机器学习 作业1

1 介绍

2 线性模型与梯度下降

2.1 特征归一化

- 见 `start_code.py`

2.2 目标函数与梯度

2.2.1 1

$$
\begin{aligned}
J(\theta) &= \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i)-y_i)^2+\lambda\theta^T\theta \\
&= \frac{1}{m}\begin{bmatrix}\theta^T x_1 - y_1 & \theta^T x_2 - y_2 & \cdots & \theta^T x_m - y_m\end{bmatrix}\begin{bmatrix}\theta^T x_1 - y_1 \\ \theta^T x_2 - y_2 \\ \cdots \\ \theta^T x_m - y_m\end{bmatrix}+\lambda\theta^T\theta \\
&= \frac{1}{m}(X\theta-y)^T(X\theta-y)+\lambda\theta^T\theta
\end{aligned}
$$

2.2.2 2

- 见 `start_code.py`

2.2.3 3

$$
\begin{aligned}
\nabla J(\theta) = \frac{\partial J(\theta)}{\partial\theta} &= \frac{1}{m}\frac{\partial((X\theta-y)^T(X\theta-y))}{\partial\theta}+\lambda\frac{\partial(\theta^T\theta)}{\partial\theta} \\
&= \frac{2}{m}X^T(X\theta-y)+2\lambda\theta
\end{aligned}
$$

2.2.4 4

- 见 `start_code.py`

2.2.5 5

- 见 `start_code.py`
- 取$\theta = 0^{d+1}, \lambda = 1$做验证，结果通过

```
> python start_code.py
loading the dataset
Split into Train and Test
Scaling all to [0, 1]
dis = 3.4443880862942937e-13
True
```

2.2.6 6

当偏置项被置为较大常数时，θ中与偏置项对应的分量$\theta_{(d+1)}$相应地为一较小量，在正则化函数中，有

$$\frac{\partial(\lambda\theta^T\theta)}{\partial\theta_{(d+1)}} = 2\lambda\theta_{(d+1)} \to 0$$

这样就降低了正则化对偏置项的影响

2.3 梯度下降

2.3.1 1

略去二阶及以上项，可以得到：

$$J(\theta+\eta h) - J(\theta) = \nabla J(\theta)^T(\theta+\eta h-\theta) = \eta\cdot\nabla J(\theta)^T h$$

当h与$\nabla J(\theta)$方向相反时，目标函数下降速度最快

2.3.2 2

取$h=-\nabla J(\theta)$即可，即：

$$\theta_{i+1} = \theta_i - \eta\cdot\nabla J(\theta_i)$$

2.3.3 3

- 见`start_code.py`

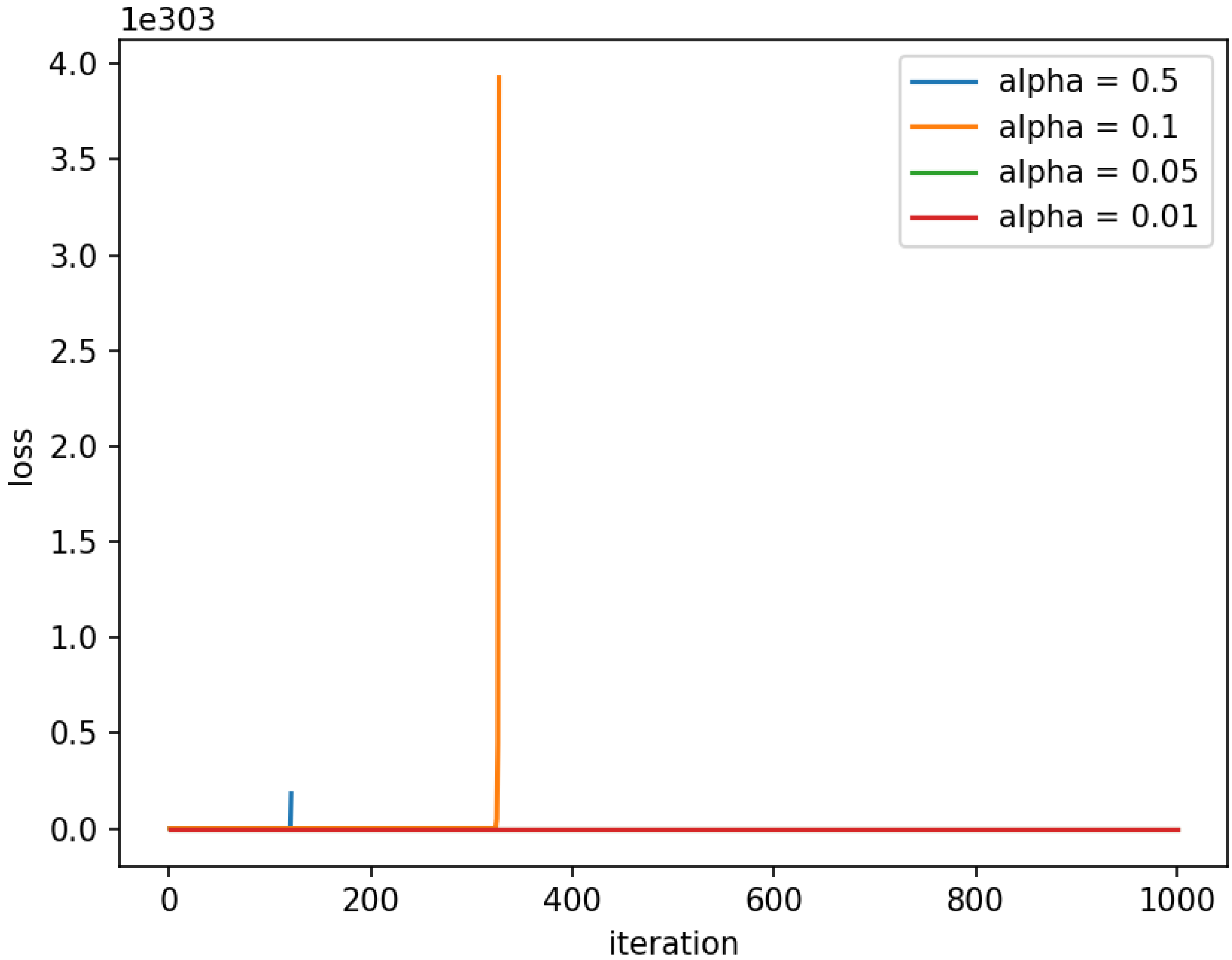
2.3.4 4

2.3.4.1 source code

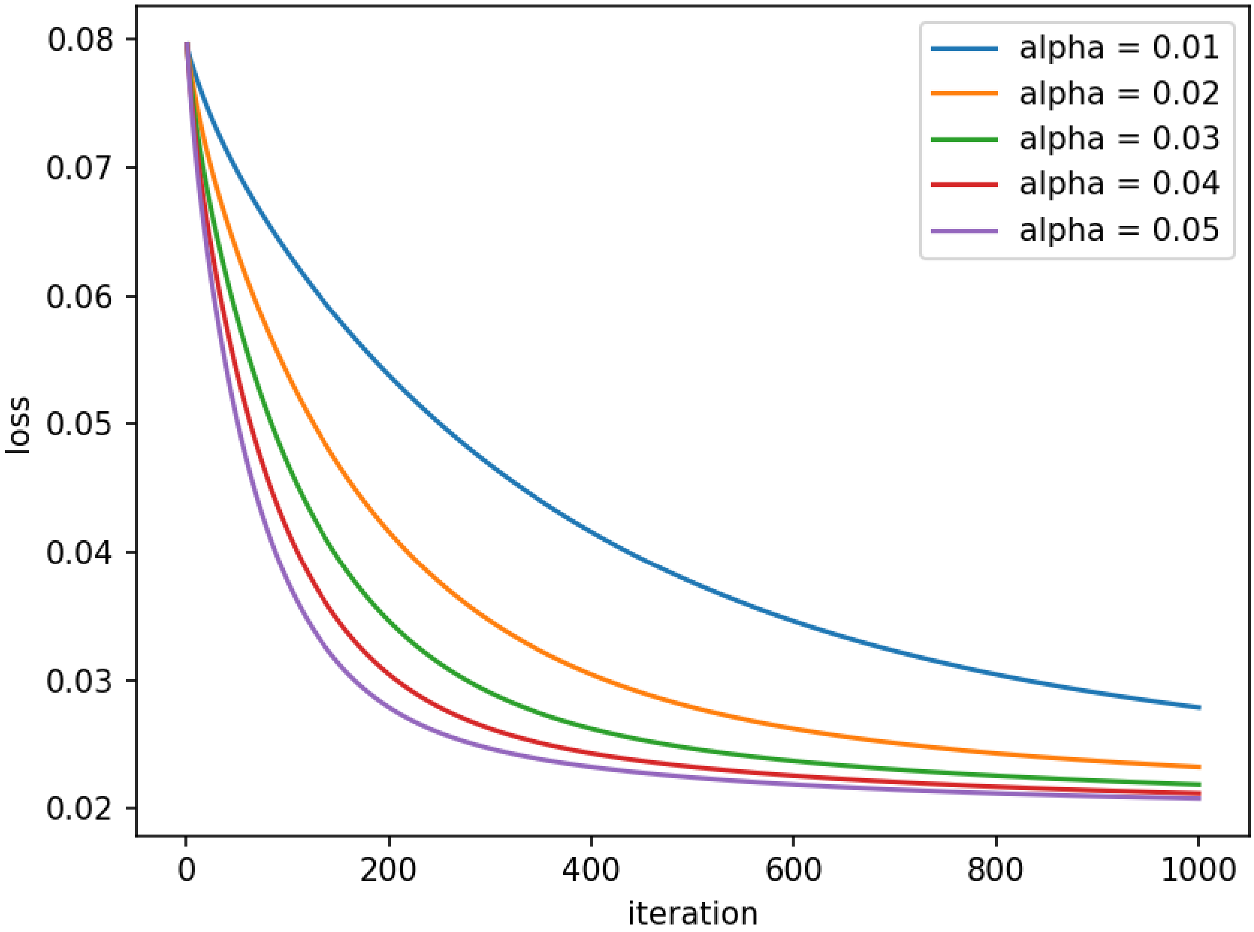
`main`函数中：

```
import matplotlib.pyplot as plt
num_iter = 1000
x_axis = np.arange(1, num_iter + 1, 1)
alpha_list = [0.5, 0.1, 0.05, 0.01]

for alpha in alpha_list:
    theta_hist_train, loss_hist_train =  batch_grad_descent(X_train, y_train, 0, alpha=alpha,
num_iter=num_iter)
    plt.plot(x_axis, loss_hist_train, label='alpha = ' + str(alpha))
    plt.xlabel('iteration')
    plt.ylabel('loss')
plt.legend()
plt.show()
```

2.3.4.2 训练结果



结果显示，步长取0.5，0.1时会导致发散；取0.05，0.01时会收敛；进一步研究步长取[0.01, 0.05]时的情形：



结果显示，步长取0.05时收敛速度最快；且此后随着步长减小，收敛速度变慢

2.4 随机梯度下降

2.4.1 1

记

$$X_n = [x_{i_1} \quad x_{i_2} \quad \cdots \quad x_{i_n}]^T$$
$$y_n = [y_{i_1} \quad y_{i_2} \quad \cdots \quad y_{i_n}]^T$$

则由2.2.3中结论，可以得到

$$\begin{aligned}\nabla J_{SGD}(\theta) &= \frac{2}{n} X_n^T (X_n \theta - y_n) + 2\lambda\theta \\ &= \frac{2}{n} \sum_{k=1}^{n} x_{i_k} (x_{i_k}^T \theta - y_{i_k}) + 2\lambda\theta\end{aligned}$$

2.4.2 2

对$\nabla J_{SGD}(\theta)$求期望：

$$E_{i_1, i_2, \cdots, i_n}[\nabla J_{SGD}(\theta)] = \frac{2}{n} \sum_{k=1}^{n} E(x_{i_k} x_{i_k}^T \theta - x_{i_k} y_{i_k}) + 2\lambda\theta$$

因为i_k是从$\{1, 2, \cdots, m\}$中独立同分布采样

$$
\begin{aligned}
E(x_{i_k}x_{i_k}^T\theta - x_{i_k}y_{i_k}) &= E(x_{i_k}x_{i_k}^T)\theta - E(x_{i_k}y_{i_k}) \\
&= (\frac{1}{m}\sum_{i=1}^{m} x_i x_i^T)\theta - (\frac{1}{m}\sum_{i=1}^{m} x_i y_i) \\
&= \frac{1}{m}\sum_{i=1}^{m}(x_i x_i^T\theta - x_i y_i)
\end{aligned}
$$

故有

$$
\begin{aligned}
E_{i_1,i_2,\cdots,i_n}[\nabla J_{SGD}(\theta)] &= \frac{2}{n}\sum_{k=1}^{n} E(x_{i_k}x_{i_k}^T\theta - x_{i_k}y_{i_k}) + 2\lambda\theta \\
&= \frac{2n}{n}\cdot\frac{1}{m}\sum_{i=1}^{m}(x_i x_i^T\theta - x_i y_i) + 2\lambda\theta \\
&= \frac{2}{m}X^T(X\theta - y) + 2\lambda\theta \\
&= \nabla J(\theta)
\end{aligned}
$$

2.4.3 3

- 见`start_code.py`

2.4.4 4

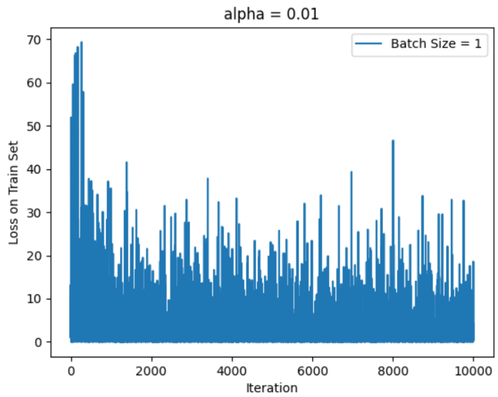
- 固定步长为0.01，此时梯度下降较为稳定，利于模型收敛
- 尝试batch size分别为1，2，4，8，16，32

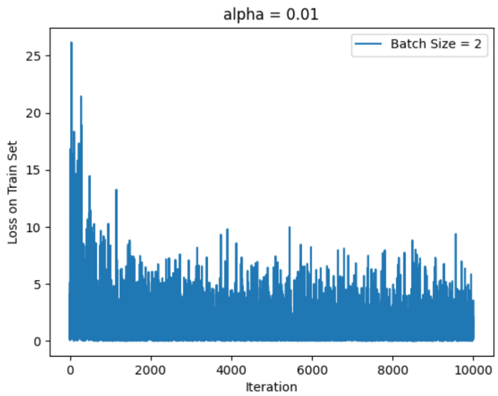
2.4.4.1 source code

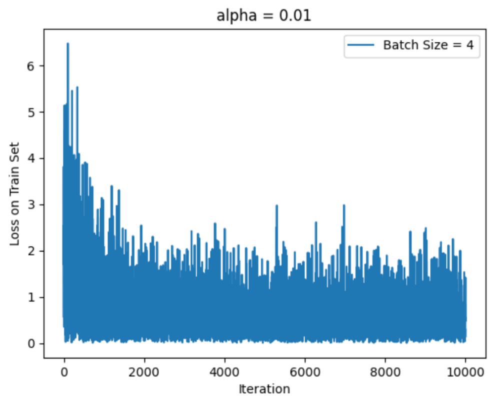
`main`函数中，有

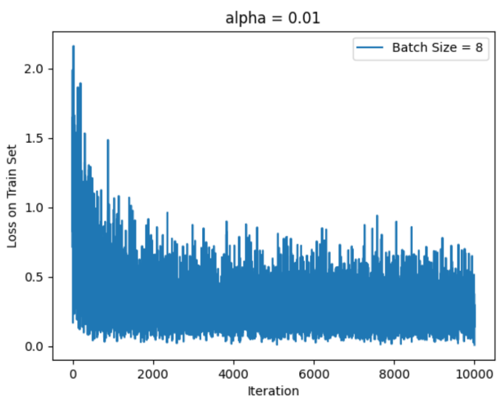
```
import matplotlib.pyplot as plt
num_iter = 10000
x_axis = np.arange(1, num_iter + 1, 1)
alpha = 0.01
batch_size_list = [1,2,4,8,16,32]

for batch_size in batch_size_list:
    theta_hist_train, loss_hist_train, validation_hist = stochastic_grad_descent(X_train,
y_train, X_test, y_test, lambda_reg= 0, alpha=alpha, num_iter=num_iter, batch_size=batch_size)
    plt.plot(x_axis, loss_hist_train, label='Batch Size = ' + str(batch_size))
    plt.ylabel('Loss on Train Set')
    plt.xlabel('Iteration')
    plt.legend()
    plt.title("alpha = "+str(alpha))
    plt.savefig('imgs/sgd_'+'bs='+str(batch_size)+'.png')
    plt.clf()
```

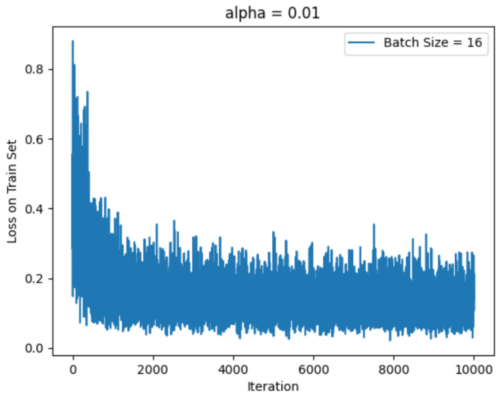
2.4.4.2 训练结果

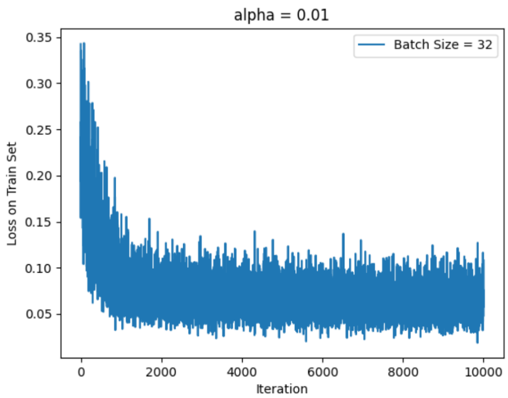

alpha = 0.01
Batch Size = 1
Loss on Train Set
Iteration
0
10
20
30
40
50
60
70
0
2000
4000
6000
8000
10000


alpha = 0.01
Batch Size = 2
Loss on Train Set
Iteration
0
5
10
15
20
25
0
2000
4000
6000
8000
10000


alpha = 0.01
Batch Size = 4
Loss on Train Set
Iteration
0
1
2
3
4
5
6
0
2000
4000
6000
8000
10000


alpha = 0.01
Batch Size = 8
Loss on Train Set
Iteration
0.0
0.5
1.0
1.5
2.0
0
2000
4000
6000
8000
10000





由此可以得到以下结论:

- batch size为1，2，4时，训练曲线并未明显收敛，有较大的震荡；而batch size为8，16，32时，训练曲线有较为明显的收敛趋势，且随着batch size增大收敛所需迭代次数减小
- batch size更大时，对全批量损失函数的梯度估计更加精准，即下降方向更加精准；从而小批量损失函数震荡小，所需迭代次数少
- batch size线性增大时，获得收敛所需迭代次数减小的收益是小于线性的，即batch size过大时也并不能带来训练速度的提升

2.5 模型选择

2.5.1 梯度下降

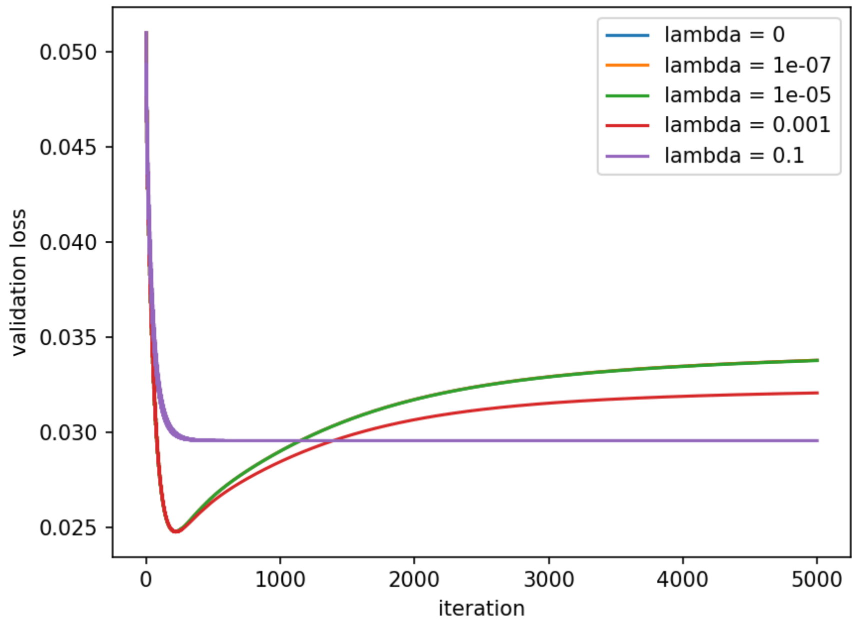
- 首先使用梯度下降来考察正则化系数的影响
- `main` 函数中

```
num_iter = 5000
alpha = 0.05
lambda_list = [0, 1e-7, 1e-5, 1e-3, 0.1, 1, 10, 100]
for lambda_reg in lambda_list:
    theta_hist_train, loss_hist_train =  batch_grad_descent(X_train, y_train,
lambda_reg=lambda_reg, alpha=alpha, num_iter=num_iter)
    # 计算验证集上的损失函数
    validation_loss = compute_regularized_square_loss(X_test, y_test, theta_hist_train[-1],
0)
    print("lambda = " + str(lambda_reg) + " " + "validation loss = " +
str(validation_loss))
```

- 得到结果

λ	Validation Loss
0	0.0338
1e-7	0.0338
1e-5	0.0338
1e-3	0.0321
0.1	0.0296
1	not convergent
10	not convergent
100	not convergent

- 可以看到，L_2正则化系数λ增大时，能有效抑制模型的过拟合，降低模型在验证集上的均方误差；但是系数过大时，会引起模型的震荡，使模型不收敛；以下是不同正则化系数下，模型随着迭代过程在验证集上的误差变化曲线：



2.5.2 随机梯度下降

- 接下来使用随机梯度下降进行研究
- `main` 函数中

```
num_iter = 10000
alpha = 0.01
batch_size = 16
lambda_list = [0, 1e-7, 1e-5, 1e-3, 0.1, 1, 10, 100]
for lambda_reg in lambda_list:
    theta_hist_train, loss_hist_train, validation_hist = stochastic_grad_descent(X_train,
y_train, X_test, y_test, lambda_reg= lambda_reg, alpha=alpha, num_iter=num_iter,
batch_size=batch_size)
    print("lambda = " + str(lambda_reg) + " " + "validation loss = " +
str(validation_hist[-1]))
```

- 得到结果

λ	**Validation Loss**
0	0.0316
1e-7	0.0319
1e-5	0.0322
1e-3	0.0313
0.1	0.0292
1	0.0449
10	0.0499
100	not convergent

- 大体上规律与梯度下降中得到的一致

3 支持向量机

3.1 次梯度

3.1.1 1

在 $1 - yw^T x >= 0$ 时，可以直接取梯度：

$$g = \frac{\partial J(w)}{\partial w} = -yx$$

此时

$$J(w_1) - J(w_0) - g^T(w_1 - w_0) \geqslant 1 - yw_1^T x - 1 + yw_0^T x + yx^T(w_1 - w_0) = 0$$

在 $1 - yw^T x < 0$ 时，取次梯度$g = 0$即可

此时

$$J(w_1) - J(w_0) - g^T(w_1 - w_0) \geqslant 0 - 0 + 0 \cdot (w_1 - w_0) = 0$$

故可以写出Hinge Loss的次梯度

$$\partial J(w) = \begin{cases} -yx & 1 - yw^T x \geqslant 0 \\ 0 & 1 - yw^T x < 0 \end{cases}$$

3.1.2 2

假设存在$x_1, x_2 \in R^d, x_2 - x_1 > 0$，以及$\lambda \in (0, 1)$，使得

$$\lambda f(x_1) + (1-\lambda) f(x_2) < f(\lambda x_1 + (1-\lambda) x_2)$$

取$f(x)$在$\lambda x_1 + (1-\lambda)x_2$处的次梯度$g$，则有

$$\begin{gathered}
\begin{cases} g^T(x_2 - \lambda x_1 - (1-\lambda)x_2) = \lambda g^T(x_2 - x_1) \leqslant f(x_2) - f(\lambda x_1 + (1-\lambda)x_2) \\ g^T(x_1 - \lambda x_1 - (1-\lambda)x_2) = (\lambda - 1) g^T(x_2 - x_1) \leqslant f(x_1) - f(\lambda x_1 + (1-\lambda)x_2) \end{cases} \\
\Rightarrow (\lambda(1-\lambda) + \lambda(\lambda - 1)) g^T(x_2 - x_1) \leqslant \lambda f(x_1) + (1-\lambda) f(x_2) - f(\lambda x_1 + (1-\lambda)x_2) < 0 \\
\Rightarrow 0 < 0
\end{gathered}$$

矛盾！故可以证明f 为凸

3.2 感知机

3.2.1 1

感知损失可以视作3.1中的合页损失函数处理，由此写出次梯度:

$$
\begin{aligned}
\partial\ell(y_i,\omega^T x_i) &= \partial max\{0,-y_i\omega^T x_i\} \\
&= \begin{cases} -y_i x_i & -y_i w^T x_i \geqslant 0 \\ 0 & -y_i w^T x_i < 0 \end{cases}
\end{aligned}
$$

依据此次梯度，使用固定步长1，可以写出如下SSGD算法：

Algorithm: SSGD

输入：训练集 $(x_1,y_1)\cdots(x_n,y_n)\in R^d\times\{-1,1\}$

$\omega^{(0)}=(0,\cdots,0)\in R^d$

$k=0$ # 迭代次数

batch_size = m # 批大小

repeat

从数据集中随机抽取m对数据:$(x_{k_1},y_{k_1})\cdots(x_{k_m},y_{k_m})$

for i = 1,2,···,m

if$(-y_i x_i^T\omega^{(k)}\leq 0)$

$w^{(k+1)}=w^{(k)}-(-y_{k_i}x_{k_i})=w^{(k)}+y_{k_i}x_{k_i}$

else

$w^{(k+1)}=w^{(k)}-0=w^{(k)}$

end if

$k=k+1$

end for

until $(k==k_{max})$

return $w^{(k)}$

- 次梯度下降算法中数据对w的作用过程与感知机算法完全相同；即在迭代充分的基础上，全批次的次梯度下降算法与感知机算法等价
- 而2.4中已经证明，随机次梯度与全批量次梯度的最终训练效果等价；即上面所述的SSGD算法与感知机算法等价

3.2.2 2

只需证明对于迭代中任何$w^{(k)}$均为输入数据的线性组合

假设返回最终结果时，$k=k_{max}$，则需要证明

$$\forall k\in 1,2,\cdots,k_{max}，\text{有}w^{(k)}=\sum_{i=1}^{n}\alpha_{k_i}x_i$$

采用归纳法证明

- 对于$k=1$

$$w^{(1)}=\sum_{i=1}^{n}I_{\{-y_i x_i^T\omega^{(0)}\leq 0\}}y_i x_i=\sum_{i=1}^{n}\alpha_{1_i}x_i$$

其中$I_{\{-y_i x_i^T\omega^{(k)}\}}$为示性变量

- 对于$k=t>1$

依据归纳假设，

$$w^{(t-1)} = \sum_{i=1}^{n} \alpha_{t-1_i} x_i$$

而

$$w^{(t)} = w^{(t-1)} + \sum_{i=1}^{n} I_{\{-y_i x_i^T \omega^{(t-1)} \le 0\}} y_i x_i$$
$$= \sum_{i=1}^{n} \alpha_{t-1_i} x_i + \sum_{i=1}^{n} I_{\{-y_i x_i^T \omega^{(t-1)} \le 0\}} y_i x_i = \sum_{i=1}^{n} \alpha_{t_i} x_i$$

故最终返回的w亦为输入数据的线性组合，证毕。

3.3 软间隔支持向量机

3.3.1 1

如下：

$$L(w,b,\alpha,\xi,\mu) = \frac{1}{2}||w||_2^2 + \frac{1}{m\lambda}\sum_{i=1}^{m}\xi_i + \sum_{i=1}^{m}\alpha_i(1-\xi_i-y_i(w^Tx_i+b)) - \sum_{i=1}^{m}\mu_i\xi_i$$
$$\alpha_i \ge 0, \mu_i \ge 0, i = 1, 2 \cdots m$$

3.3.2 2

推导如下：

$$\begin{cases} \frac{\partial L}{\partial w} = 0 \Rightarrow w = \sum_{i=1}^{m} \alpha_i y_i x_i \\ \frac{\partial L}{\partial b} = 0 \Rightarrow \sum_{i=1}^{m} \alpha_i y_i = 0 \\ \frac{\partial L}{\partial \xi_i} = 0 \Rightarrow \frac{1}{m\lambda} = \alpha_i + \mu_i \end{cases}$$

从而可以得到对偶形式:

$$\max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \alpha_i\alpha_j y_i y_j (x_i \cdot x_j)$$
$$s.t. \sum_{i=1}^{m} \alpha_i y_i = 0$$
$$0 \le \alpha_i \le \frac{1}{m\lambda}, 1 \le i \le m$$

3.3.3 3

- 原问题

$$\min_{w,b,\xi} \frac{\lambda}{2}||w||_2^2 + \frac{1}{m}\sum_{i=1}^{m}\xi_i$$
$$s.t. \quad y_i(w^T\Phi(x_i)+b) \ge 1-\xi_i$$
$$\xi_i \ge 0,\ 1 \le i \le m$$

- 对偶问题

$$\max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \alpha_i\alpha_j y_i y_j k(x_i, x_j)$$
$$s.t. \sum_{i=1}^{n} \alpha_i y_i = 0$$
$$0 \le \alpha_i \le \frac{1}{m\lambda}, 1 \le i \le m$$

3.3.4 4

由3.1中合页损失函数的次梯度，有

$$\partial|_w \max\{0,1-y_i(w^Tx_i+b)\}=\begin{cases}-y_ix_i & y_iw^Tx_i+b<1\\ 0 & y_iw^Tx_i+b\geq 1\end{cases}$$
$$\partial|_b \max\{0,1-y_i(w^Tx_i+b)\}=\begin{cases}-y_i & y_iw^Tx_i+b<1\\ 0 & y_iw^Tx_i+b\geq 1\end{cases}$$

故可以写出

$$\begin{aligned}\partial J_i|_w &= \lambda w+\partial|_w \max\{0,1-y_i(w^Tx_i+b)\}\\ &=\begin{cases}\lambda w-y_ix_i & y_iw^Tx_i+b<1\\ \lambda w & y_iw^Tx_i+b\geq 1\end{cases}\\ \partial J_i|_b &= \lambda w+\partial|_w \max\{0,1-y_i(w^Tx_i+b)\}\\ &=\begin{cases}-y_i & y_iw^Tx_i+b<1\\ 0 & y_iw^Tx_i+b\geq 1\end{cases}\end{aligned}$$

3.3.5 5

Algorithm: SSGD

输入：训练集 $(x_1,y_1)\cdots(x_m,y_m)\in R^d\times\{-1,1\}$
$\omega^{(0)}=(0,\cdots,0)\in R^d$
$b^{(0)}=0$
$k=0$ # 迭代次数
batch_size = n # 批大小
repeat
从数据集中随机抽取n对数据:$(x_{k_1},y_{k_1})\cdots(x_{k_n},y_{k_n})$
选择步长策略$\alpha^{(k)}$
for i = 1,2,···,n
if$(y_iw^Tx_i+b<1)$
$w^{(k+1)}=w^{(k)}-\alpha^{(k)}(\lambda w^{(k)}-y_{k_i}x_{k_i})$
$b^{(k+1)}=b^{(k)}-\alpha^{(k)}(-y_i)=b^{(k)}+\alpha^{(k)}y_i$
else
$w^{(k+1)}=w^{(k)}-\alpha^{(k)}\lambda w^{(k)}$
$b^{(k+1)}=b^{(k)}-\alpha^{(k)}\cdot 0=b^{(k)}$
end if
$k=k+1$
end for
until $(k==k_{max})$
return $w^{(k)},b^{(k)}$

3.3.6 6

设无$\xi_i\geq 0$约束时，问题最优解为$f(w^*,b^*,\xi^*)$；引入$\xi_i\geq 0$约束后，问题最优解变为$f(w^{*\prime},b^{*\prime},\xi^{*\prime})$

首先，显然约束增加后，最优解不会变小，即

$$f(w^{*\prime},b^{*\prime},\xi^{*\prime})\geq f(w^*,b^*,\xi^*)$$

对于$f(w^*,b^*,\xi^*)$，构造$\hat{\xi^*}$使得：

$$\hat{\xi^*}_i=|\xi_i^*|$$

显然有

$$\hat{\xi^*}_i \geq 0$$
$$y_i(w^{*T}x_i + b^*) \geq 1 - \xi_i^* \geq 1 - \hat{\xi^*}_i$$

故$f(w^*, b^*, \hat{\xi^*})$也为加入$\xi_i \geq 0$约束后的解，同时显然有

$$\begin{cases} f(w^*, b^*, \hat{\xi^*}) = f(w^*, b^*, \xi^*) \\ f(w^*, b^*, \hat{\xi^*}) \geq f(w^{*'}, b^{*'}, \xi^{*'}) \end{cases}$$
$$\Rightarrow f(w^{*'}, b^{*'}, \xi^{*'}) \leq f(w^*, b^*, \xi^*)$$

故得到

$$f(w^{*'}, b^{*'}, \xi^{*'}) = f(w^*, b^*, \xi^*)$$

即无论有没有这一约束，问题的最优解不变

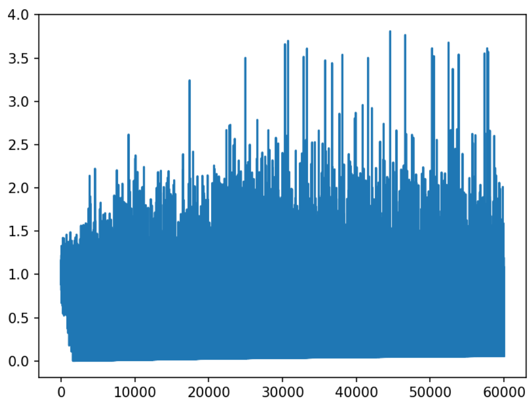
3.4 情绪检测

3.4.1 1

- 见`start_code.py`

3.4.2 2

- 初始参数：$batch\_size = 1, \lambda = 0.0001, \alpha = 0.05$，训练集上损失函数曲线（在`3.4.2`中，后面均简称为"训练曲线"）如下：

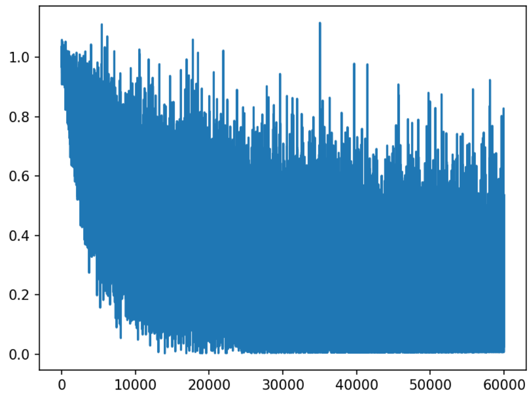



```
train_accuracy:  0.9549079754601227
val_accuracy:  0.853940708604483
```

3.4.2.1 批大小调整

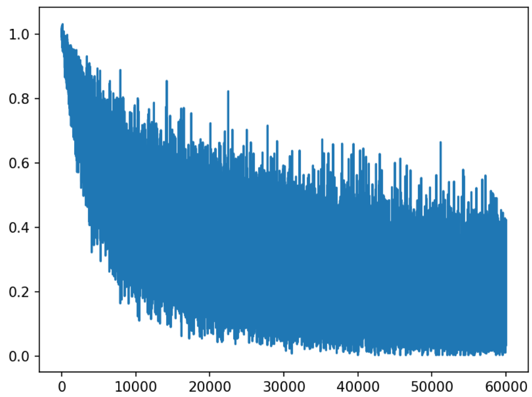
- $batch_size$太小时，训练曲线震荡过大，考虑增大

- $batch\_size = 8, \lambda = 0.0001, \alpha = 0.05$，训练曲线如下：




```
train_accuracy:  0.9564417177914111
val_accuracy:  0.8640636297903109
```

- $batch\_size = 16, \lambda = 0.0001, \alpha = 0.05$，训练曲线如下：

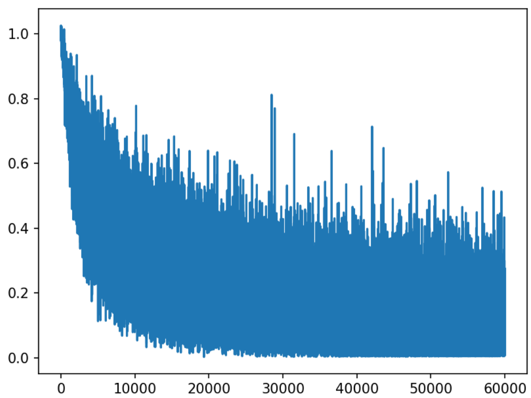



```
train_accuracy:  0.9576687116564417
val_accuracy:  0.866232827187274
```

此时下降趋势较为明显，采用$batch_size = 16$进行后续调参

3.4.2.2 步长调整

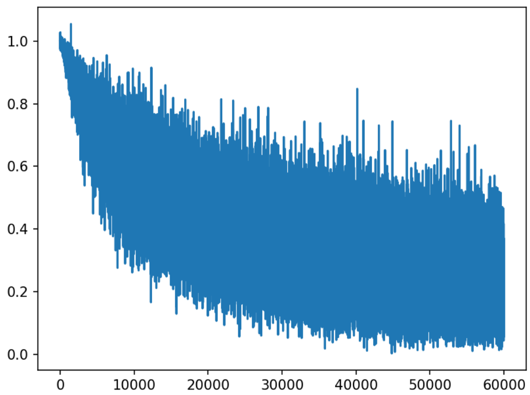
- $batch\_size = 16, \lambda = 0.0001, \alpha = 0.1$




```
train_accuracy:  0.9705521472392638
val_accuracy:  0.8546637744034707
```

此时有过拟合的趋势，考虑调小步长

- $batch\_size = 16, \lambda = 0.0001, \alpha = 0.03$

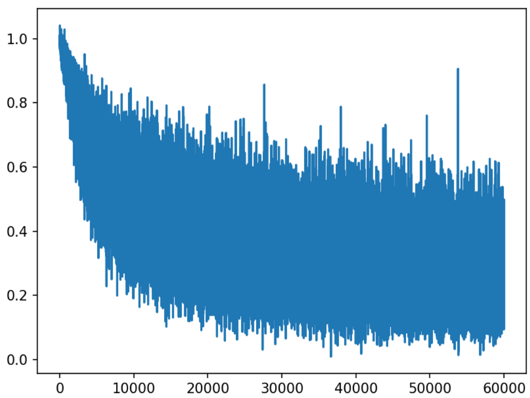



```
train_accuracy:  0.942638036809816
val_accuracy:  0.8626174981923355
```

准确率与

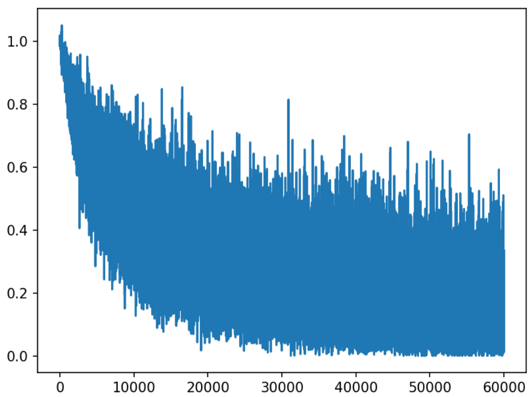
3.4.2.3 正则化参数调整

- $batch\_size = 16, \lambda = 0.0005, \alpha = 0.05$



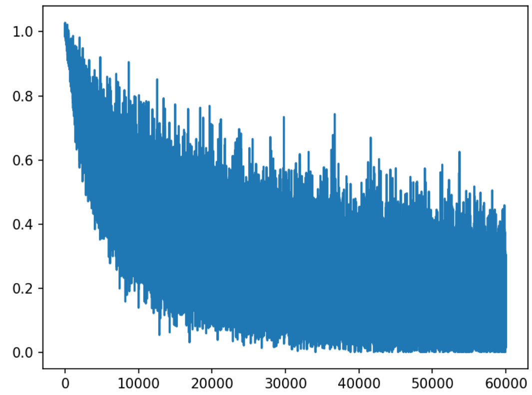

```
train_accuracy:  0.9457055214723926
val_accuracy:  0.8611713665943601
```

- $batch\_size = 16, \lambda = 0.00005, \alpha = 0.05$




```
train_accuracy:  0.9592024539877301
val_accuracy:  0.8647866955892987
```

- $batch\_size = 16, \lambda = 0.00001, \alpha = 0.05$




```
train_accuracy:  0.9598159509202454
val_accuracy:  0.8626174981923355
```

- $\lambda = 0.0001$时表现最佳，故有最终参数确定：$batch\_size = 16, \lambda = 0.00005, \alpha = 0.05$
- 最终表格

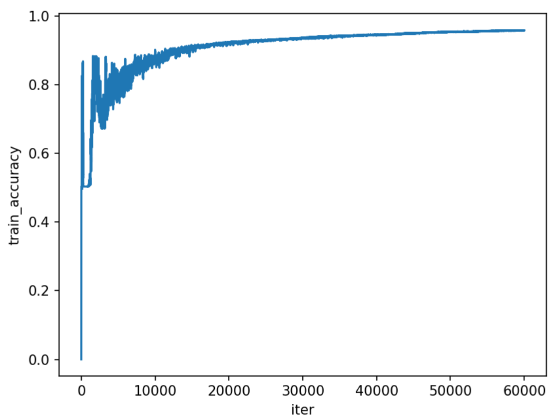
batch_size	λ	α	train_accuracy	val_accuracy
1	0.0001	0.05	0.9549	0.8539
8	0.0001	0.05	0.9564	0.8641
16	0.0001	0.05	0.9577	0.8662
16	0.0001	0.1	0.9706	0.8547
16	0.0001	0.03	0.9426	0.8626
16	0.0005	0.05	0.9457	0.8612
16	0.00005	0.05	0.9592	0.8648
16	0.00001	0.05	0.9598	0.8626

3.4.3 3

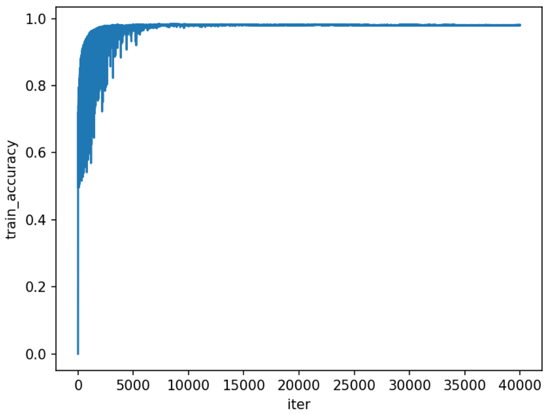
- 代码实现见`start_code.py`

3.4.3.1 训练结果

- 首先是`3.4.1`中算法在训练集上的正确率曲线：



- 以下为$\lambda-$强凸问题的SGD算法在训练集上的正确率曲线：



- 可以看到，新算法在7500步时已经基本收敛，而原算法在50000步后才基本收敛；新算法对模型的收敛加速十分显著

3.4.3.2 理论分析

对于λ–强凸函数的SGD算法，有[1]

$$\sum_{t=1}^{T}(E[f(w^{(t)})]-f(w^*))\leq E[\sum_{t=1}^{T}(\frac{||w^{(t)}-w^*||^2-||w^{(t+1)}-w^*||^2}{2\eta_t}-\frac{\lambda}{2}||w^{(t)}-w^*||^2)]+\frac{\rho^2}{2}\sum_{t=1}^{T}\eta_t$$

此时在常数项引入$\eta_t=1/(\lambda t)$，则可以得到：

$$\sum_{t=1}^{T}(E[f(w^{(t)})]-f(w^*))\leq\frac{\rho^2}{2\lambda}\sum_{t=1}^{T}\frac{1}{t}\leq\frac{\rho^2}{2\lambda}(1+\log(T))$$

从而达到$O(\frac{logT}{T})$的收敛速率

而如果只是固定学习率$\eta_t=t$，则会得到较大的常数项，收敛速度变慢

3.4.4 4

考虑以下等价问题

$$\min_{\alpha}\frac{\lambda}{2}\alpha^TK\alpha+\frac{1}{n}\sum_{i=1}^{n}\max\{0,1-y_i\sum_{j=1}^{n}\alpha_jk(x_i,x_j)\}$$

- 代码见`start_code.py`
- 线性核，循环10000步，batch_size为16，步长为0.1，lambda为0.0001
  - 代码

    ```
    num_iter = 10000
    batch_size = 16
    lambda_reg = 0.0001
    kernel = "linear"
    theta_hist = kernel_svm_subgrad_descent(X_train_vect, y_train, lambda_reg=lambda_reg,
    num_iter=num_iter, batch_size=batch_size, kernel=kernel)
    ```

  - 训练结果如下：

```
train_accuracy:  0.8205521472392638
val_accuracy:  0.779464931308749
```

- 高斯核，循环10000步，batch_size为16，步长为0.1，lambda为0.0001
  - 代码

```python
num_iter = 10000
batch_size = 16
lambda_reg = 0.0001
kernel = "linear"
theta_hist = kernel_svm_subgrad_descent(X_train_vect, y_train, lambda_reg=lambda_reg,
num_iter=num_iter, batch_size=batch_size, kernel=kernel)
# 计算高斯核准确率
X_train_vect_gaussian = np.exp(-X_train_vect**2/2)
X_val_vect_gaussian = np.exp(-X_val_vect**2/2)
y_pred_train = np.sign(X_train_vect_gaussian@theta_hist[-1])
train_accuracy = np.sum(y_pred_train==y_train)/len(y_train)
y_pred_val = np.sign(X_val_vect_gaussian@theta_hist[-1])
val_accuracy = np.sum(y_pred_val==y_val)/len(y_val)
print("train_accuracy: ", train_accuracy)
print("val_accuracy: ", val_accuracy)
```

  - 训练结果为

```
train_accuracy:  0.49662576687116566
val_accuracy:  0.5112075198843095
```

- 可以看到线性核具有不错的拟合能力，但高斯核则几乎没有分类效果；推测是因为本实验中的数据集线性性较好，使用非线性核的拟合效果不佳
- 由此可以得出核函数引入无法提高当前模型准确率，因为当前数据集为较好的线性分布，非线性假设空间的引入无法提高分类的准确率

3.4.5 5

经过以上的分析，最终确定采用λ–强凸SGD算法训练最终模型。

- batch_size $=16$，循环次数为10000，$\lambda=0.0025$
- 结果

```
val_accuracy:  0.8712942877801888
F1-Score:  0.8744710860366713
confusion_matrix:  [[620.  87.]
 [ 91. 585.]]
```

$$Val\_Accuracy = 0.8713$$
$$F1\_Score = 0.8745$$
$$M_{confusion} = \begin{bmatrix} 620 & 87 \\ 91 & 585 \end{bmatrix}$$

1. Understanding Machine Learning, P196 ↩