```
# Task 1
# comp136 project2 Task 1
# beibei du
# 10/27/2016
import os
import sys
import math
import numpy as np
from numpy.linalg import inv
import matplotlib.pyplot as plt

def read_file(filename):
	result = []
	with open(filename, 'r') as f:
		for line in f:
			line = line.strip('\n')
			line = line.split(',')
			result.append(line)
	result = np.array(result)
	result = result.astype(float)
	return result

def read_file_r(filename):
	result = []
	with open(filename, 'r') as f:
		for line in f:
			line = line.strip('\n')
			result.append(float(line))
	return result

# calculate w
def calculate_w(lam, tr, tr_r):
	feature = len(tr[0])
	example = len(tr)
	iden = np.identity(feature)
	iden = np.dot(lam, iden)
	tr_trans = np.transpose(tr)
	sum_1 = np.add(np.dot(tr_trans,tr),iden)
	sum_inverse = inv(sum_1)
	prod = np.dot(sum_inverse, tr_trans)
	w = np.dot(prod,tr_r)
	return w

# calculate mse
```

```
def calculate_mse(w, t, t_r):
	N = len(t)
	pro = np.dot(t, w)
	dif = np.subtract(pro, t_r)
	squ = np.square(dif)
	mse = np.sum(squ) / float(N)
	return mse

def Regularization(tr, tr_r, te, te_r):
	MSE_tr = []
	MSE_te = []
	for lam in range(151):
			w = calculate_w(lam, tr, tr_r)
			mse_train = calculate_mse(w, tr, tr_r)
			mse_te = calculate_mse(w, te, te_r)
			MSE_tr.append(mse_train)
			MSE_te.append(mse_te)
	return MSE_tr, MSE_te

def main():

	tr_file = ['train-crime.csv', 'train-wine.csv','train-100-10.csv', 'train-100-100.csv', 'train-
1000-100.csv']
	tr_r_file = ['trainR-crime.csv', 'trainR-wine.csv', 'trainR-100-10.csv', 'trainR-100-100.csv',
'trainR-1000-100.csv']
	te_file = ['test-crime.csv', 'test-wine.csv','test-100-10.csv', 'test-100-100.csv', 'test-1000-
100.csv']
	te_r_file = ['testR-crime.csv', 'testR-wine.csv','testR-100-10.csv', 'testR-100-100.csv',
'testR-1000-100.csv']



	tr_mse = []
	te_mse = []

	file_number = len(tr_file)
	for i in range(file_number):
		tr = read_file(tr_file[i])
		tr_r = read_file_r(tr_r_file[i])
		te = read_file(te_file[i])
		te_r = read_file_r(te_r_file[i])
		if tr_file[i] == 'train-1000-100.csv':
			MSE_tr, MSE_te = Regularization(tr, tr_r, te, te_r)
			tr_mse.append(MSE_tr)
			te_mse.append(MSE_te)
```

```
                MSE_tr, MSE_te = Regularization(tr[0:50], tr_r[0:50], te, te_r)
                tr_mse.append(MSE_tr)
                te_mse.append(MSE_te)
                MSE_tr, MSE_te = Regularization(tr[0:100], tr_r[0:100], te, te_r)
                tr_mse.append(MSE_tr)
                te_mse.append(MSE_te)
                MSE_tr, MSE_te = Regularization(tr[0:150], tr_r[0:150], te, te_r)
                tr_mse.append(MSE_tr)
                te_mse.append(MSE_te)

            # Task 1 Regularization
            else:
                MSE_tr, MSE_te = Regularization(tr, tr_r, te, te_r)
                tr_mse.append(MSE_tr)
                te_mse.append(MSE_te)

        #print tr_mse
        #print te_mse

        "150  opt lamda:23, opt tse: 4.84894305335"
        "100  opt lamda:19, opt tse: 5.20591195733"
        "50   opt lamda: 8, opt tse: 5.54090222919"
        "1000-100 opt lamda: 27, opt tse: 4.31557063032"
        "100-100 opt lamda: 22, opt tse: 5.07829980059"
        "100-10 opt lamda: 8 , opt tse: 4.15967850948"
        "wine opt lamda: 2, opt tse: 0.625308842305"
        "crime opt lamda:75, opt tse: 0.389023387713"
        #[ 0.389023387713,0.625308842305,4.15967850948,5.07829980059,4.31557063032,5.
54090222919,5.20591195733,4.84894305335]

        plt.figure(1)
        lam = range(151)
        plt.plot(lam, tr_mse[0], 'r', label = 'train crime mse')
        plt.plot(lam, te_mse[0], 'g', label = 'test crime mse')
        plt.title('crime MSE')
        plt.ylabel("mse")
        plt.legend(loc = 0)
        plt.savefig("crime.png")
        plt.clf()

        plt.figure(2)
        lam = range(151)
        plt.plot(lam, tr_mse[1], 'r', label = 'train wine mse')
```

```
plt.plot(lam, te_mse[1], 'g', label = 'test wine mse')
plt.title('wine MSE')
plt.ylabel("mse")
plt.xlabel('lambda')
plt.legend(loc = 4)
plt.savefig("wine.png")
plt.clf()

plt.figure(3)
lam = range(151)
plt.plot(lam, tr_mse[2], 'r', label = 'train-100-10 mse')
plt.plot(lam, te_mse[2], 'g', label = 'test-100-10 mse')
plt.axhline(y = 3.78, xmin = 0, xmax = 150, hold = None, label = 'true mse')
plt.title('100-10 MSE')
plt.ylabel("mse")
plt.xlabel('lambda')
plt.legend(loc = 0)
plt.savefig("100-10.png")
plt.clf()

plt.figure(4)
lam = range(151)
plt.plot(lam, tr_mse[3], 'r', label = 'train-100-100 mse')
plt.plot(lam, te_mse[3], 'g', label = 'test-100-100 mse')
plt.axhline(y = 3.78, xmin = 0, xmax = 150, hold = None, label = 'true mse')
plt.title('100-100 MSE')
plt.ylabel('mse')
plt.xlabel('lambda')
plt.legend(loc = 5)
plt.ylim([0.0, 50.0])
plt.savefig("100-100.png")
plt.clf()


plt.figure(5)
lam = range(151)
plt.plot(lam, tr_mse[4], 'r', label = 'train-1000-100 mse')
plt.plot(lam, te_mse[4], 'g', label = 'test-1000-100 mse')
plt.axhline(y = 4.015, xmin = 0, xmax = 150, hold = None, label = 'true mse')
plt.title('1000-100 MSE')
plt.ylabel("mse")
plt.xlabel('lambda')
plt.legend(loc = 0)
plt.savefig("1000-100.png")
```

```
        plt.clf()

        plt.figure(6)
        lam = range(151)
        plt.plot(lam, tr_mse[5], 'r', label = 'train-50(1000)-100 mse')
        plt.plot(lam, te_mse[5], 'g', label = 'test-50(1000)-100 mse')
        plt.axhline(y = 4.015, xmin = 0, xmax = 150, hold = None, label = 'true mse')
        plt.title('50(1000)-100 MSE')
        plt.ylabel("mse")
        plt.xlabel('lambda')
        plt.ylim([0.0, 20.0])
        plt.legend(loc = 2)
        plt.savefig("50(1000)-100.png")
        plt.clf()

        plt.figure(7)
        lam = range(151)
        plt.plot(lam, tr_mse[6], 'r', label = 'train-100(1000)-100 mse')
        plt.plot(lam, te_mse[6], 'g', label = 'test-100(1000)-100 mse')
        plt.axhline(y = 4.015, xmin = 0, xmax = 150, hold = None, label = 'true mse')
        plt.title('100(1000)-100 MSE')
        plt.ylabel("mse")
        plt.legend(loc = 0)
        plt.ylim([0.0, 20.0])
        plt.savefig("100(1000)-100.png")
        plt.clf()

        plt.figure(8)
        lam = range(151)
        plt.plot(lam, tr_mse[7], 'r', label = 'train-150(1000)-100 mse')
        plt.plot(lam, te_mse[7], 'g', label = 'test-150(1000)-100 mse')
        plt.axhline(y = 4.015, xmin = 0, xmax = 150, hold = None, label = 'true mse')
        plt.title('150(1000)-100 MSE')
        plt.ylabel("mse")
        plt.xlabel('lambda')
        plt.legend(loc = 0)
        plt.savefig("150(1000)-100.png")
        plt.clf()

main()
```

```
# Task 2
# comp136 project2 Task 2
# beibei du
# 10/27/2016

import os
import sys
import math
import numpy as np
from numpy.linalg import inv
import random
import matplotlib.pyplot as plt

def read_file(filename):
	result = []
	with open(filename, 'r') as f:
		for line in f:
			line = line.strip('\n')
			line = line.split(',')
			result.append(line)
	result = np.array(result)
	result = result.astype(float)
	return result

def read_file_r(filename):
	result = []
	with open(filename, 'r') as f:
		for line in f:
			line = line.strip('\n')
			result.append(float(line))
	return result

# calculate w
def calculate_w(lam, tr, tr_r):
	feature = len(tr[0])
	example = len(tr)
	iden = np.identity(feature)
	iden = np.dot(lam, iden)
	tr_trans = np.transpose(tr)
	sum_1 = np.add(np.dot(tr_trans,tr),iden)
	sum_inverse = inv(sum_1)
	prod = np.dot(sum_inverse, tr_trans)
	w = np.dot(prod,tr_r)
	return w
```

```
# calculate mse
def calculate_mse(w, t, t_r):
	N = len(t)
	pro = np.dot(t, w)
	dif = np.subtract(pro, t_r)
	squ = np.square(dif)
	mse = np.sum(squ) / float(N)
	return mse

def main():
	tr = read_file('train-1000-100.csv')
	tr_r = read_file_r('trainR-1000-100.csv')
	te = read_file('test-1000-100.csv')
	te_r = read_file_r('testR-1000-100.csv')
	lam_val = [15, 27, 50]
	train_size = [10, 20,30, 40,50, 100, 150, 200, 250, 300,350, 400,450, 500,550, 600,650,
700,750,800]
	lam_mse_list = []
	tr = tr.tolist()
	for lam in lam_val:
		lam_mse = []
		for i in train_size:
			te_mse_list = []
			for j in range(10):
				train = random.sample(tr, i)
				train_r = []
				for item in train:
					index = tr.index(item)
					train_r.append(tr_r[index])
				train = np.array(train)
				w = calculate_w(lam, train, train_r)
				te_mse = calculate_mse(w, te, te_r)
				te_mse_list.append(te_mse)
			te_mse_avg = sum(te_mse_list) / float(10)
			lam_mse.append(te_mse_avg)
		print "lamda = %f"%lam
		print lam_mse
		lam_mse_list.append(lam_mse)

	plt.figure(1)
	x = range(20)
	plt.plot(x, lam_mse_list[0], 'r', marker = '*', label = 'lambda = 15')
	plt.plot(x, lam_mse_list[1], 'g', marker = '*', label = 'lambda = 27')
```

```
        plt.plot(x, lam_mse_list[2], 'b', marker = '*',label = 'lambda = 50')
        plt.axhline(y = 4.015, xmin = 0, xmax = 150, hold = None, label = 'True mse')
        plt.xticks(x, map(lambda x: r'$\frac{%s}{N}$'%str(x), train_size))
        plt.title('Learning Curves')
        plt.ylabel("mse")
        plt.xlabel('train size')
        plt.ylim(3.5,12.0)
        plt.legend(loc = 1)
        plt.show()
main()


# Task 3

# comp136 project2 Task 3
# beibei du
#  10/27/2016
import datetime
import os
import sys
import math
import numpy as np
from numpy.linalg import inv
import matplotlib.pyplot as plt

# to store optimal lambda
lam_list = []

def read_file(filename):
        result = []
        with open(filename, 'r') as f:
                for line in f:
                        line = line.strip('\n')
                        line = line.split(',')
                        result.append(line)
        result = np.array(result)
        result = result.astype(float)
        return result

def read_file_r(filename):
        result = []
        with open(filename, 'r') as f:
                for line in f:
                        line = line.strip('\n')
```

```
                result.append(float(line))
        return result

# calculate w
def calculate_w(lam, tr, tr_r):
        feature = len(tr[0])
        example = len(tr)
        iden = np.identity(feature)
        iden = np.dot(lam, iden)
        tr_trans = np.transpose(tr)
        sum_1 = np.add(np.dot(tr_trans,tr),iden)
        sum_inverse = inv(sum_1)
        prod = np.dot(sum_inverse, tr_trans)
        w = np.dot(prod,tr_r)
        return w

# calculate mse
def calculate_mse(w, t, t_r):
        N = len(t)
        pro = np.dot(t, w)
        dif = np.subtract(pro, t_r)
        squ = np.square(dif)
        mse = np.mean(squ)
        return mse

# cross validation to pick lambda
def cross_val(tr, tr_r):
        tr_size = len(tr)
        interval = tr_size / 10
        lam_mse = []  # record mse_avg for every lam
        for lam in range(151):
                mse_list = [] # record mse for every splits
                for i in range(10):
                        start_index = i * interval
                        if i == 9:
                                end_index = tr_size
                        else:
                                end_index = (i+1) * interval
                        tr_1 = tr[0:start_index]
                        tr_r_1 = tr_r[0:start_index]
                        train = []
                        train_r = []
                        train_test = tr[start_index:end_index]
                        train_test_r = tr_r[start_index:end_index]
```

```
                tr_2 = tr[end_index:]
                tr_r_2 = tr_r[end_index:]
                train = np.concatenate((tr_1, tr_2))
                train_r = np.array(tr_r_1 + tr_r_2)
                w = calculate_w(lam, train, train_r)
                mse = calculate_mse(w, train_test, train_test_r)
                mse_list.append(mse)
            mse_avg = np.mean(mse_list)
            lam_mse.append(mse_avg)

    # return lam with minimum mse value
    lam_min = lam_mse.index(min(lam_mse))
    return lam_min

def Regularization(tr, tr_r, te, te_r):
    global lam_list
    lam = cross_val(tr, tr_r)
    lam_list.append(lam)
    w = calculate_w(lam, tr, tr_r)
    mse_te = calculate_mse(w, te, te_r)
    return mse_te

def main():
    global lam_list
    tr_file = ['train-crime.csv', 'train-wine.csv','train-100-10.csv', 'train-100-100.csv', 'train-
1000-100.csv']
    tr_r_file = ['trainR-crime.csv', 'trainR-wine.csv','trainR-100-10.csv', 'trainR-100-100.csv',
'trainR-1000-100.csv']
    te_file = ['test-crime.csv', 'test-wine.csv','test-100-10.csv', 'test-100-100.csv', 'test-1000-
100.csv']
    te_r_file = ['testR-crime.csv', 'testR-wine.csv','testR-100-10.csv', 'testR-100-100.csv',
'testR-1000-100.csv']

    te_mse = []
    file_number = len(tr_file)
    run_time = []
    for i in range(file_number):
    # start time
        a = datetime.datetime.now()
        tr = read_file(tr_file[i])
        tr_r = read_file_r(tr_r_file[i])
        te = read_file(te_file[i])
        te_r = read_file_r(te_r_file[i])
        if tr_file[i] == 'train-1000-100.csv':
```

```
                MSE_te = Regularization(tr, tr_r, te, te_r)
                te_mse.append(MSE_te)
                b = datetime.datetime.now()
                run_time.append((b-a).total_seconds())
                MSE_te = Regularization(tr[0:50], tr_r[0:50], te, te_r)
                te_mse.append(MSE_te)
                b = datetime.datetime.now()
                run_time.append((b-a).total_seconds())
                MSE_te = Regularization(tr[0:100], tr_r[0:100], te, te_r)
                te_mse.append(MSE_te)
                b = datetime.datetime.now()
                run_time.append((b-a).total_seconds())
                MSE_te = Regularization(tr[0:150], tr_r[0:150], te, te_r)
                te_mse.append(MSE_te)
                b = datetime.datetime.now()
                run_time.append((b-a).total_seconds())
                # Task 1 Regularization
        else:
                MSE_te = Regularization(tr, tr_r, te, te_r)
                te_mse.append(MSE_te)
                b = datetime.datetime.now()
                run_time.append((b-a).total_seconds())

    #print "run time:"
    #print run_time
    print "lam list"
    print lam_list
    print "test mse is:"
    print te_mse


    # run time: [1.850564, 0.214724, 0.128529, 1.296833, 3.708794, 4.776038, 6.029359,
7.403829]
    #lam list :[150, 2, 12, 20, 39, 24, 30, 46]
    #test mse is:[0.39233899203438133, 0.62530884230477923, 4.1757091596711433,
5.0808888179185328, 4.3227223508824659, 5.9344653841860655, 5.2599827410154809,
4.9341926077600835]
main()

# Task 4
# comp136 project2 Task 4
# beibei du
# 10/27/2016
import os
```

```
import sys
import math
import numpy as np
from numpy.linalg import inv
from numpy import linalg as la
import datetime

counter_list = []
# store run time
run_time = []
def read_file(filename):
	result = []
	with open(filename, 'r') as f:
		for line in f:
			line = line.strip('\n')
			line = line.split(',')
			result.append(line)
	result = np.array(result)
	result = result.astype(float)
	return result

def read_file_r(filename):
	result = []
	with open(filename, 'r') as f:
		for line in f:
			line = line.strip('\n')
			result.append(float(line))
	return result

# calculate w
def calculate_w(lam, tr, tr_r):
	feature = len(tr[0])
	example = len(tr)
	iden = np.identity(feature)
	iden = np.dot(lam, iden)
	tr_trans = np.transpose(tr)
	sum_1 = np.add(np.dot(tr_trans,tr),iden)
	sum_inverse = inv(sum_1)
	prod = np.dot(sum_inverse, tr_trans)
	w = np.dot(prod,tr_r)
	return w

# calculate mse
def calculate_mse(w, t, t_r):
```

```
    N = len(t)
    pro = np.dot(t, w)
    dif = np.subtract(pro, t_r)
    squ = np.square(dif)
    mse = np.mean(squ)
    return mse

def select_mN (tr, tr_r, alpha, beta):
    # get eigenvalue
    tr_trans = np.transpose(tr)
    pro = np.dot(tr_trans, tr)
    beta_phi_pro = np.dot(beta, pro)
    lamdas = la.eigvals(beta_phi_pro)


    # get transpose of sn <3.54>
    size = len(beta_phi_pro)
    iden = np.identity(size)
    sN_inverse = np.add(np.dot(alpha, iden), beta_phi_pro)
    sN =  inv(sN_inverse)

    # get Mn <3.53>
    phi_t =  np.dot(tr_trans, tr_r)
    sN_phi_t = np.dot(sN, phi_t)
    mN = np.dot(beta, sN_phi_t)

    # get gamma <3.91>
    gamma = 0.0
    for lam in lamdas:
        if isinstance(lam, complex):
            index = np.where(lamdas == lam)
            lam = lam.real
            lamdas[index] = lam
        gamma += (float(lam) / float(lam + alpha))

    # get alpha <3.92>
    mN_trans = np.transpose(mN)
    mN_pro = np.dot(mN_trans, mN)
    alpha_new = gamma / mN_pro

    # get beta (3.95>
    N = len(tr)
    mN_phi = np.dot(tr, mN)
    diff = np.subtract(tr_r, mN_phi)
```

```
        sum_squ = np.sum(np.square(diff))
        beta_new = sum_squ / float(N - gamma)
        beta_new = np.reciprocal(beta_new)

        return mN, alpha_new, beta_new


# modle select to pick mN (w)
def model_select(tr, tr_r):
        global counter_list
        alpha = 1.0
        beta = 1.0
        mN, alpha_new, beta_new= select_mN(tr, tr_r, alpha, beta)
        count = 0
        while (abs(alpha - alpha_new) >= 0.00001 ) and (abs(beta - beta_new) >= 0.00001):
                alpha, beta = alpha_new, beta_new
                mN, alpha_new, beta_new= select_mN(tr, tr_r, alpha, beta)
                count = count + 1
        else:
                counter_list.append(count)
                return mN


def Regularization(tr, tr_r, te, te_r):
        w = model_select(tr, tr_r)
        mse_te = calculate_mse(w, te, te_r)
        return mse_te

def main():
        global counter_list
        global run_time
        tr_file = ['train-crime.csv', 'train-wine.csv','train-100-10.csv', 'train-100-100.csv','train-
1000-100.csv']
        tr_r_file = ['trainR-crime.csv', 'trainR-wine.csv', 'trainR-100-10.csv', 'trainR-100-
100.csv','trainR-1000-100.csv']
        te_file = ['test-crime.csv', 'test-wine.csv','test-100-10.csv','test-100-100.csv', 'test-1000-
100.csv']
        te_r_file = ['testR-crime.csv', 'testR-wine.csv', 'testR-100-10.csv', 'testR-100-100.csv',
'testR-1000-100.csv']


        te_mse = []

        file_number = len(tr_file)
```

```
        for i in range(file_number):
                a = datetime.datetime.now()
                tr = read_file(tr_file[i])
                tr_r = read_file_r(tr_r_file[i])
                te = read_file(te_file[i])
                te_r = read_file_r(te_r_file[i])
                if tr_file[i] == 'train-1000-100.csv':
                        MSE_te = Regularization(tr, tr_r, te, te_r)
                        te_mse.append(MSE_te)
                        b = datetime.datetime.now()
                        run_time.append((b-a).total_seconds())
                        MSE_te = Regularization(tr[0:50], tr_r[0:50], te, te_r)
                        te_mse.append(MSE_te)
                        b = datetime.datetime.now()
                        run_time.append((b-a).total_seconds())
                        MSE_te = Regularization(tr[0:100], tr_r[0:100], te, te_r)
                        te_mse.append(MSE_te)
                        b = datetime.datetime.now()
                        run_time.append((b-a).total_seconds())
                        MSE_te = Regularization(tr[0:150], tr_r[0:150], te, te_r)
                        te_mse.append(MSE_te)
                        b = datetime.datetime.now()
                        run_time.append((b-a).total_seconds())

                else:
                        MSE_te = Regularization(tr, tr_r, te, te_r)
                        te_mse.append(MSE_te)
                        b = datetime.datetime.now()
                        run_time.append((b-a).total_seconds())
        #print tr_file
        #print "counter:"
        #print counter_list
        #print "run time:"
        #print run_time
        print "test mse is:"
        print te_mse
# run time: [0.151428, 0.034536, 0.007779, 0.105999, 0.117914, 0.154577, 0.183432, 0.19713]
# [11, 13, 3, 13, 3, 8, 6, 2]
#[0.39110220806201251, 0.6267448249308214, 4.1801330312023159, 7.3525369250732879,
4.3383499509291168, 5.7895752937487694, 5.7339307345248693, 5.2489966154820733]
main()
```

```
# Plot.py
# figure analysi for task3, task4, task5

import matplotlib.pyplot as plt
import numpy as np

s = ['crime', 'wine', '100-10', '100-100', '1000-100', '50(1000)-100', '100(1000)-100', '150(1000)-
100']
# part 1 test mse
y1
=[ 0.389023387713,0.625308842305,4.15967850948,5.07829980059,4.31557063032,5.540902
22919,5.20591195733,4.84894305335]
lambda_1 = [75,2,8,22,27,8,19,23]
# cross validation mse
y2 = [0.39227040752452375, 0.62530884230477923, 4.1757091596711433,
5.0808888179185328, 4.3227223508824659, 5.9344653841860655, 5.2599827410154809,
4.9341926077600835]
lambda_2 = [150,2,12,20,39,24,30,46]
# model selection mse
y3 = [0.39110220806201251, 0.6267448249308214, 4.1801330312023159,
7.3525369250732879, 4.3383499509291168, 5.7895752937487694, 5.7339307345248693,
5.2489966154820733]

x = range(8)
plt.figure(1)
plt.plot(x, y1, 'r', marker = '*')
plt.plot(x, y2, 'g', marker = 's')
plt.plot(x, y3, 'b', marker = 'o')
plt.xticks(x, s, rotation = 10)
plt.legend(['regularization', 'cross validation', 'model selection'], loc = 2)
plt.ylabel('MSE')
plt.xlabel('data set')
plt.show()

plt.figure(8)
diff = list(np.array(y3) - np.array(y2))
plt.plot(x, diff, 'r', marker = '*')
plt.xticks(x, s, rotation = 10)
plt.legend(['model selection - cross validation'], loc = 2)
plt.ylabel('MSE difference')
plt.xlabel('data set')
plt.show()
```

```
# task 3 compare MSE of part1 and cross validation
plt.figure(2)
plt.plot(x, y1, 'r', marker = '*')
plt.plot(x, y2, 'g', marker = 's')
plt.xticks(x, s, rotation = 10)
plt.legend(['regularization', 'cross validation'], loc = 2)
plt.ylabel('MSE')
plt.xlabel('data set')
plt.show()


# Task 3 in terms of lambda
plt.figure(3)
plt.plot(x, lambda_1, 'r', marker = '*')
plt.plot(x, lambda_2, 'g', marker = 's')
plt.xticks(x, s, rotation = 10)
plt.legend(['regularization', 'cross validation'], loc = 2)
plt.ylabel('Optimal lambda')
plt.xlabel('data set')
plt.show()

# Task 3 in fixed number of features
plt.figure(4)
x1 = range(5)
s1 = ['50(1000)-100', '100(1000)-100', '150(1000)-100', '1000-100', '100-100']
y1_1 =[ 5.54090222919,5.20591195733,4.84894305335,4.31557063032,5.07829980059]
y2_1 = [5.9344653841860655, 5.2599827410154809,
4.9341926077600835,4.3227223508824659,5.0808888179185328]
plt.plot(x1, y1_1, 'r', marker = '*')
plt.plot(x1, y2_1, 'g', marker = 's')
plt.xticks(x1, s1, rotation = 10)
plt.legend(['regularization', 'cross validation'], loc = 0)
plt.ylabel('TEST MSE')
plt.xlabel('data set')
plt.show()

plt.figure(5)
y1_lambda = [8,19,23,27,22]
y2_lambda = [24,30,46,39, 20]
plt.plot(x1, y1_lambda, 'r', marker = '*')
plt.plot(x1, y2_lambda, 'g', marker = 's')
plt.xticks(x1, s1, rotation = 10)
plt.legend(['regularization', 'cross validation'], loc = 0)
```

```
plt.ylabel('lambda')
plt.xlabel('data set')
plt.show()


# Task4 compare MSE of model selection and part1
plt.figure(6)
plt.plot(x, y1, 'r', marker = '*')
plt.plot(x, y3, 'g', marker = 's')
plt.xticks(x, s, rotation = 10)
plt.legend(['regularization', 'model selection'], loc = 2)
plt.ylabel('MSE')
plt.xlabel('data set')
plt.show()

# Task4 fixed number of features
plt.figure(7)
x1 = range(5)
s1 = ['50(1000)-100', '100(1000)-100', '150(1000)-100', '1000-100', '100-100']
y1_1 =[ 5.54090222919,5.20591195733,4.84894305335,4.31557063032, 5.07829980059]
y3_1 = [5.7895752937487694, 5.7339307345248693,
5.2489966154820733,4.3383499509291168,7.3525369250732879]
plt.plot(x1, y1_1, 'r', marker = '*')
plt.plot(x1, y3_1, 'g', marker = 's')
plt.xticks(x1, s1, rotation = 10)
plt.legend(['regularization', 'model selection'], loc = 0)
plt.ylabel('TEST MSE')
plt.xlabel('data set')
plt.show()
```