# ­Assignment 2

本次实验实现了三种方法的线性分类，分别用least square model和perceptron来对数据点进行两类分类，还实现了logistic方法对文本进行四类分类。

## 0. 综述

### 0.1 结构

* source python file: 调用、运行其他文件中的class
  + source.py
* solution files: **每个文件均独立实现了一个部分，并各自封装成class**
  + least\_square\_model.py: 实现了least square model的两类数据点分类
  + perceptron.py: 实现了perceptron algorithm的两类数据点分类
  + logistic.py: 实现了logistic regression的四类文本分类

### 0.2 库

* 除了其他库以外，还使用了 argparse 来运行项目，需要通过以下命令行安装：
* pip install argparse

### 0.3 运行方法

* 在下文的每个关键部分，都会有获得对应结果的命令行，在项目的目录直接运行即可
* 打印help message：
* python source.py -h

## 1. Part I

### 1.1 least square model

#### 代码结构

* 代码都在class LSM中
* self.train()训练出和b的值；
* self.predict()进行预测；
* self.plot()画图；
* self.accuracy()计算正确率；
* self.run()运行整个流程

#### 运行

* python source.py --algorithm least\_square
* 输出训练结果和b的值，以及正确率

#### 实现

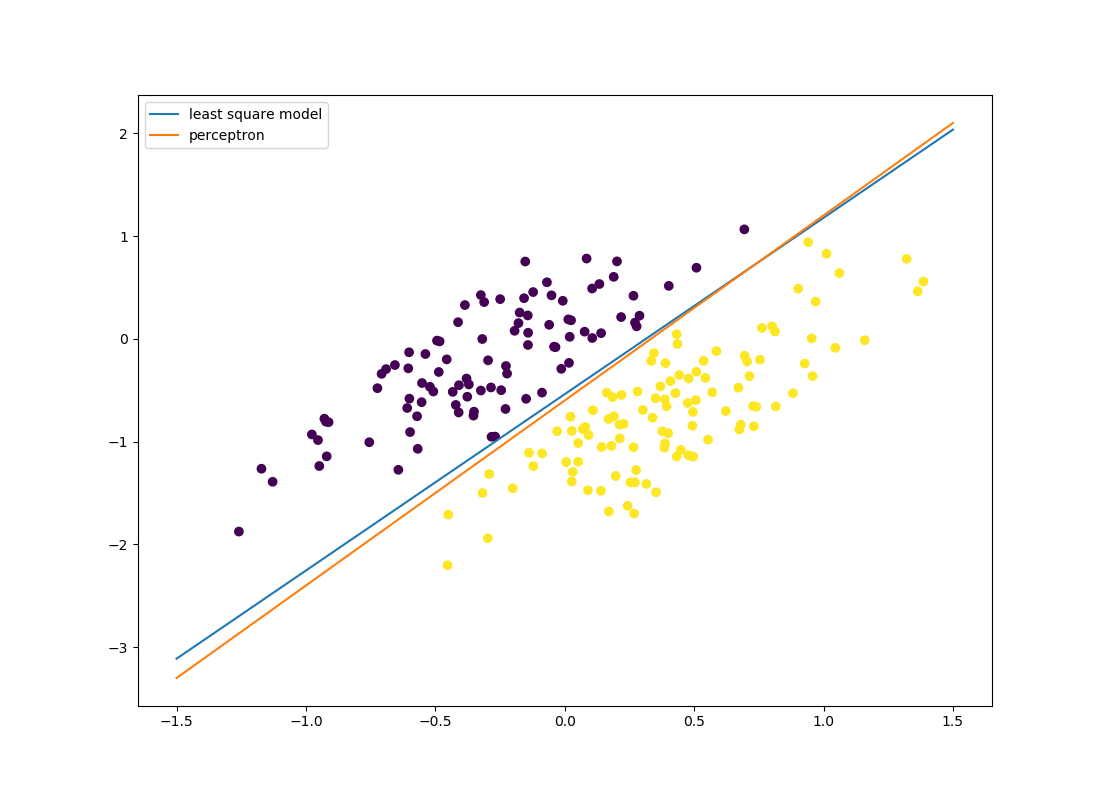
* 由于两类数据的label是True(1)和False(0)，因此两类数据的分界线的y值应为0.5，所以拟合曲线为。其中和b是需要求解的值
* 根据以下课本公式可以迅速计算出拟合向量。其中，是X的伪逆，可以通过numpy的linalg.pinv()函数快速求得。而就是y。
* 为了能更快地求解w和b，令，也就是在X矩阵右边添加一列1，再获得对应的伪逆，再根据3.35计算出，而且
* 进行预测的时候，对于数据点，可以计算出：

。

* 上述两个公式可以使用任意一个。当时，判为True；当时，判为False。

#### 结果

* 分界线方程为
* 正确率：1.0
* 含有分界线的图如下：



### 1.2 perceptron

#### 代码结构

* 代码都在class Perceptron中
* self.preprocess\_y()预处理y
* self.train()训练出和b的值；
* self.predict()进行预测；
* self.plot()画图；
* self.accuracy()计算正确率；
* self.run()运行整个流程

#### 运行

* python source.py --algorithm perceptron
* 输出训练结果和b的值，以及正确率

#### 实现

* 对y进行预处理，把False改为-1；另外True原本就对应1。因此两类数据的分界线的y值应为0，所以拟合曲线为。其中和b是需要求解的值
* 为了计算方便，同样令，也就是在X矩阵右边添加一列1，再计算出，那么可以根据获得w和b
* 初始值
* 当时，按照下方公式更新w，直至不再有变化，而，那么可以获得w和b
* 当进行预测的时候，对于数据点，可以计算出：

。

* 上述两个公式可以使用任意一个。当时，判为True；当时，判为False。

#### 结果

* 分界线方程为
* 正确率：1.0
* 含有分界线的图在上文，和least square model的分界线在同一个图中

## 2. Part II

### 2.1 代码结构

* 代码都在class Logistic中
* self.get\_vocabulary()获得词典
* self.preprocess\_X()预处理X
* self.preprocess\_y()预处理y
* self.softmax()计算softmax的值
* self.loss()计算loss
* self.graident()按照公式计算gradient
* self.check\_graident()检验gradient公式的正确性
* self.shuffle\_dataset()打乱数据集的顺序
* self.accuracy()计算正确率；
* self.train()训练模型，并获得loss、train\_accuracy、validate\_accuracy的list
* self.show()训练出模型，并画出loss、train\_accuracy、validate\_accuracy的变化
* self.show\_batch\_diff()训练出不同batch size的模型，并画出loss、train\_accuracy、validate\_accuracy的变化
* self.show\_lamb\_diff()训练出不同lamb的模型，并画出loss、train\_accuracy、validate\_accuracy的变化
* self.show\_alpha\_diff()训练出不同alpha的模型，并画出loss、train\_accuracy、validate\_accuracy的变化

### 2.2 处理data

* 遍历data中的text
  + 通过text.translate(str.maketrans("", "", string.punctuation))把punctuation去掉
  + 通过re.sub('[' + string.whitespace + ']', ' ', text)把whitespace去掉
  + 通过text.lower().split(' ')把text小写化，并以空格分割成一个个词
  + 储存每个text对应的words列表
* 新建大小为(data量\*字典量)的全0数据，每行代表一篇文章对应字典中的词的存在情况
  + 0表示字典中这个词在这篇文章中不存在
  + 1表示字典中这个词在这篇文章中存在
* 遍历每篇文章的词列表
  + 如果这个词在词典中，则对应位置标1

### 2.3 处理target

dataset = np.zeros([len(y\_data), self.categories])  
dataset[np.arange(len(y\_data)), y\_data] = 1

* 新建(target量\*类型量)的全0数组
* 把每行的target值对应的位置标为1

### 2.4 梯度公式推导

* Loss的标量公式如下：
* 其中：
* 通过链式求导来求解loss对w的偏导：
* 其中：
* 那么在求偏导前首先要求出：
* 另外还有：
* 现在可以代入求解loss对w的偏导：
* 通过链式求导来求解loss对b的偏导，首先要求出
* 带入求解loss对b的偏导：
* 各用一行代码实现numpy的向量化运算

w\_gradient = 2 \* self.lamb \* w - X\_train.T @ (y\_train - y\_pred) / X\_train.shape[0]  
b\_gradient = - np.sum(y\_train - y\_pred, axis=0) / X\_train.shape[0]

### 2.5 L2正则化

* 根据上文的loss公式，可以看出使用了L2正则化，正则化只包含了w，而没有b。另外，正则化的系数为
* 公式中正则化只包含了w，而没有b，因为：当模型过于复杂的时候，会出现over-fitting的情况，在训练集上达到很高的正确率，但是泛化能力却降低了。因此需要正则化，给模型的复杂度加上penalty，当模型越复杂时，penalty越大，这样可以降低over-fitting的程度。而对于线性拟合，决定模型复杂度的是w。当w的L2很大的时候，预测结果的变化波动很大，使得复杂度变高，而b的大小只会使数据整体偏移，而不会对结果的波动变化有影响，也不会使模型复杂度有影响。因此不需要正则化b。

### 2.6 检验梯度计算的正确性

#### 运行

* python source.py --algorithm logistic --n check
* 检验梯度计算的正确性

#### 原理

* 按照导数的定义计算梯度：
* 如果导数定义结果和公式计算结果的任意对应值的差error，小于某个给定的小阈值，则可以认为梯度计算正确

#### 实现

* 由于计算量很大，因此首先shuffle一下数据集，取出前8条数据，并循环7个epoch，并计算梯度用于验证
* 在每个epoch中，用公式计算出w、b的梯度，并通过导数定义计算出w、b的梯度，把每个对应值的差记录下来
* 如果最大的差值小于给定的小阈值，则说明梯度公式是正确的

#### 结果

* 在导数定义公式中的的情况下，最大差值可以小于阈值，说明梯度公式是正确的

#### 思考：为什么不用数值计算的方式求梯度？

* 原因：通过公式计算快很多
* 公式计算快：可以通过numpy实现向量化运算
* 导数定义计算慢：如果要向量化运算，则每次运算需要的空间（N为数据量，K为类型数），空间代价是巨大的，计算耗时也比较长；如果不用向量化运算，需要用两层for循环，每次求解梯度时需要次导数运算，计算耗时长
* 因此使用公式计算梯度

### 2.7 logistic regression

#### 运行

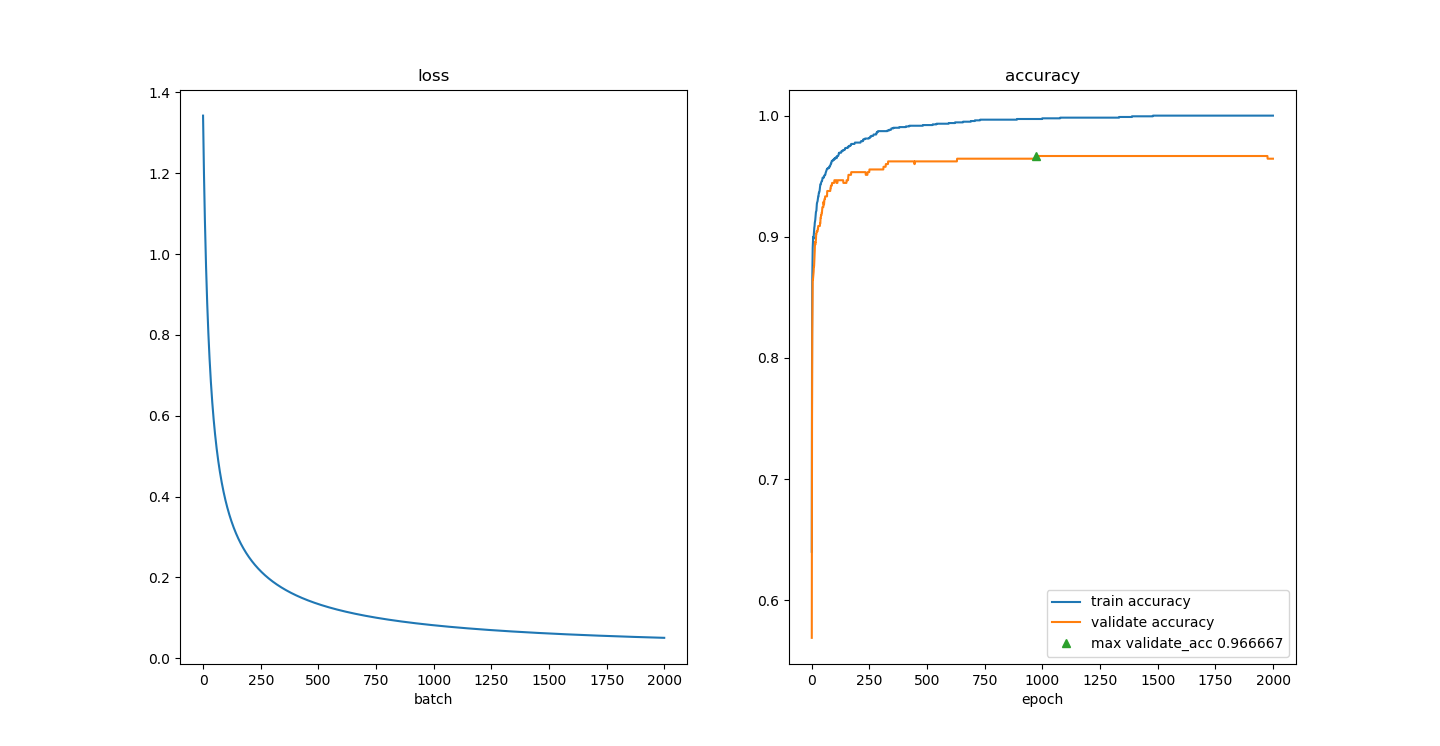
* python source.py --algorithm logistic --n run
* 输出每个epoch的在训练集上的正确率和检验集上的正确率
* 最后用达到最高检验集正确率的w、b，在测试集上计算正确率，并输出

#### 实现

* **把训练集的4/5作为训练集，剩下的作为­­­检验集，在后面的模型训练中也是如此划分**
* 正则化系数为1e-4，学习率为0.1，运行的epoch数为2000，是full batch的
* 初始化w和b为0
* 循环2000个epoch，在每个epoch中先shuffle训练集，再计算梯度，更新w、b，记录loss、训练集上的正确率、检验集上的正确率
* 最后用达到最高检验集正确率的w、b，在测试集上计算正确率，并输出
* 画出loss、训练集上的正确率、检验集上的正确率的变化曲线

#### 结果

* 训练集上的正确率达到了1，检验集最高正确率为0.966667
* 达到最高检验集正确率的w、b在测试集上的正确率为0.925134

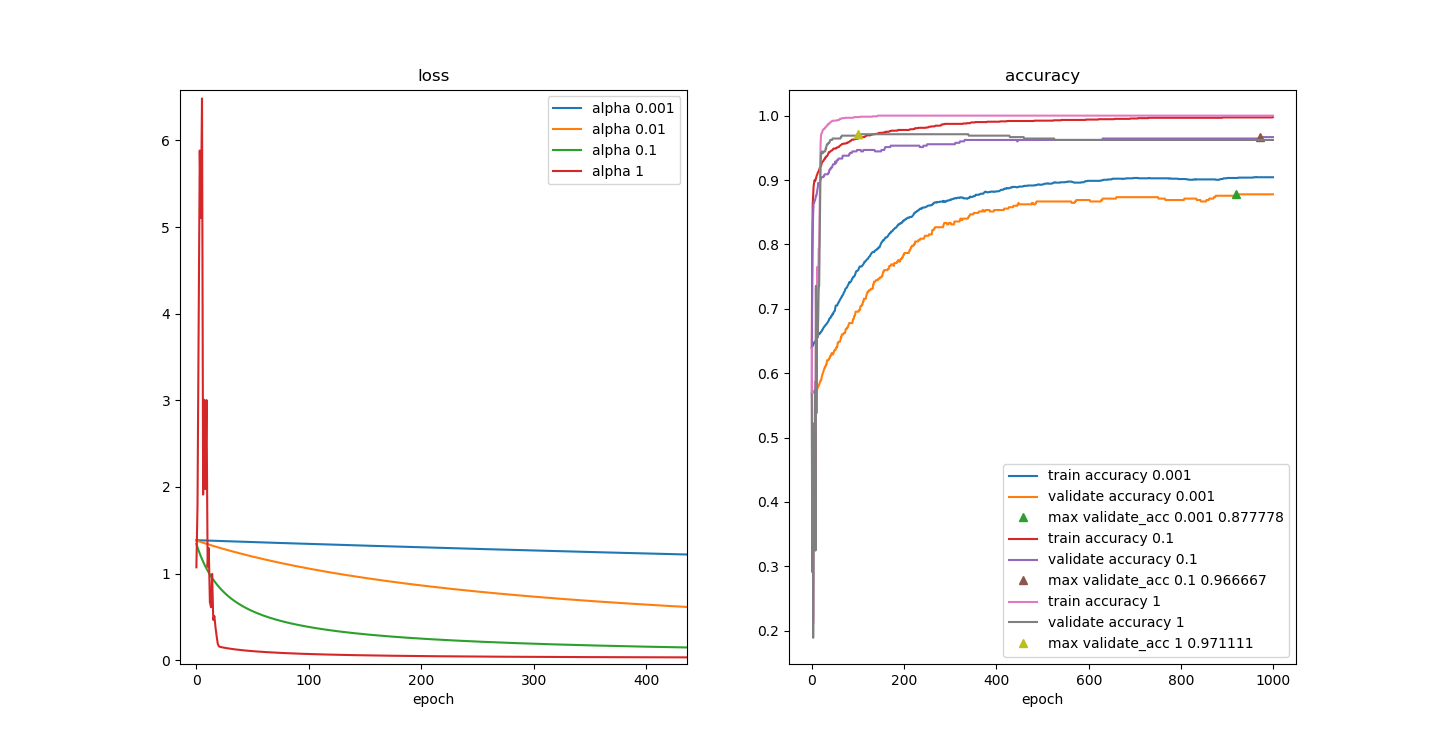


### 2.8 如何决定学习率

#### 运行

* python source.py --algorithm logistic --n alpha
* 训练4个不同学习率的模型，学习率为0.001、0.01、0.1、1
* 输出每个epoch的在训练集上的正确率和检验集上的正确率
* 最后用达到最高检验集正确率的w、b，在测试集上计算正确率，并输出
* 画出loss、训练集上的正确率、检验集上的正确率的变化曲线

#### 结果



#### 分析

* 通过上述loss值变化，可以发现，学习率越大，loss下降的越快，但是loss的变化也更不稳定；而训练集和检验集的正确率也上升的更快
* 如果学习率过大，则可能无法收敛

#### 决定学习率的策略

* 学习率要尽量大，使得收敛更快；但是学习率不能过大，而使得不能收敛
* 可以先在一个较小的数据集上运行多次，获得一个大致的学习率范围，再在这个范围中，用全部数据集上运行多次，获得更准确的学习率

### 2.9 什么时候结束训练

#### 停止条件

* 前一周期所有的都太小，小于某个指定的阈值
* 前一周期错误率小于某个指定的阈值
* 超过某个指定的epoch数

#### 指定epoch数

* 我使用的方法是指定epoch数，取为1000
* 如何指定epoch数？epoch数越小，训练所需时间越短；但是epoch数要足够大，能让validation集上的正确率达到最大值，并且能显示出由于overfitting而下降的趋势，这样才能确认获得最好的validation集正确率

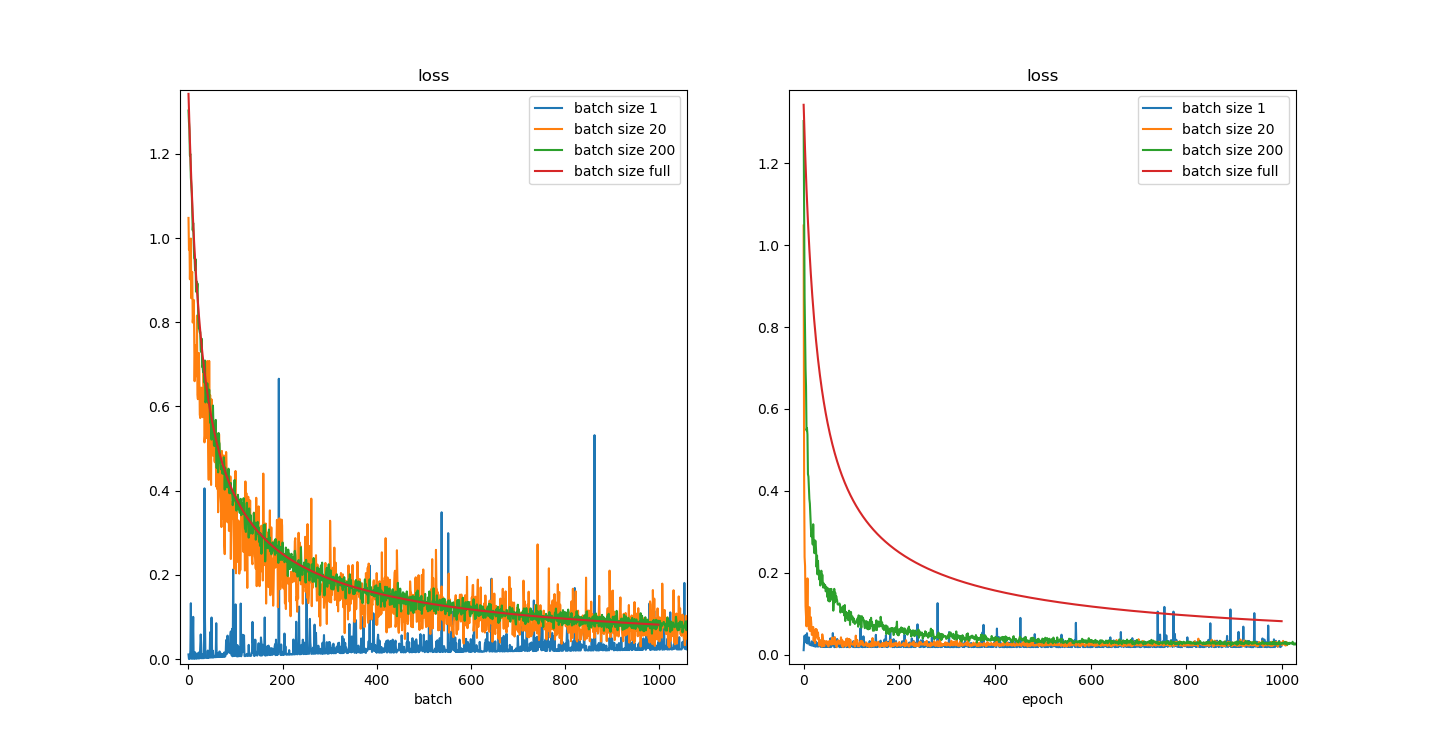
### 2.9 不同的batch size

#### 运行

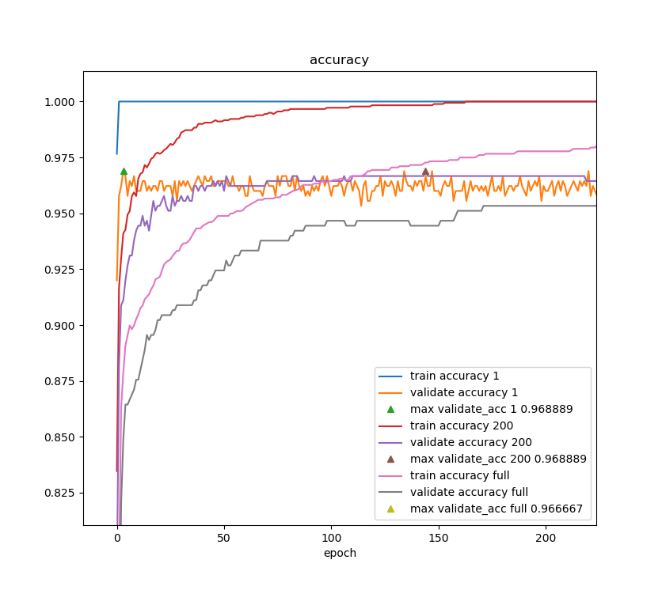
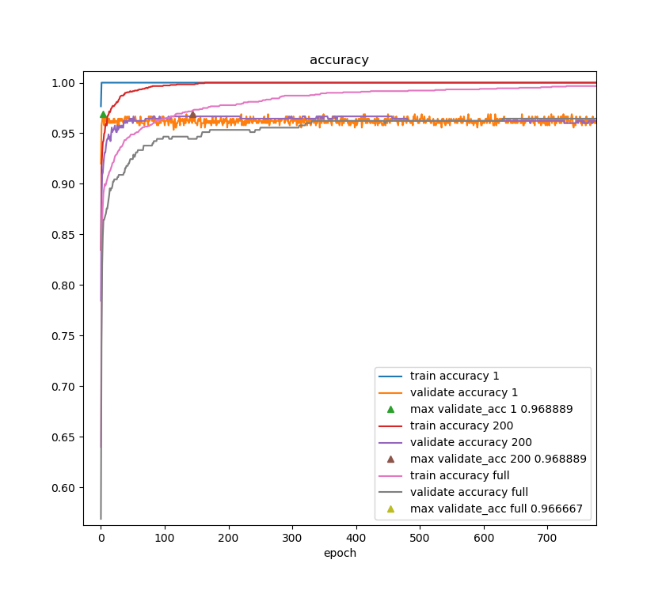
* python source.py --algorithm logistic --n batch
* 训练4个不同batch size的模型，batch size为1、20、200、full batch
* 在训练模型的过程中输出对应的正确率
* 最后画图

#### 结果

* 左图为统一运行的batch数量的结果。右图为统一运行的epoch数量的结果



* 左图为不同batch size在训练集、检验集上的正确率变化；右图为左图的放大。



#### 分析

* batch size越小，loss的下降方向越不准确，loss的波动越大
* 除了batch size为1的情况外，loss的下降大致趋势是一样的
* batch size越小，loss下降的越快，收敛得更快

#### 优缺点

* batch size较大：loss下降的方向更准确，波动小，但loss下降的慢，所用的数据量大
* batch size较小：loss下降的波动大，而loss下降更快，所用的数据量少
* 而mini batch则是两种情况的折中，loss下降准确，波动小，且loss下降的快，收敛更快

### 2.10 不同的lambda

#### 运行

* python source.py --algorithm logistic --n lambda
* 训练4个不同lambda的模型，为1e-6、1e-4、1e-2、1
* 在训练模型的过程中输出对应的正确率
* 最后画图

#### 结果

* 左图为不同lambda的loss变化。右图训练集、检验集上正确率的变化

### 

#### 分析

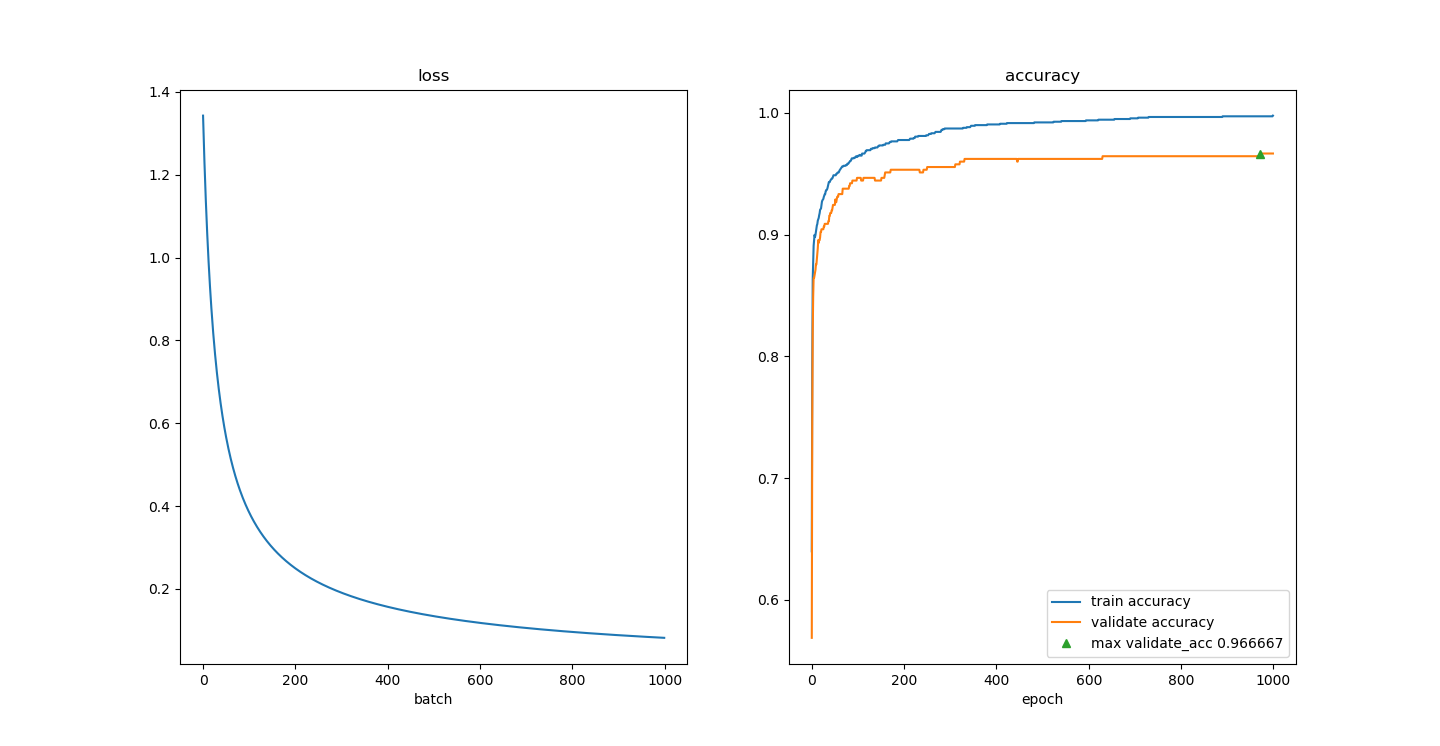
* lambda越大，penalty越大，loss下降慢、下降量越少，准确率越快达到最大值，值越低
* lambda越小，panalty越小，loss下降快，下降量多，准确率更高
* 但是lambda越小，越容易over-fitting

### 2.10 不同的batch size的测试集结果

* 以下左图均为loss变化，右图均为训练集、检验集上的正确率变化

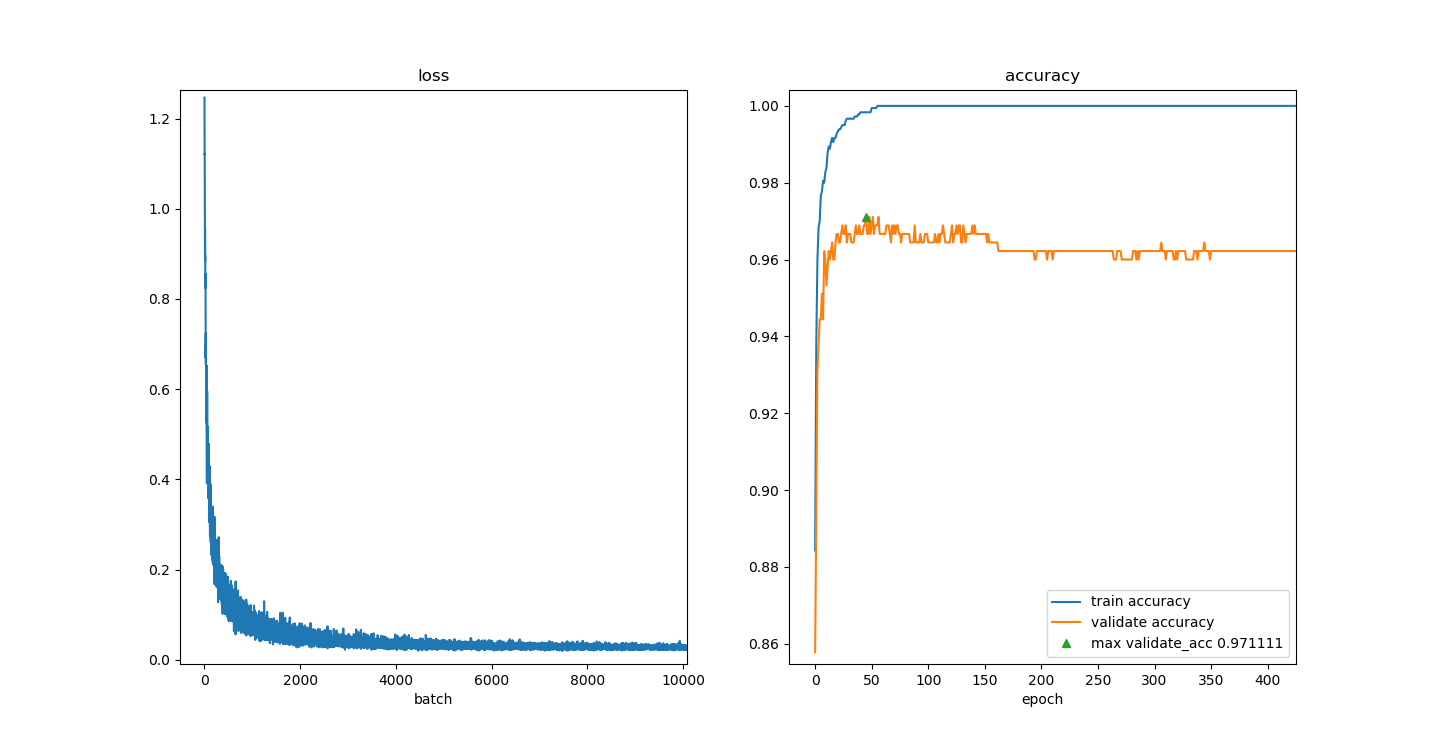
#### full batch

* 测试集正确率：0.925134



#### mini batch

* 测试集正确率：0.924465



#### full batch

* 测试集正确率：0.922460

