HW2 Siyong Liu

Section 1

1

$$\frac{1}{N}||X\hat{b}-y||^2 = \frac{1}{N}||X(X^TX)^{-1}X^T(Xb+\epsilon)-(Xb+\epsilon)||_2^2$$
$$= \frac{1}{N}||(X(X^TX)^{-1}X^T-I)\epsilon||_2^2$$

2

$$Symmetric: A = X(X^TX)^{-1}X^T = X((X^TX)^{-1})^TX^T = A^T$$

$$A^2 = A^TA = X((X^TX)^{-1})^TX^TX(X^TX)^{-1}X^T = X(X^TX)^{-1}X^T = A$$

rank(A) = rank(X) = diagonal eigenmatrix $A-I = \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 0 \end{bmatrix}$ with d of 0 and N-d of 1, and

$A-I = QDQ^{-1}$, $QDQ^{-1}\epsilon$= $\begin{bmatrix} 0 & & & \\ & \ddots & & \\ & & -\epsilon_i & \\ & & & -\epsilon_j \end{bmatrix}$ with (N-d)$\epsilon_{i,j,\dots}$

$\mathbb{E} = [\frac{1}{N}||(X(X^TX)X^T-I)\epsilon||_2^2] = \mathbb{E}[\frac{1}{N}||(A-I)\epsilon||_2^2] = \mathbb{E}[\frac{1}{N}\sum_{i=1}^{N-d}\epsilon_i^2] = \frac{N-d}{N}\sigma^2$

3

when d dropping to N, (N-d) in function above will close to 0 and will cause the error close to 0

Section 2

4

```python
def feature_normalization(train, test):
    # train_const = train[:, 0].T
    train = train[:, 1:]
    test = test[:, 1:]
    train_max = train.max(axis=0)
    train_min = train.min(axis=0)
    train_normalized = (train - train_min) / (train_max - train_min)
    test_normalized = (test - train_min) / (train_max - train_min)
    # train_normalized = np.c_[train_const, train_normalized]
    return train_normalized, test_normalized
```

```python
# test normalize
X = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
y = np.array([1,6,11])
feature_normalization(X, y)
>>>
(array([[1. , 0. , 0. ],
        [4. , 0.5, 0.5],
        [7. , 1. , 1. ]]),
 array([0. , 0.5, 1. ]))
```

5

$$J(\theta) = \frac{1}{m}(X\theta - y)_2^2$$

6

$$\nabla J(\theta) = \frac{2}{m}X^T(X\theta - y)$$

7

$$\theta = \theta - \frac{2}{m}X^T(X\theta - y) * \eta$$

8

```python
def compute_square_loss(X, y, theta):
    loss = X @ theta - y
    return 1 / X.shape[0] * (loss.T @ loss)
```

```python
# test  mse
X = np.array([[1, 4], [2, 5], [3, 6]])
y = np.array([7, 8, 10])
theta = np.array([1, 1])
compute_square_loss(X, y, theta)
>>>
2.0
```

9

```
def compute_square_loss_gradient(X, y, theta):
    loss = X @ theta - y
    return 2 / X.shape[0] * (X.T @ loss)
```

```
# test gradient mse
X = np.array([[1, 4], [2, 5], [3, 6]])
y = np.array([7, 8, 10])
theta = np.array([1, 1])
compute_square_loss_gradient(X, y, theta)
>>>
array([ -4.66666667, -12.66666667])
```

10

```
def grad_checker(X, y, theta, epsilon=0.01, tolerance=1e-4):
    true_gradient = compute_square_loss_gradient(X, y, theta)  # The true
gradient
    num_features = theta.shape[0]
    # Initialize the gradient we approximate
    approx_grad = np.zeros(num_features)
    for i in range(num_features):
        e_i = np.zeros(num_features)
        e_i[i] = 1
        mse_plus = compute_square_loss(X, y, theta + epsilon * e_i)
        mse_minus = compute_square_loss(X, y, theta - epsilon * e_i)
        approx_grad[i] = (mse_plus - mse_minus)/(2 * epsilon)
        distance = np.linalg.norm(approx_grad - true_gradient, ord = 2)
    return distance < tolerance

def generic_gradient_checker(X, y, theta, objective_func, gradient_func,
                             epsilon=0.01, tolerance=1e-4):
    true_gradient = gradient_func(X, y, theta)  # The true gradient
    num_features = theta.shape[0]
    # Initialize the gradient we approximate
    approx_grad = np.zeros(num_features)

    for i in range(num_features):
        e_i = np.zeros(num_features)
        e_i[i] = 1
        mse_plus = objective_func(X, y, theta + epsilon * e_i)
        mse_minus = objective_func(X, y, theta - epsilon * e_i)
        approx_grad[i] = (mse_plus - mse_minus)/(2 * epsilon)
        distance = np.linalg.norm(approx_grad - true_gradient, ord = 2)
    return distance < tolerance
```

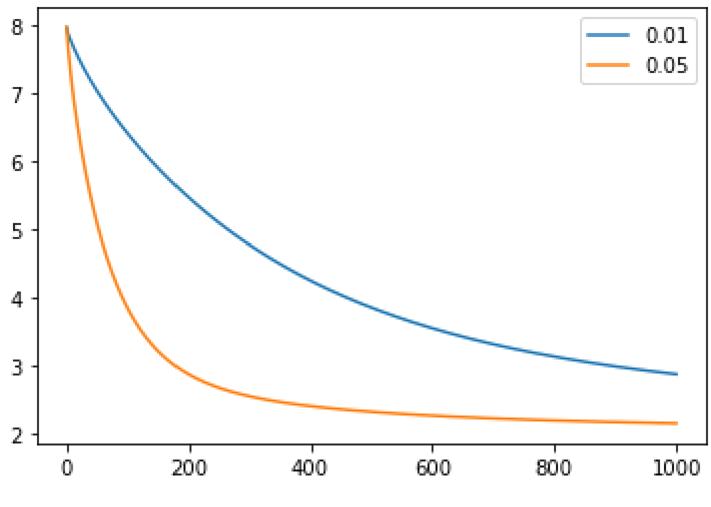
```
# test generic grad checker
theta = np.ones(X_train.shape[1])
grad_checker(X_train,y_train,theta)
>>>
True
```

11

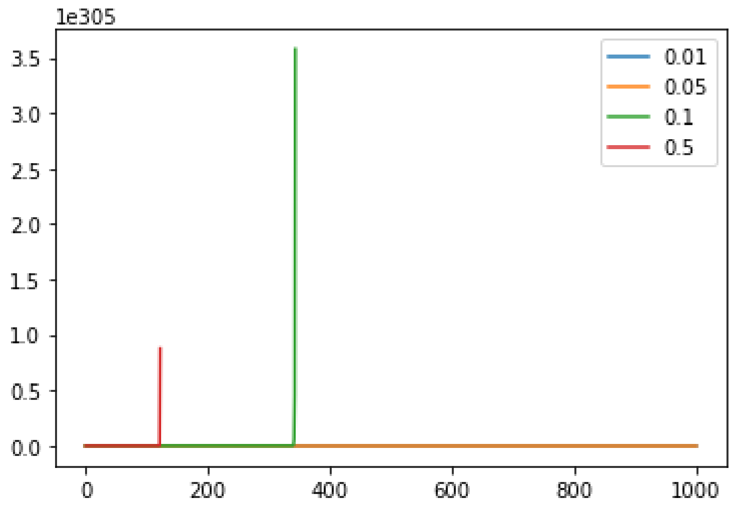
```
def batch_grad_descent(X, y, alpha, num_step=1000, grad_check=False):
    num_instances, num_features = X.shape[0], X.shape[1]
    theta_hist = np.zeros((num_step + 1, num_features))  #Initialize theta_hist
    loss_hist = np.zeros(num_step + 1)  #Initialize loss_hist
    theta = np.zeros(num_features)  #Initialize theta

    for i in range(num_step + 1):
        theta_hist[i, ] = theta
        loss_hist[i] = compute_square_loss(X, y, theta)
        if grad_check:
            assert grad_checker(X, y, theta)
        theta = theta - compute_square_loss_gradient(X, y, theta) * alpha
    return theta_hist, loss_hist
```

12

```
def step_size_plot(X, y, step_size_list):
    step_size = step_size_list
    for alpha in step_size:
        _, loss_hist = batch_grad_descent(X, y, alpha, num_step=2000)
        plt.plot(loss_hist, label=alpha)
    plt.legend()
    plt.show()
step_size_plot(X_train, y_train, step_size_list = [.01, .05])
```




```
step_size_plot(X_train, y_train, step_size_list = [.01, .05, 0.1, 0.5])
```



0.01 converge slow

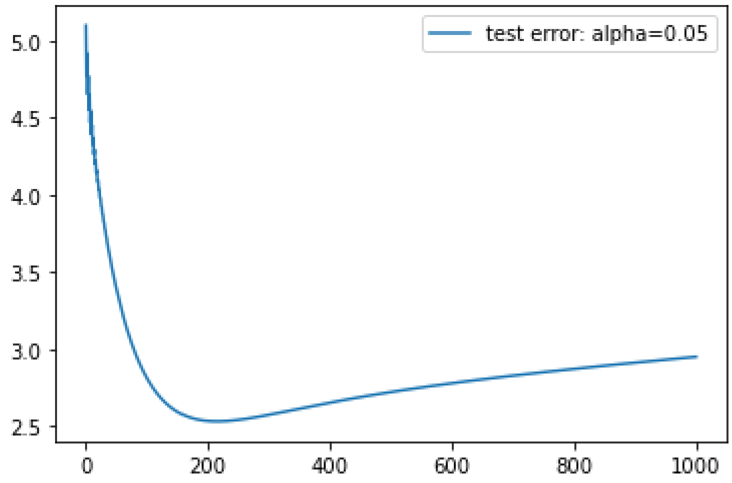
0.05 converge faster

0.1, 0.5 diverge, bigger step size may cause diverge

> if using $J(\theta) = \frac{0.5}{m}(X\theta - y)_2^2$ as objective function rather than $J(\theta) = \frac{1}{m}(X\theta - y)_2^2$, stepsize = 0.1 will not diverge

13

```
theta_loss, loss_hist = batch_grad_descent(X_train, y_train, alpha=.05)
test_error = []
for theta in theta_loss:
    test_error.append(compute_square_loss(X_test, y_test, theta))
plt.plot(range(len(theta_loss)), test_error, label="test error: alpha=0.05")
plt.legend()
plt.show()
```



14

$$\nabla J_\lambda(\theta) = \frac{2}{m} X^T (X\theta - y) + 2\lambda\theta$$
$$\theta = \theta - \alpha \nabla J_\lambda(\theta_n)$$

15

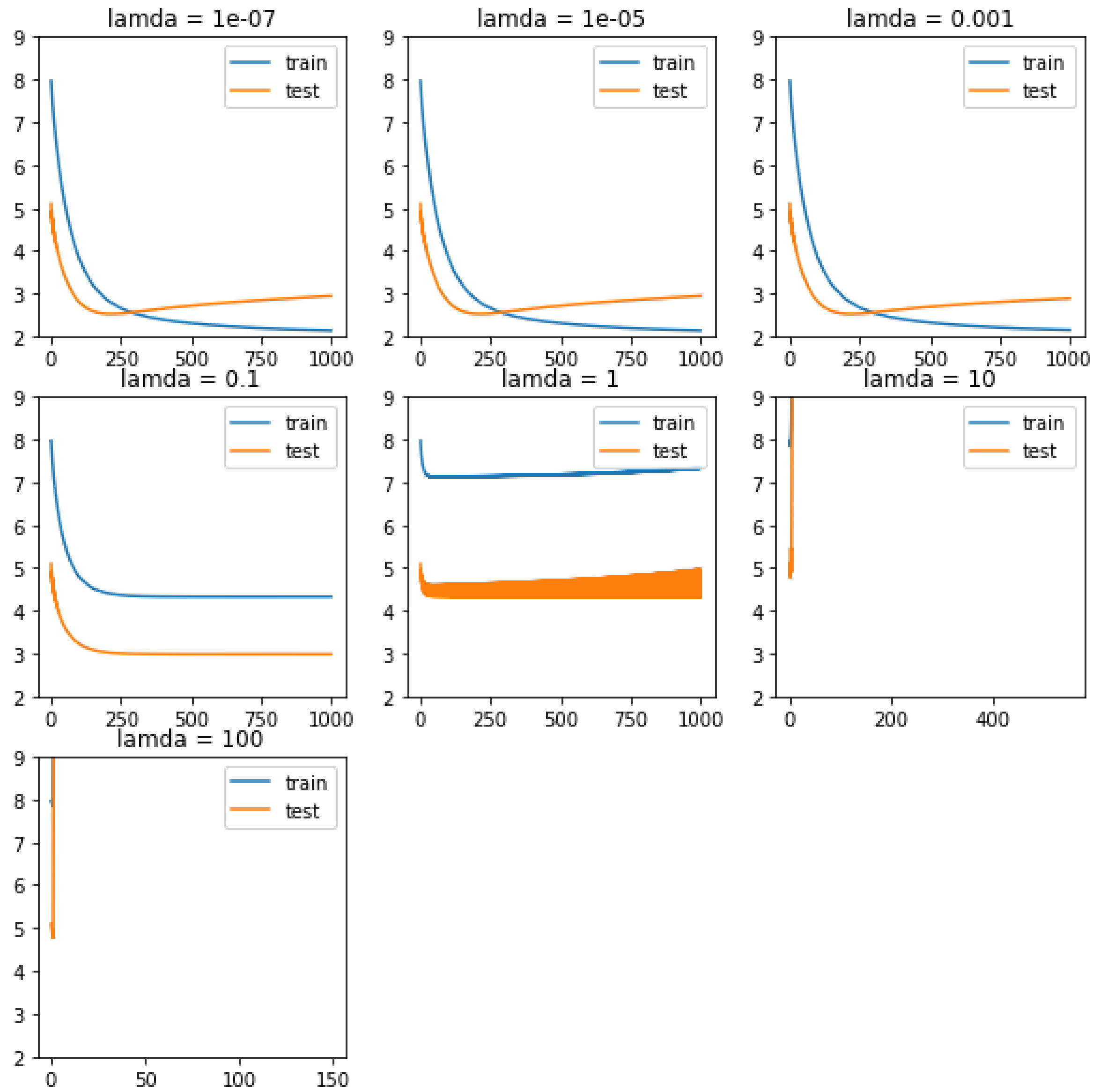
```
def compute_regularized_square_loss_gradient(X, y, theta, lambda_reg):
    mseg = compute_square_loss_gradient(X, y, theta)
    return mseg + 2 * lambda_reg * theta
```

16

```
def regularized_grad_descent(X, y, alpha=0.05, lambda_reg=10**-2,
num_step=1000):
    num_instances, num_features = X.shape[0], X.shape[1]
    theta = np.zeros(num_features)  # Initialize theta
    theta_hist = np.zeros((num_step+1, num_features))  # Initialize theta_hist
    loss_hist = np.zeros(num_step+1)  # Initialize loss_hist

    for i in range(num_step + 1):
        theta_hist[i, ] = theta
        loss_hist[i] = compute_square_loss(X, y, theta)
        theta = theta - \
            compute_regularized_square_loss_gradient(
                X, y, theta, lambda_reg) * alpha
    return theta_hist, loss_hist
```

17

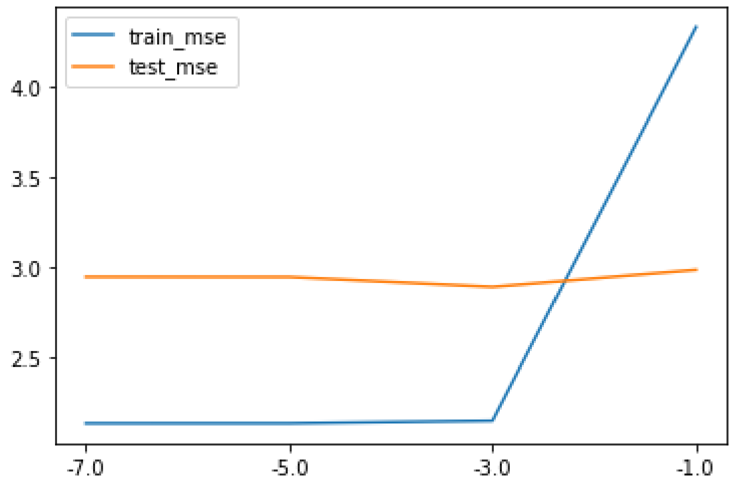
```
# detect overfitting
fig = plt.figure(figsize=(10, 10))
lambda_reg_list = [1e-7, 1e-5, 1e-3, 1e-1, 1, 10, 100]
for i, lambda_reg in enumerate(lambda_reg_list):
    theta_hist, loss_hist = regularized_grad_descent(
        X_train, y_train, lambda_reg=lambda_reg)
    test_loss = []
    for theta in theta_hist:
        test_loss.append(compute_square_loss(X_test, y_test, theta))
    plt.subplot(3, 3, i+1)
    plt.plot(range(len(loss_hist)), test_loss, label=lambda_reg)
    plt.ylim([1,5])
    plt.title(f"lamda = {lambda_reg}")
plt.show()
```



Test error decrease then increase when $\lambda = 10^{-7}, 10^{-5}, 10^{-3}$, when training error is always decrease

18

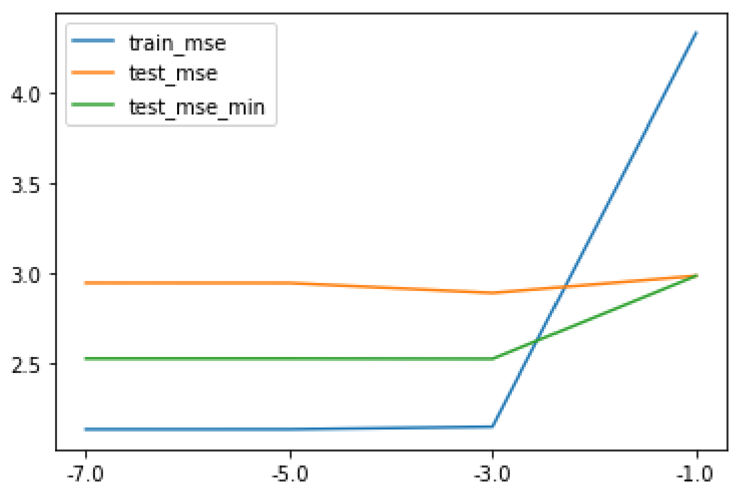
```
# select lambda
lambda_reg_list = [1e-7, 1e-5, 1e-3, 1e-1]
train_mse = []
test_mse = []
for lambda_reg in lambda_reg_list:
    theta_hist, loss_hist = regularized_grad_descent(
        X_train, y_train, alpha=0.05, lambda_reg=lambda_reg)
    train_mse.append(loss_hist[-1])
    test_mse.append(compute_square_loss(X_test, y_test, theta_hist[-1]))
plt.plot(train_mse, label='train_mse')
plt.plot(test_mse, label='test_mse')
plt.xticks(range(len(lambda_reg_list)), np.log10(lambda_reg_list))
plt.legend()
plt.show()
```



select $\lambda = 0.001$, test error and train error are both small.

19

```
# early stop
lambda_reg_list = [1e-7, 1e-5, 1e-3, 1e-1]
train_mse = []
test_mse = []
test_mse_min = []
for lambda_reg in lambda_reg_list:
    theta_hist, loss_hist = regularized_grad_descent(
        X_train, y_train, alpha=0.05, lambda_reg=lambda_reg)
    test_error = []
    for theta in theta_hist:
        test_error.append(compute_square_loss(X_test, y_test, theta))
    train_mse.append(loss_hist[-1])
    test_mse.append(compute_square_loss(X_test, y_test, theta_hist[-1]))
    test_mse_min.append(np.min(test_error))
plt.plot(train_mse, label='train_mse')
plt.plot(test_mse, label='test_mse')
plt.plot(test_mse_min, label='test_mse_min')
plt.xticks(range(len(lambda_reg_list)), np.log10(lambda_reg_list))
plt.legend()
plt.show()
```



20

Select $\lambda = 0.001$, it arrive the minimum error(around 2.5) much faster
better than 0.1 because test error are much smaller
better than 1e-5 and 1e-7 because it generalize well than smaller λ

21

$$
\begin{aligned}
J_\lambda(\theta) &= \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2 + \lambda\frac{1}{m}\sum_{i=1}^{m}\theta^T\theta \\
&= \frac{1}{m}\sum_{i=1}^{m}((h_\theta(x_i) - y_i)^2 + \lambda\theta_i^T\theta_i)
\end{aligned}
$$

$$
f_i(\theta) = (h_\theta(x_i) - y_i)^2 + \lambda\theta^T\theta
$$

22

$$
\begin{aligned}
\mathbb{E}[\nabla f_i(\theta)] &= \frac{1}{m}\sum_{i=1}^{m}(2x_i^T(\theta^T x_i - y_i) + 2\lambda\theta) \\
&= \frac{2}{m}\sum_{i=1}^{m}x_i^T(\theta^T x_i - y_i) + 2\lambda\theta \\
&= \frac{2}{m}((X\theta - y)^T) + 2\lambda\theta \\
&= \nabla J_\lambda(\theta)
\end{aligned}
$$

23

$$
\theta = \theta - \eta * (2x_i^T(\theta^T x_i - y_i) + 2\lambda\theta)
$$

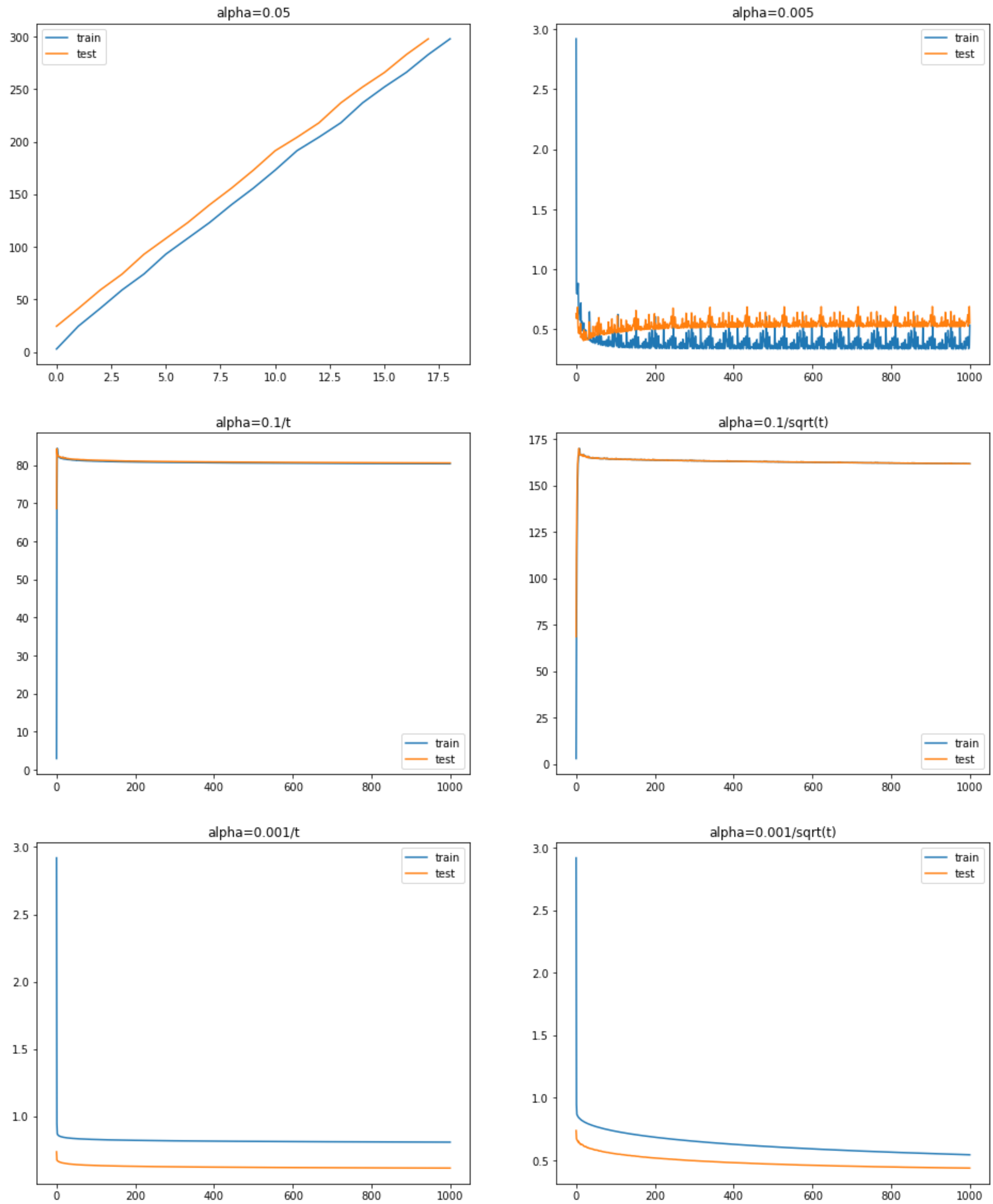
24

```
from sklearn.utils import shuffle
def stochastic_grad_descent(X, y, alpha=0.01, lambda_reg=10**-2, num_epoch=1000,
eta0=False):
    num_instances, num_features = X.shape[0], X.shape[1]
    theta = np.ones(num_features)  # Initialize theta

    # Initialize theta_hist
    theta_hist = np.zeros((num_epoch, num_instances, num_features))
    loss_hist = np.zeros(num_epoch)  # Initialize loss_hist

    for i in range(num_epoch):
        theta_hist[i, ] = theta
        loss_hist[i] = compute_square_loss(
            X, y, theta) + lambda_reg * (theta.T @ theta)
        if alpha == '0.1/sqrt(t)':
            step_size = 0.1 / np.sqrt(i + 1)
        elif alpha == '0.1/t':
            step_size = 0.1 / (i + 1)
        elif alpha == '0.001/sqrt(t)':
            step_size = 0.001 / np.sqrt(i + 1)
        elif alpha == '0.001/t':
            step_size = 0.001 / (i + 1)
        else:
```

```
            step_size = alpha
        X, y = shuffle(X, y, random_state=0)
        for j in range(num_instances):
            Xi = np.reshape(X[j], (-1, X[j].shape[0]))
            yi = np.reshape(y[j], 1)
            theta = theta - step_size * \
                compute_regularized_square_loss_gradient(Xi, yi, theta,
lambda_reg)
            theta_hist[i, j, ] = theta
    return theta_h  ist, loss_hist
```

25

```
# choose step size
lambda_reg = 0.001
alphas = [0.05, 0.005, '0.1/t', '0.1/sqrt(t)', '0.001/t', '0.001/sqrt(t)']
fig = plt.figure(figsize=(16, 20))
for i, alpha in enumerate(alphas):
  plt.subplot(3, 2, i+1)
  train_theta_hist, train_loss_hist = stochastic_grad_descent(X_train, y_train,
alpha, lambda_reg)
  plt.plot(np.log10(train_loss_hist).T, label='train')
  test_loss_hist = np.zeros(1000)
  for j, thetas in enumerate(train_theta_hist):
    test_loss_hist[j] = compute_square_loss(X_test, y_test, thetas[-1]) \
      + lambda_reg * (thetas[-1].T @ thetas[-1])
  plt.plot(np.log10(test_loss_hist).T, label='test')
  plt.title(f"alpha={alpha}")
  plt.legend()
plt.show()
```



- λ=0.05 diverge
- $\lambda = 0.005, \frac{1}{t}, \frac{1}{\sqrt{t}}$ converge
- SGD are more efficient in the first several gradient but need to select a correct α to get a converge result

Section 3

26

$$
\begin{aligned}
L(\theta) &= \tfrac{1}{2m}\sum_{i=1}^{m}(1+y_i)log(1+e^{-h_{\theta,b}(x_i)}) + (1-y_i)log(1+e^{h_{\theta,b}(x_i)}) \\
&= \tfrac{1}{2m}\sum_{i=1}^{m}(1-y_i)log(1+e^{-h_{\theta,b}(x_i)}) + (1+y_i)log(1+e^{-h_{\theta,b}(x_i)}) \\
&= \tfrac{1}{2m}\sum_{i=1}^{m} 2log(1+e^{-h_{\theta,b}(x_i)}) + (1-y_i)h_{\theta,b}(x_i)
\end{aligned}
$$

$$
L(\theta) = \begin{cases} 2log(1+e^{-h_{\theta,b}(x_i)}) & y_i = 1 \\ 2log(1+e^{h_{\theta,b}(x_i)}) & y_i = -1 \end{cases}
$$

$$
L(\theta) = \frac{1}{m}log(1+e^{-y_i h_{\theta,b}(x_i)})
$$

27

$$
L(\theta) = \frac{1}{2m}\sum_{i=1}^{m}((1+y_i)log(1-e^{h_\theta}) + (1-y_i)log(1-e^{h_\theta})) + \alpha|\theta|
$$

28

```
def classification_error(clf, X, y):
    y_pred = clf.predict(X)
    score = np.sum(y_pred != y) / X.shape[0]
    return score
```

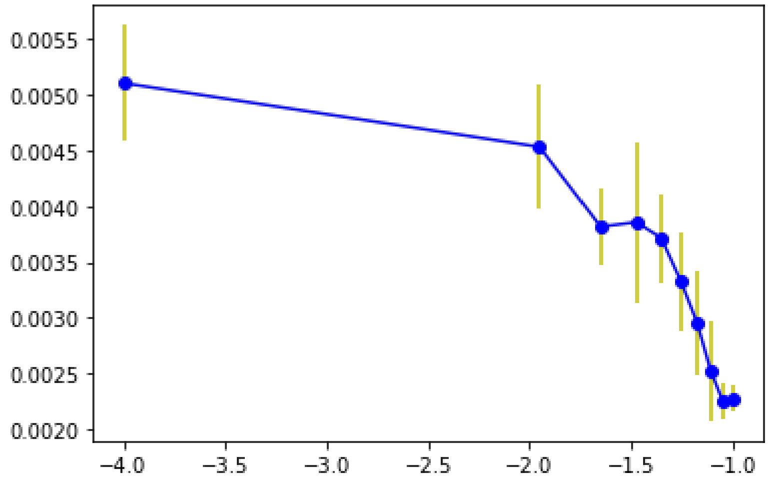
```
# test
clf = SGDClassifier(loss='log', max_iter=1000,
                    tol=1e-3,
                    penalty='l1', alpha=0.01,
                    learning_rate='invscaling',
                    power_t=0.5,
                    eta0=0.01,
                    verbose=1)
clf.fit(X_train, y_train)
assert round(classification_error(clf, X_test, y_test),13) \
    == round(1 - clf.score(X_test, y_test), 13)
>>>
True
```

29

```
y_err_hist = np.zeros((10, 10))
theta_hist = np.zeros((10, 10, X_train.shape[1]))
alphas = np.linspace(1e-4, 1e-1, 10)

for i in range(10):
    for j in range(10):
        clf = SGDClassifier(loss='log', max_iter=1000,
                            tol=1e-3,
                            penalty='l1', alpha=alphas[i],
                            learning_rate='invscaling',
                            power_t=0.5,
```

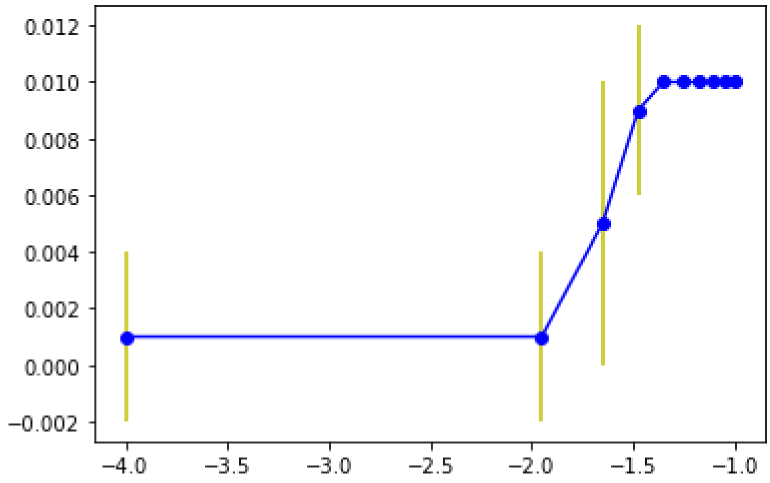
```
                                eta0=0.01,
                                verbose=1)
        clf.fit(X_train, y_train)
        theta_hist[i, j, ] = clf.coef_
        y_err_hist[i, j] = classification_error(clf, X_test, y_test)
```

```
plt.errorbar(np.log10(alphas),
             np.mean(y_err_hist, axis=1),
             yerr=np.std(y_err_hist, axis=1),
             fmt='o-', ecolor='y', color='b')
```



If also using sample of test set as in the code function pydoc:

```
X_test, y_test = sub_sample(100, X_test, y_test)
```



30

SGD randomly select the data to calculate gradient in each epoch. Average helps minimize the error select from outlier or noisy in each epoch.

31

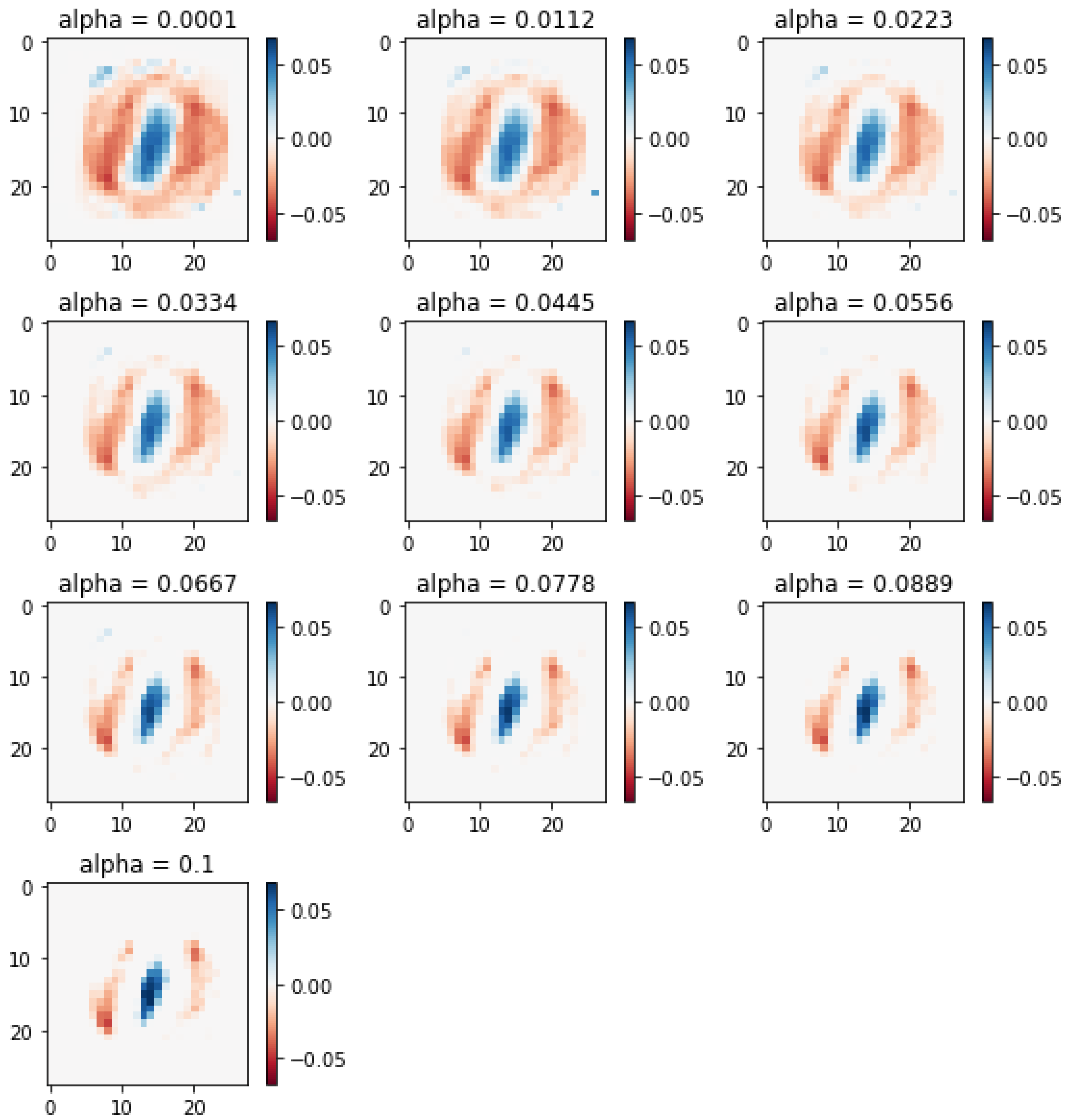
$\alpha = 0.0889$ arrive the minimum error

If also using sample(first 100) of test set:

$\alpha = 0.01$ arrive the minimum error

32

```
thetas = np.mean(theta_hist, axis = 1)

fig = plt.figure(figsize=(10, 10))
for i, theta in enumerate(thetas):
    plt.subplot(4, 3, i+1)
    plt.imshow(theta.reshape((28,28)), cmap=plt.cm.RdBu, vmax=0.3, vmin=-0.3)
    plt.colorbar()
plt.show()
```



33

- Higher Gradient cause a large penalty, and model fit less noisy.
  E.g., Compare alpha=0.1, 0.00334. It has less pixel which are very light
- θ shows which feature effect the classification more and less.
- L1 regularization select the feature has large effect, witch means will have less pixel are significant, but the reset pixel with dark red and dark blue
- l2 it will have more 'pixel'(feature) have red or blue color, although they are shallow