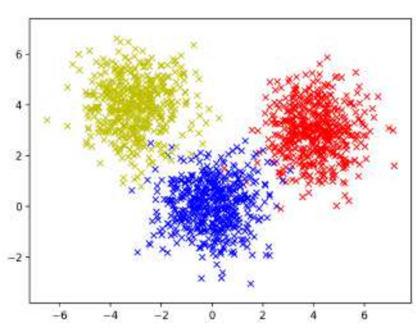


图 1: 大数据集样本分布

2 模型构建

在本节中，首先将构建两个线性分类模型：一个生成模型 (Generative Model)——朴素贝叶斯模型，和一个判别模型 (Discriminative Model)——逻辑斯谛回归；然后比较它们之间的差异。

2.1 生成模型

本节叙述朴素贝叶斯法，及其代码实现。

2.1.1 朴素贝叶斯

朴素贝叶斯 (naive Bayes) 法是基于贝叶斯定理与特征条件独立假设的分类方法。对于给定的训练数据集，首先基于特征条件独立假设学习输入输出的联合概率分布；然后基于此模型，对给定的输入 x，利用贝叶斯定理求出后验概率最大的输出 y。朴素贝叶斯实现简单，学习和预测的效率都很高，是一种常用的方法。

2.1.2 问题描述

设输入空间 $\chi \subseteq \mathbb{R}^2$ 为 2 维向量的集合，输出空间为类标记集合 $\gamma = \{c_1, c_2, c_3\}$。输入为特征向量 $x \in \chi$，输出为类标记 $y \in \gamma$。P(X,Y) 是 X 和 Y 的联合概率分布。训练数据集

$$T = \{(x_0, y_0), (x_1, y_1), ..., (x_N, y_N)\}$$

由 P(X,Y) 独立同分布产生。朴素贝叶斯法通过训练数据集学习联合概率分布 P(X,Y)。具体地，学习以下先验概率分布及条件概率分布。先验概率分布

$$P(Y = c_k), k = 1, 2, 3 \tag{2}$$

条件概率分布

$$P(X = x|Y = c_k) = P(X^{(1)} = x^{(1)}, X^{(2)} = x^{(2)}|Y = c_k) \tag{3}$$

于是学习到联合概率分布 P(X,Y)。朴素贝叶斯法对条件概率分布作了条件独立性的假设

$$\begin{aligned} P(X = x|Y = c_k) &= P(X^{(1)} = x^{(1)}, X^{(2)} = x^{(2)}|Y = c_k) \\ &= \prod_{j=1}^{3} P(X^{(j)} = x^{(j)}|Y = c_k) \end{aligned} \tag{4}$$

朴素贝叶斯法分类时，对给定的输入 x，通过学习到的模型计算后验概率 $P(Y=c_k|X=x)$，将后验概率最大的类作为 x 的类输出。后验概率计算根据贝叶斯定理进行：

$$P(Y=c_k|X=x)=\frac{P(X=x|Y=c_k)P(Y=c_k)}{\sum_k P(X=x|Y=c_k)P(Y=c_k)} \tag{5}$$

将公式 (3) 代入 (4)，有

$$P(Y=c_k|X=x)=\frac{P(Y=c_k)\prod_j P(X^{(j)}=x^{(j)}|Y=c_k))}{\sum_k P(Y=c_k)\prod_j P(X^{(j)}=x^{(j)}|Y=c_k)} \tag{6}$$

这是朴素贝叶斯法分类的基本公式。于是，朴素贝叶斯分类器可以表示为

$$y=f(x)=\underset{c_k}{argmax}\frac{P(Y=c_k)\prod_j P(X^{(j)}=x^{(j)}|Y=c_k))}{\sum P(Y=c_k)\prod_j P(X^{(j)}=x^{(j)})|Y=c_k)} \tag{7}$$

注意到，分母对所有 c_k 都是相同的，所以，

$$y=\underset{c_k}{argmax}\,P(Y=c_k)\prod_j P(X^{(j)}=x^{(j)}|Y=c_k)) \tag{8}$$

2.1.3 实现细节

本节将介绍朴素贝叶斯算法的实现细节，分为如下 4 个部分:

- Step 1: 计算先验概率
- Step 2: 计算条件概率
- Step 3: 计算后验概率
- Step 4: 确定实例的类

Step 1: 计算先验概率分布

我们需要根据数据所属的类别，来计算每个类别的概率，即先验概率。

$$P(Y=c_k)=\frac{\sum_{i=1}^N I(y_i=c_k)}{N}, k=1,2,3$$

因此首先需要将训练数据按类分开，可以创建一个 dictionary 对象，键为类别，值为该类别所对应的全部数据记录。由如下的 separate_by_class() 函数实现。

```python
# Split the dataset by class values, returns a dictionary
def separate_by_class(dataset):
    separated = dict()
    for i in range(len(dataset)):
```

```python
        vector = dataset[i]
        class_value = vector[-1]
        if (class_value not in separated):
            separated[class_value] = list()
        separated[class_value].append(vector)
    return separated
```

Step 2: 计算条件概率分布

对于连续变量 x，如何去估计似然度 $P(x|y_i)$ 呢？我们可以假设在 y_i 的条件下，x 服从高斯分布（正态分布）。根据正态分布的概率密度函数即可计算出 $P(x|y_i)$，公式如下，

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{(x-\mu)^2}{2\sigma^2}) \tag{9}$$

因此，对于给定的数据集，我们需要其两个统计信息：均值 μ 和标准差 σ，公式如下，

$$\begin{aligned} \mu &= \frac{sum(X)}{count(X)} \\ \sigma &= \sqrt{\sum_{1}^{N} \frac{(x_i - mean(x))^2}{N-1}} \end{aligned} \tag{10}$$

首先，把数据集按照类标记分为若干个子数据集；然后，计算子数据集中每列数据的均值和标准差；最后，返回统计列表。由如下的 summarize_by_class() 函数实现。

```python
# Split dataset by class then calculate statistics for each row
def summarize_by_class(dataset):
    separated = separate_by_class(dataset)
    summaries = dict()
    for class_value, rows in separated.items():
        # Calculate the mean, stdev and count for each column in a dataset
        summary=[(mean(column), stdev(column), len(column)) for column in zip(
                                                                    *rows)]
        del (summary[-1])
        summaries[class_value] = summary
    return summaries
```

由返回的统计列表，容易计算条件概率

$$P(X^{(j)} = a_j)|Y = c_k) = \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{(a_j-\mu)^2}{2\sigma^2})$$

Step 3: 计算后验概率

对给定的实例 $X = (x^{(1)}, x^{(2)})^T$，计算

$$P(Y = c_k) \prod_{j=1}^{n} (X^j = x^j | Y = c_k), k = 1, 2, 3$$

由函数 calculate_class_probabilities() 实现。

```python
# Calculate the probabilities of predicting each class for a given row
def calculate_class_probabilities(summaries, row):
    total_rows = sum([summaries[label][0][2] for label in summaries])
    probabilities = dict()
    for class_value, class_summaries in summaries.items():
        # P(class)
        probabilities[class_value] = summaries[class_value][0][2] / float(total_rows)
        for i in range(len(class_summaries)):
            mean, stdev, _ = class_summaries[i]
            probabilities[class_value] *= calculate_probability(row[i], mean, stdev)
    return probabilities
```

Step 4: 确定实例的类

计算出后验概率后，可以确定实例 x 的类

$$y = \underset{c_k}{argmax}\, P(Y = c_k) \prod_{j=1}^{n} P(X^{(j)} = x^{(j)} | Y = c_k)$$

由函数 predict() 实现。

```python
# Predict the class for a given row
def predict(summaries, row):
    probabilities = calculate_class_probabilities(summaries, row)
    best_label, best_prob = None, -1
    for class_value, probability in probabilities.items():
        if best_label is None or probability > best_prob:
            best_prob = probability
            best_label = class_value
    return best_label
```

2.2 判别模型

本节叙述逻辑斯蒂回归，及其代码实现。

2.2.1 逻辑斯谛回归

逻辑斯谛回归 (logistic regression) 是统计学习中的经典分类方法，属于对数线性模型。

2.2.2 问题描述

二元逻辑斯谛回归模型是如下的条件概率分布：

$$P(Y=0|x)=\frac{exp(-wx+b)}{1+exp(-wx+b)}$$
$$P(Y=1|x)=\frac{1}{1+exp(-wx+b)} \tag{11}$$

这里，$x\in\mathbb{R}^n$ 是输入，$Y\in\{0,1\}$ 是输出，$w\in\mathbb{R}^n$ 和 $b\in\mathbb{R}$ 是参数，w 称为权值向量，b 称为偏置，wx 为 w 和 x 的内积。

对于给定的输入实例 x，按照公式 (11) 可以求得 $P(Y=1|x)$ 和 $P(Y=0|x)$ 。逻辑斯谛回归比较两个条件概率值的大小，将实例 x 分到概率值较大的那一类。

有时为了方便,将权值向量和输入向量加以扩充,仍记作 w,x,即 $w=(b,w^{(1)},w^{(2)},...,w^{(n)})^T$，$x=(1,x^{(1)},x^{(2)},...,x^{(n)})^T$ 。这时，逻辑斯谛回归模型如下，

$$P(Y=0|x)=\frac{exp(-wx)}{1+exp(-wx)}$$
$$P(Y=1|x)=\frac{1}{1+exp(-wx)} \tag{12}$$

2.2.3 实现细节

本节将介绍逻辑斯谛回归算法的实现细节，分为如下 3 个部分：

- Step 1：作出预测
- Step 2：参数估计
- Step 3: 多项逻辑斯谛回归

Step 1: 作出预测

对于给定的训练数据集 $T=\{(x_1,y_1),(x_2,y_2),...,(x_N,y_N),\}$，其中，$x_i\in\mathbb{R}^n,y_i\in\{0,1\}$，用逻辑斯谛回归模型对每个样本 $x^{(n)}$ 进行预测，输出其标签为 1 的后验概率，记为 $\hat{y}^{(n)}$，

$$\hat{y}^{(n)}=\sigma(w^Tx^{(n)})$$

由于 $\hat{y}\in\{0,1\}$，样本 $(x^(n),y^(n))$ 的真实条件概率可以表示为

$$P(y^{(n)}=1|x^{(n)})=y^{(n)},$$
$$P(y^{(n)}=0|x^{(n)})=1-y^{(n)}$$

这时，线性函数的值越接近正无穷，概率的值就越接近 1；这时，线性函数的值越接近负无穷，概率的值就越接近 0（如图 2 所示）。由函数 dis_predict() 实现。

```python
# Make a prediction with coefficients
def dis_predict(row, coefficients):
    yhat = coefficients[0]
    for i in range(len(row) - 1):
        yhat += coefficients[i + 1] * row[i]
    return sigmoid(yhat)
```

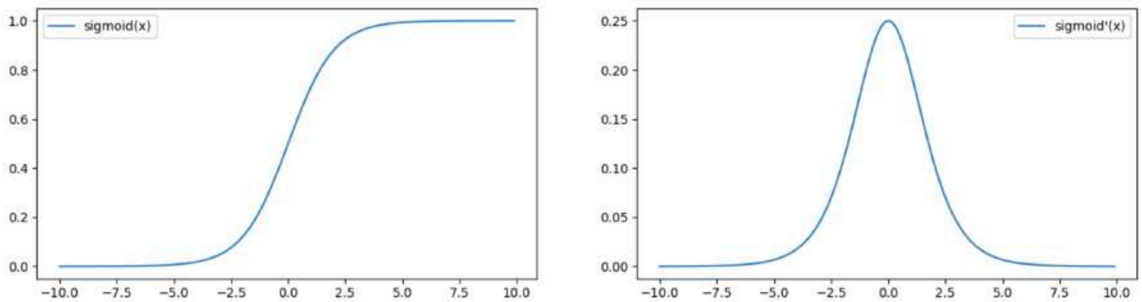


图 2: 逻辑斯谛分布的分布函数与密度函数

Step 2: 参数估计

逻辑斯谛回归采用交叉熵作为损失函数，并使用梯度下降法来对参数进行优化。其风险函数为

$$L(w) = -\frac{1}{N}\sum_{i=1}^{N}[y^{(n)}log\hat{y}^{(n)} + (1-y^{(n)})log(1-\hat{y}^{(n)})] \tag{13}$$

风险函数 $L(w)$ 关于参数 w 的偏导数为

$$\frac{\partial L(w)}{\partial w} = -\frac{1}{N}\sum_{n=1}^{N}x^{(n)}(y^{(n)} - \hat{y}^{(n)}) \tag{14}$$

采用梯度下降法，逻辑斯谛回归的训练过程为: 初始化 $w_0 \leftarrow 0$，然后通过下式来迭代更新参数:

$$w_{t+1} \leftarrow w_t + \alpha\frac{1}{N}\sum_{n=1}^{N}x^{(n)}(y^{(n)} - \hat{y}^{(n)}_{w_t}) \tag{15}$$

其中 α 是学习率，$\hat{y}^{(n)}_{w_t}$ 是当参数为 w_t 时，逻辑斯谛回归模型的输出。由函数 coefficients_sgd() 实现，

```python
def coefficients_sgd(train, l_rate, n_epoch):
    coef = [0.0 for i in range(len(train[0]))]
    for epoch in range(n_epoch):
        sum_error = 0
```

```
        for row in train:
            yhat = dis_predict(row, coef)
            error = row[-1] - yhat
            sum_error += error ** 2
            coef[0] = coef[0] + l_rate * error
            for i in range(len(row) - 1):
                coef[i + 1] = coef[i + 1] + l_rate * error * row[i]
    return coef
```

Step 3: 多项逻辑斯谛回归

用 3 个二分类器来实现一个三项分类器。3 个二分类器分别检测类别 1、类别 2 和类别 3，最后用 argmax 函数，看哪个分类预测的概率最高，就将实例 x 划分为哪一类。由函数 logistic_regression() 实现。

```
# Linear Regression Algorithm With Stochastic Gradient Descent
def logistic_regression(train, test, l_rate, n_epoch):
    predictions = list()
    coefs=list()
    for i in range(0,3):
        train_copy=copy.deepcopy(train)
        for row in train_copy:
            if(row[-1]==i+1):
                row[-1]=1
            else:
                row[-1]=0
        coef = coefficients_sgd(train_copy, l_rate, n_epoch)
        coefs.append(coef)
    for row in test:
        maxp=-1
        label=0
        for i in range(0,3):
            yhat = dis_predict(row, coefs[i])
            if(yhat>maxp):
                maxp=yhat
                label=i+1
        predictions.append(label)
    return (predictions)
```

2.3 模型比较

1. 生成方法由数据学习联合概率分布 $P(X,Y)$，然后求出条件概率分布 $P(Y|X)$ 作为预测的模型，即生成模型：
$$p(Y|X) = \frac{P(X,Y)}{P(X)}$$
而判别方法由数据直接学习条件概率分布 $P(Y|X)$ 作为预测的模型。

2. 生成模型表示了给定输入 X 产生输出 Y 的生成关系，而判别方法关心的是对给定的输入 X，应该预测什么样的输出 Y。

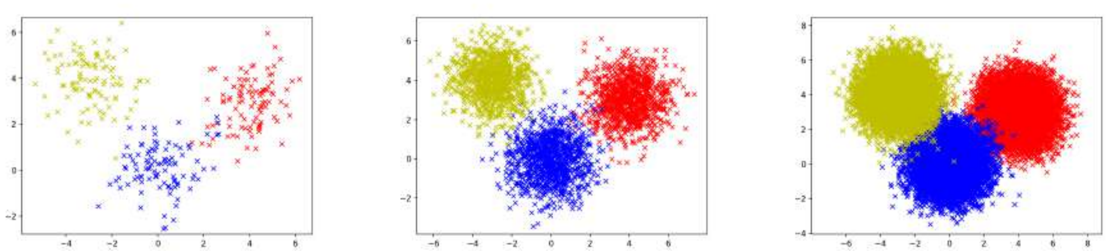
3. 生成方法的学习收敛速度更快，即当样本容量增加的时候，学到的模型可以更快地收敛于真实模型

4. 判别方法直接学习的是条件概率 $P(Y|X)$，直接面对预测，往往学习的准确率更高。

3 实验

在这一部分中，将重新组织数据集的规模，调整不同高斯分布之间的重叠，通过 5 次交叉检验的方法，从运行时间和分类准确度的角度，来检验模型的性能。

3.1 调整数据规模



(a) 100×3　　(b) $1{,}000\times 3$　　(c) $10{,}000\times 3$

图 3: 调整数据规模

生成模型		
数据集规模	准确率 (%)	运行时间 (s)
50	99.33	0.000
100	97.00	0.001
200	99.33	0.001
500	99.40	0.003
1000	99.10	0.005
2000	99.35	0.012
5000	99.21	0.030
10000	99.19	0.061

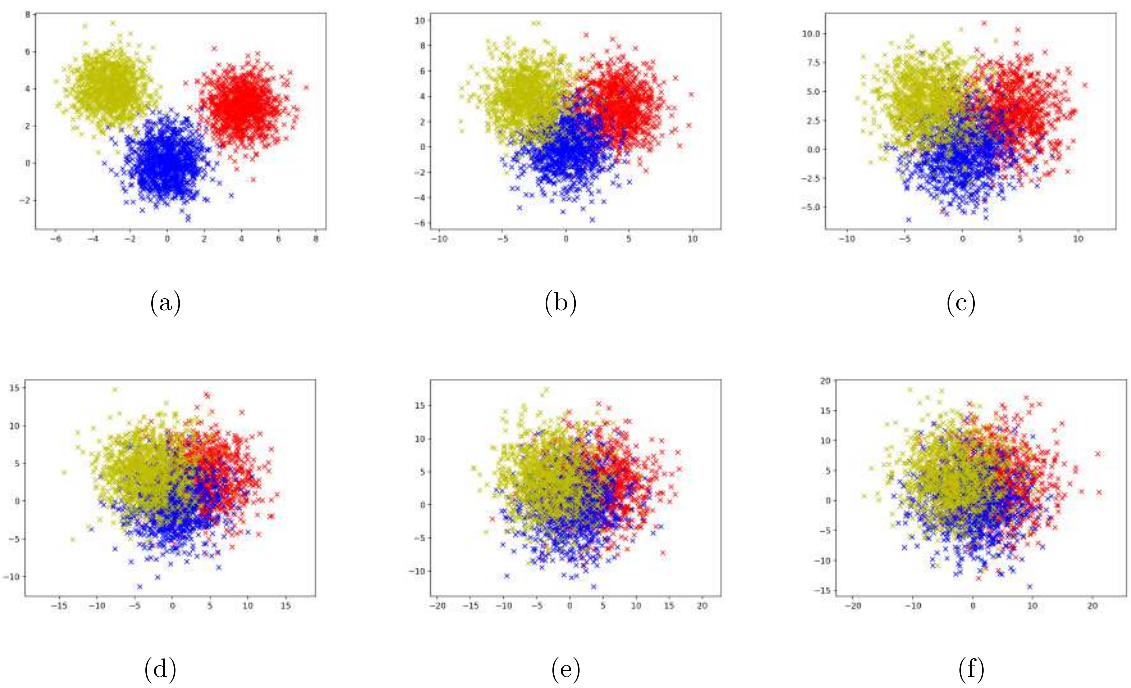
判别模型		
数据集规模	准确率 (%)	运行时间 (s)
50	98.67	0.052
100	98.00	0.010
200	99.00	0.218
500	98.33	0.518
1000	98.83	1.242
2000	98.83	2.307
5000	98.89	5.717
10000	99.03	12.042

表 1: 数据规模对生成/判别模型准确率及运行时间的影响

由表 1 可看出，随着数据规模的扩大，

- 准确率方面，生成模型和判别模型都无明显变化，均能保持较高的准确率
- 运行时间方面，生成模型运行时间随着数据规模线性增长，判别模型运行时间呈非线性增长

3.2 调整数据重叠



(a) (b) (c)

(d) (e) (f)

图 4: 调整数据重叠

(a)-(f) 协方差分别为，[[1, 0], [0, 1]]，[[3, 0], [0, 3]]，[[5, 0], [0, 5]]，[[10, 0], [0, 10]]，[[15, 0], [0, 15]]，[[20, 0], [0, 20]]

生成模型		
协方差	准确率 (%)	运行时间 (s)
[[1, 0], [0, 1]]	99.33	0.005
[[2, 0], [0, 2]]	95.30	0.005
[[3, 0], [0, 3]]	88.57	0.005
[[4, 0], [0, 4]]	85.20	0.006
[[5, 0], [0, 5]]	81.97	0.006
[[10, 0], [0, 10]]	69.97	0.005
[[15, 0], [0, 15]]	63.13	0.005
[[20, 0], [0, 20]]	58.93	0.005

判别模型		
协方差	准确率 (%)	运行时间 (s)
[[1, 0], [0, 1]]	98.77	1.063
[[2, 0], [0, 2]]	95.00	1.083
[[3, 0], [0, 3]]	87.60	1.146
[[4, 0], [0, 4]]	84.70	1.157
[[5, 0], [0, 5]]	79.60	1.144
[[10, 0], [0, 10]]	69.60	1.149
[[15, 0], [0, 15]]	62.60	1.154
[[20, 0], [0, 20]]	58.73	1.090

表 2: 数据重叠对生成/判别模型准确率及运行时间的影响

由图 4 和表 2 可知，随着重叠率增加，

- 生成模型和判别模型准确率都降低
- 数据规模不变，运行时间不随着协方差变化

4 代码运行

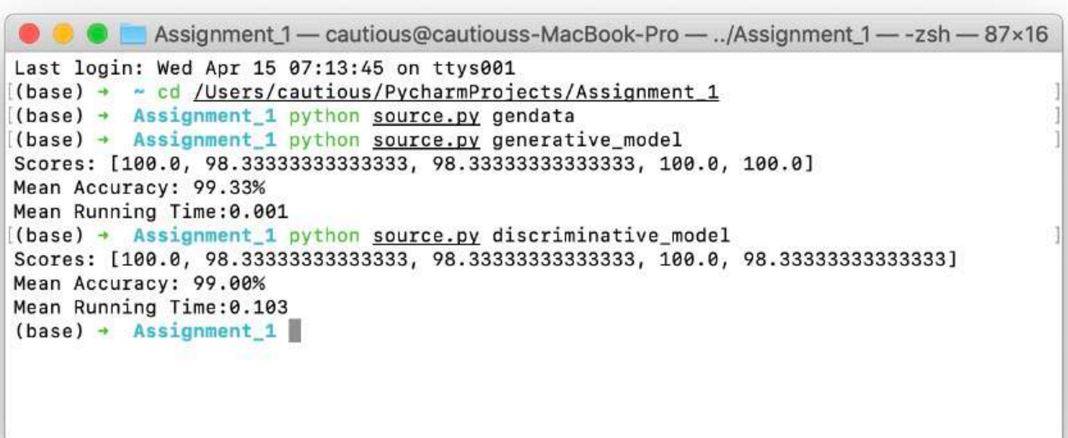


图 5: 代码运行示例

产生数据:

```
$ python source.py gendata
```

可调参数:

- 数据规模 sz (default sz:100)
- 均值 mean (default mean:[4, 3],[0, 0],[-3, 4])
- 协方差 cov (default cov:[[1, 0], [0, 1]]

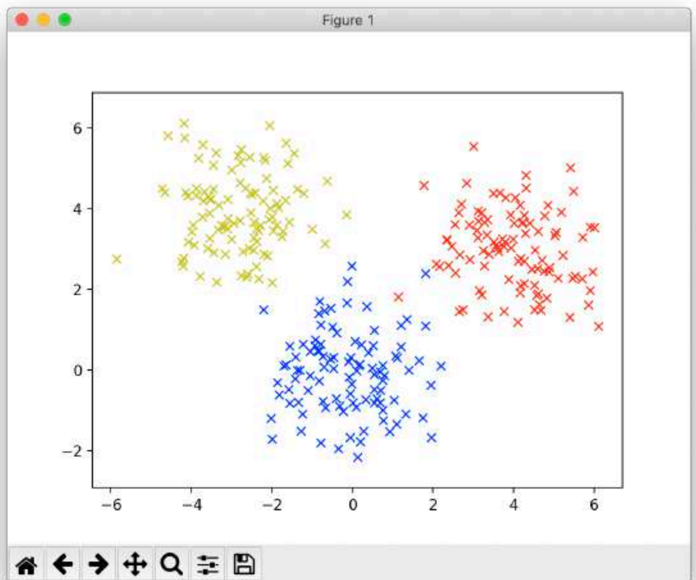


图 6: gendata() 运行结果

产生模型:

```
$ python source.py generative_model
```

判别模型:

```
$ python source.py generative_model
```

可调参数:

- 学习率 l_rate (default l_rate: 0.1)
- 训练周期 n_epoch (default n_epoch: 100)

运行结果: 5 折交叉检验的平均准确率和平均运行时间