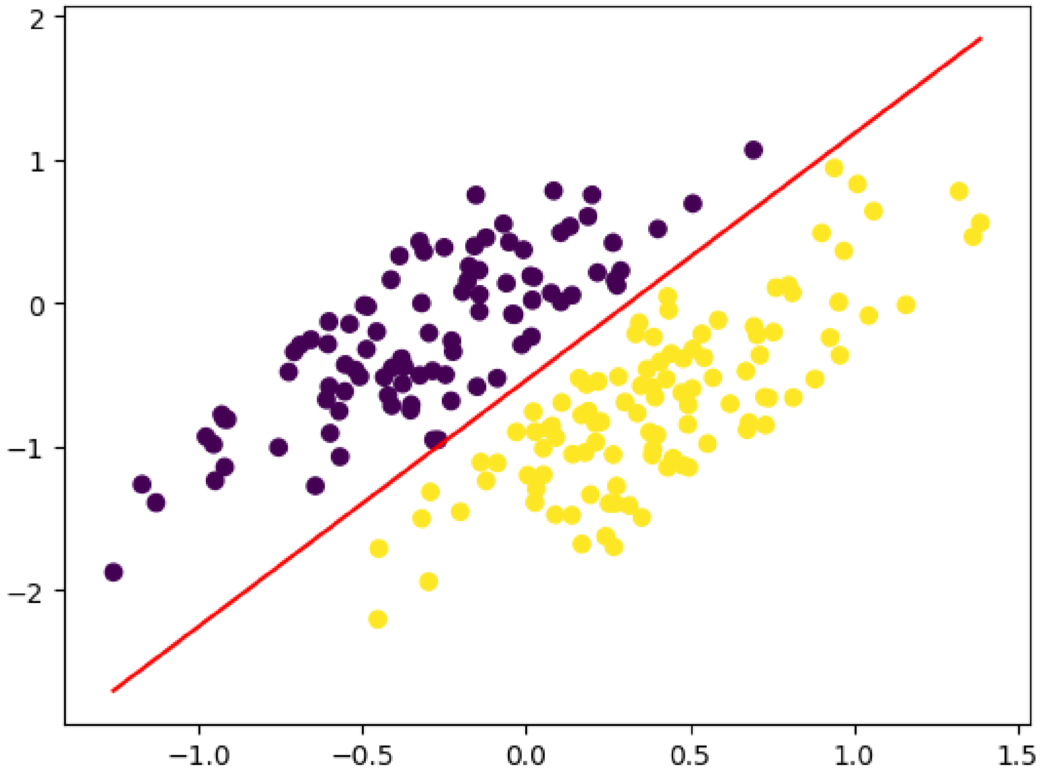
PRML Assignment 2 Report

Part 1

- **Least Square Model**

  对于最小平方方法，判别函数为$y_k(x) = w_k^T * x + w_{k0}$，也可以表示为$y(x) = W^T * x$，其中$x$即为输入向量，在该数据集中为二维坐标向量$(x, y)$，因此可以使用$ax_1 + bx_2 + c$的形式来表示该判别函数。因此，利用参数矩阵$W$的公式$W = (X^T X)^{-1} X^T T$，即可得相应的判别函数。

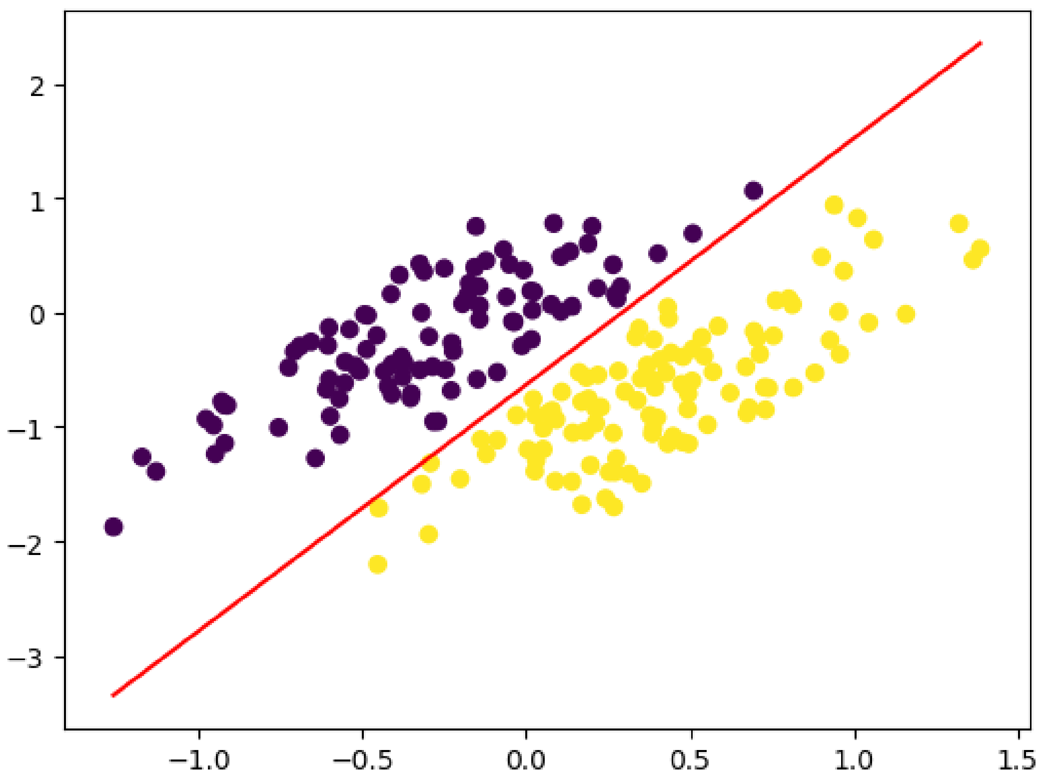
  结果如下： $a = 1.654557, b = -0.964092, c = -0.518672, accuracy = 1.000000$



- **Perceptron**

  对于感知器模型，同样判别函数有$y(x) = W^T x$或$ax_1 + bx_2 + c$的形式。根据算法原理，给定判别函数参数初始值$a, b, c = 1$，并且利用$loss = y * (ax_1 + bx_2 + c)$来判断该输入是否被错误分类。在每次遍历数据集后，若仍存在损失loss，则利用随机梯度下降算法对参数进行修正，即通过随机函数在被错分的数据集中随机选取数据(x_{1i}, x_{2i}, y)。根据$loss$分别对a, b, c求梯度，即可得到参数的修正公式为
  $a = a + \eta * x_{1i} * y, b = b + \eta * x_{2i} * y, c = c + \eta * y$。

  结果如下： $a = 1.024418, b = -0.475181, c = -0.299999, accuracy = 1.000000$



Part 2

Q1

1、preprocess：预处理方面，分别对于数字、标点符号、空白字符清洗，利用`re.sub('\d+',' '.str)`替换数字为空格，忽略$string.punctuation$内字符，将$string.whitespace$内字符替换为空格。在进行完以上清洗后，将所有字符转化为小写字符，并按空格split切分，每条data得到一个对应的$datalist$。在此基础上，对所有数据中word进行计数，形成$key: word, value: frequency$的字典。为了防止得到的multi-hot码过长，需要对词频进行筛选，设定$mincount = 10$，去除字典中频数小于$mincount$的词，得到最终用于构建multi-hot码的字典$worddict$。以上为预处理部分进行的操作。

2、multi-hot：利用预处理阶段得到的单词列表$datalist$与构建multi-hot所需字典$worddict$生成multi-hot码。对于$datalist$中每一条$data$，初始化一个1*len(worddict)大小的multi-hot码；进而对于该$data$中每个$word$，若该$word$出现在字典中，则将multi-hot码相应位置修改为1(位置作为value存储在字典中，即$worddict[word]$)。最终即可将所有document均表示为multi-hot向量形式，存储在大小为len(datalist)*len(worddict)的矩阵内。

3、one-hot：同样为了方便运算，需要将每条data对应种类dataset_train.traget也转换为one-hot形式，共有len(dataset_train.target_names)个类，因此初始化1*len(dataset_train.target_names)向量存放每条数据对应的类别。就该案例而言，类别共4类，在数据中以0，1，2，3的形式存储，因此可以分别用1000，0100，0010，0001来表示四个种类(将相应位置修改为1即可)。最终即可将数据对应种类表示成one-hot形式，存储在大小为len(datalist)*len(target_names)的矩阵内。

Q2

公式推导：

- 先求$\frac{\partial L}{\partial W_{i,j}}$

推导过程：采用直接求对矩阵W的梯度的方法来推导上式，即已知矩阵W，函数L(W)的函数值为标量，求$\frac{\partial L}{\partial W}$。可以得到矩阵导数与微分的关系即$dL=\sum_{i,j}\frac{\partial L}{\partial W_{i,j}}dW_{i,j}=tr((\frac{\partial L}{\partial W})^T dW)$，在此对于本式的正确性不做验证。因此，推导即转化为先求dL的表达式，然后再套上迹tr，最后将表达式$tr(dL)$和$tr((\frac{\partial L}{\partial W})^T dW)$进行比对，总而得到所需的$\frac{\partial L}{\partial W}$。

首先求取dl，对于回归方程中参数W^Tx+b，不妨令其为z，方便运算，具体计算如下：

（注：由于对$w_{i,j}$项求梯度，因此忽视$\sum$项与$\frac{1}{N}$项，此外将公式中log替换为ln，简化运算）

$$L=-y^T ln\frac{exp(z)}{1^T exp(z)}=-y^T(z-ln(\begin{pmatrix}1^T exp(z)\\ \cdots\\ 1^T exp(z)\end{pmatrix}))=ln(1^T exp(z))-y^T z$$

根据微分法则可得：

$$d(ln(1^T exp(z)))=\frac{1}{1^T exp(z)}\odot d(1^T exp(z))$$

$$d(1^T exp(z))=1^T d(exp(z))=1^T(exp(z)\odot dz)$$

在此基础上即可计算dL：

$$dL=\frac{1^T(exp(z)\odot dz)}{1^T exp(z)}-y^T dz$$

根据矩阵迹的恒等式即可得到以下结果:

$$dL=tr(\frac{(1\odot exp(z))^T dz}{1^T exp(z)})-tr(y^T dz)=tr((\frac{(exp(z))^T}{1^T exp(z)}-y^T)dz)$$

$$=tr((\hat{y}-y)^T dz)=tr((\frac{\partial L}{\partial z})^T dz)$$

对于之前定义的z，有$dz=d(W^Tx+b)=(dW^T)x+W^T dx=(dW^T)x$

并且根据迹的恒等式$tr(ABC)=tr(BCA)=tr(CAB)$，因此可以得到：

$$dL=tr((\frac{\partial L}{\partial z})^T(dW^T)x)=tr(x(\frac{\partial L}{\partial z})^T dW^T)=tr((\frac{\partial L}{\partial W^T})^T dW^T)$$

也就是说$\frac{\partial L}{\partial W^T}=\frac{\partial L}{\partial z}x^T=(\hat{y}-y)x^T$

即可得到**最终结果：**$\frac{\partial L}{\partial W}=-x(y-\hat{y})^T$

- 再求$\frac{\partial L}{\partial b_i}$

  推导过程：与上一步采用相同的推导方法，利用之前得到的计算结果dL

  可以得到**最终结果：**$\frac{\partial L}{\partial b}=-(y-\hat{y})^T$

- 对于正则化项$\lambda\|W\|_2^2$

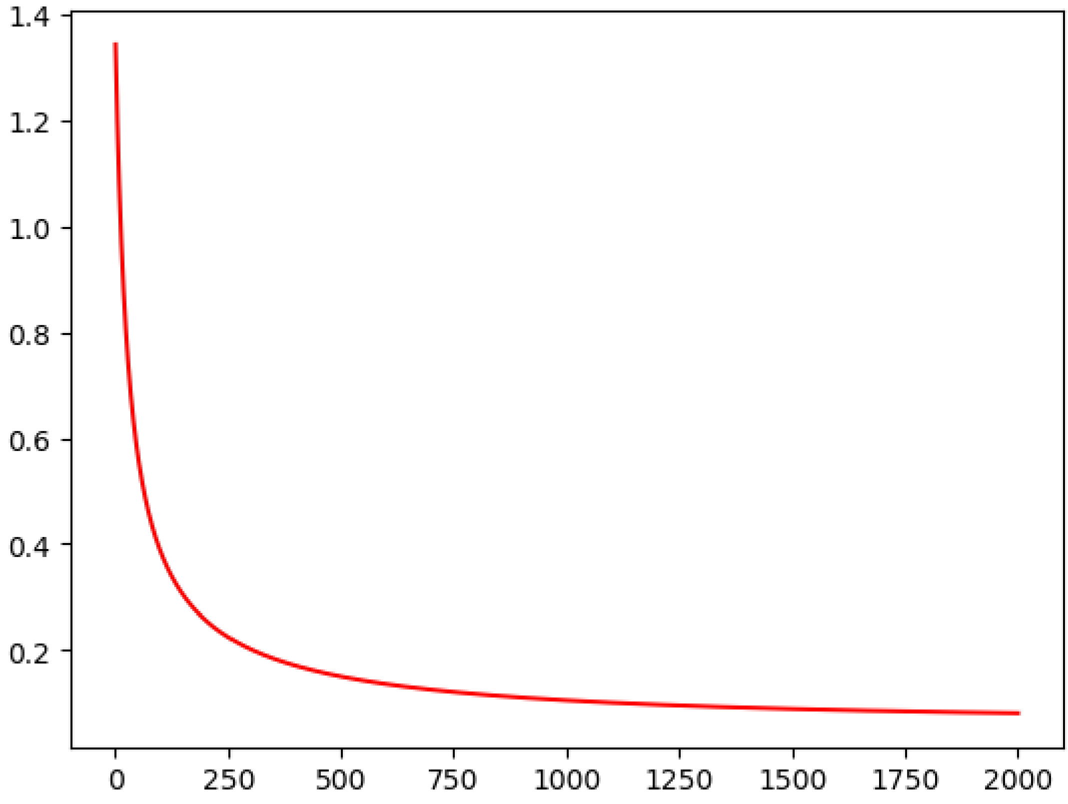
  推导过程：$\frac{\partial\lambda\|W\|_2^2}{\partial w_{i,j}}=\frac{\partial\lambda\sum_{j=1}^{d}|w_{i,j}|^2}{\partial w_{i,j}}=2*\lambda*w_{i,j}$

（1）在拟合过程中，通常倾向于让权值尽可能小，最后构造一个所有参数都比较小的模型，因为一般认为参数值小的模型比较简单，能够适应不同的数据集。可以设想一下，对于一个回归方程，若参数很大，那么只要数据偏移一点点，就会对结果造成很大的影响；但如果参数足够小，数据偏移得多一点也不会对结果曹诚什么影响。因此，采用L2正则化来获得值很小的参数，也能在一定程度上避免过拟合现象。可以看到，对于$y=softmax(W^Tx+b)$，若利用$\lambda\|W\|_2^2$对W进行了L2正则化，则最终得到的参数W会很小，所以无论b是否进行L2正则化，当数据x发生偏移，在W^T*x乘法项的影响下，对结果的影响都很小。因此，没有必要对于bias项也进行L2正则化。

（2）在以上公式推导过程中，能够得到需要的梯度的解析解，但由于计算过程中涉及较多参数，反向传播计算的梯度很容易出现误差，导致迭代后参数效果很差。可以采用梯度检验的方法，确认代码中反向传播计算的梯度是否正确：通过计算数值梯度，得到梯度的近似值，然后将该近似值和反向传播得到的梯度进行比较，若两者相差很小则证明了梯度的计算是无误的，即利用了同一梯度的两种计算方式，由于数值梯度得到的总是接近于正确的数值解，因此可用来当作解析梯度的参照。

Q3

Loss Curve如下(Full Batch Gradient Descent):



参数：learning rate = 0.1，λ = 0.001，迭代次数iteration = 2000

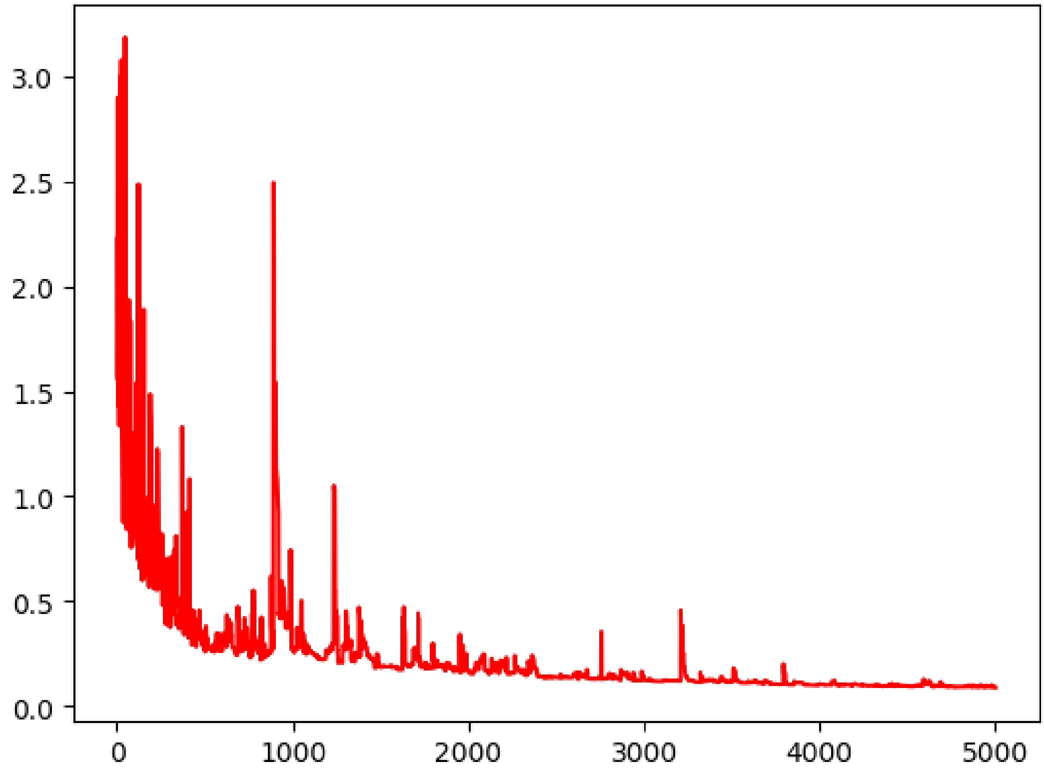
（1）训练模型时，合适的学习率对于模型的训练格外重要，因此自始至终保持同样的学习率是不合适的。在训练初期参数刚刚开始学习时，此时参数与最优解相差较远，需要保持一个较大的学习率以尽快逼近最优解；但模型训练到后期时，参数与最优解已经隔得比较近，若仍然保持最初的学习率，容易越过最优点，而在最优点附近来回振荡，即通俗来说容易学过头。因此可以采取以下方式修正学习率：开始设置一个较大的学习率rate，每经过一定次数的迭代，就将学习率减半，通过以上方式，即可实现初期较大学习率而快速逼近最优解，而后期较小学习率慢慢逼近最优解的目的。

（2）有以下的几种条件用于判断训练过程是否终止：首先，若前一周期所有的ΔW(即W的修改量)都太小，且小于某一指定的阈值border，则终止模型的训练；其次，若前一周期误分类的元组百分比小于某一指定的阈值border，则终止模型的训练；最后，通过预先指定迭代的周期数，若超过该周期数，则终止模型的训练，在本次assignment中采用该方式作为终止条件。

Q4

（1）

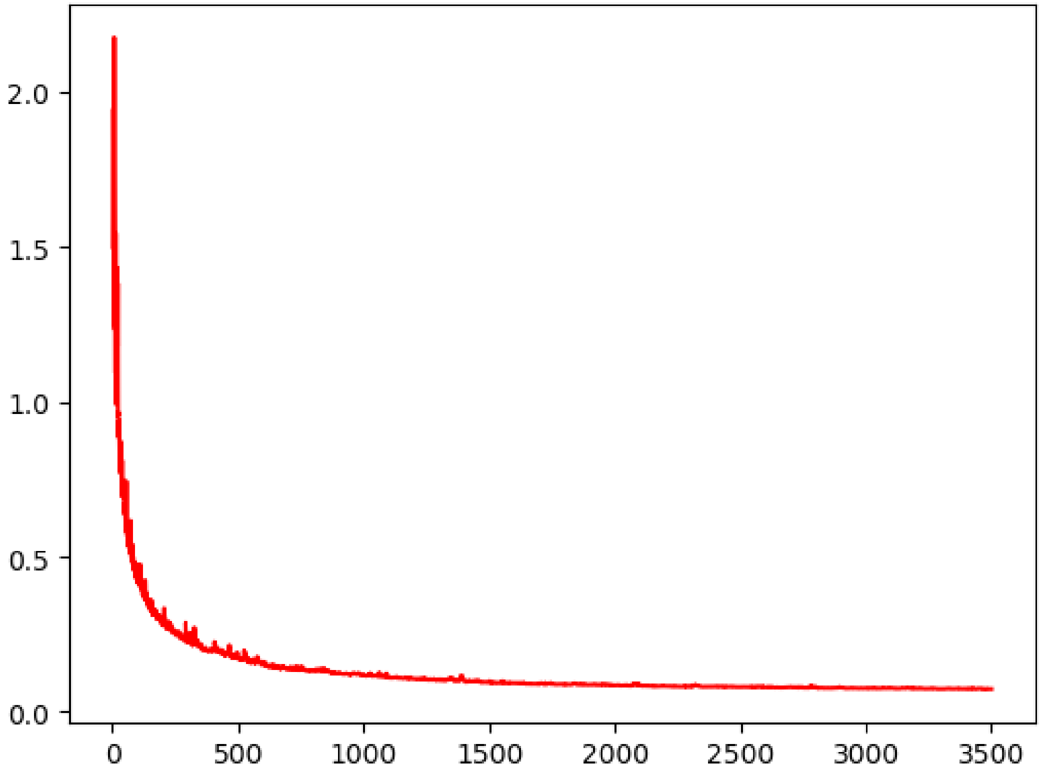
- **Loss Curve如下(Stochastic Gradient Descent):**



参数：learning rate = 0.1，λ = 0.001，迭代次数iteration = 5000

从曲线可以看出，SDG的代价函数随着迭代次数呈震荡式下降，不像Q3中的FBDG方法每次更新都朝着最优点的方向逼近。由于该方法每次随机选择一个样本进行参数更新，有可能出现方向是背离最优点的情况，即产生了如上图所示的震荡曲线。因此，该方法收敛时的迭代次数会有较大的变动，若每次随机选择的样本均向最优点逼近，则较快即可收敛；若每次随机选择的样本都背离最优点更新，则收敛较慢。

- **Loss Curve如下(Batched Gradient Descent):**



参数：batch=4，learning rate = 0.1，λ = 0.001，迭代次数iteration = 3500

从曲线可以看出，BDG的代价函数仍然随着迭代次数呈震荡形状下降，但与SDG相比则相对平滑(相对稳定)。通过每次选择batch条数据进行参数的更新，相对于每次选择一条更加稳定，保证在绝大多数情况下，发生的更新都是朝着最优点逼近的方向进行更新，因此在提高了稳定性的情况下，也加快了收敛所需的迭代次数。但由于batch作为一个超参数，batch的取值关系到了方法的稳定性和收敛速率，因此需要调batch这个参数，来得到训练该模型的最优取值。

（2）分别从三种方法的优缺点进行分析：

	pros	cons
Full Batch Gradient Descent	1、每次训练使用样本的所有数据，最终得到的也是全局最优解；2、从迭代次数上，FBDG的迭代次数较少；3、易于并行实现	1、当样本数据量过多时，由于每次训练需要用到所有数据，导致训练过程很慢
Stochastic Gradient Descent	1、每次仅随机选择样本中一条数据使用，训练速度快，尤其在样本量很大时，相比FBDG效率提升很大	1、SDG中噪音较FBDG多，使SDG并不是每次迭代都向着整体最优化方向，即相对准确度下降，不是全局最优；2、从迭代次数上，SDG的迭代次数较多；3、不易于并行实现
Batched Gradient Descent	该方法为以上两个方法的折衷：1、每次使用Batch个样本，训练速度相对FBDG得到很大的提升；2、噪声相对SDG较少，使得几乎每次迭代都向最优方向变化，保证相对较高准确率，同时BDG的迭代次数相对SDG也较少	1、新增加了超参数Batch，需要额外时间调整参数Batch，使得在选定的参数情况下，模型训练情况能够达到最优

Q5

（1）Full Batch Gradient Descent:模型参数learning rate=0.1,λ=0.01,Iteration=2000

得到结果如下：(注 loss curve见Question 3部分)

2000次迭代后，current_loss = 0.08111578397141636088

```
Iteration: 2000 , current_loss: 0.08111578397141636088
```

accuracy_test = 0.92847593582887699704

```
acc_test = 0.92847593582887699704
```

（2）Stochastic Gradient Descent:模型参数learning rate=0.1,λ=0.01,Iteration=5000

得到结果如下：(注 loss curve见Question 4部分)

5000次迭代后，current_loss = 0.08870243808683755948

```
Iteration: 5000 , current_loss: 0.08870243808683755948
```

accuracy_test = 0.91711229946524064349

```
acc_test = 0.91711229946524064349
```

（3）Batched Gradient Descent:模型参数batch=4,learning rate=0.1,λ=0.01,Iteration=3500

得到结果如下：(注 loss curve见Question 4部分)

5000次迭代后，current_loss = 0.07367182491700265123

```
Iteration: 3500 , current_loss: 0.07367182491700265123
```

accuracy_test = 0.92513368983957222635

```
acc_test = 0.92513368983957222635
```