6.4

$$\nabla g \left[\frac{\partial}{\partial b} g \quad \frac{\partial}{\partial w_1} g \cdots \cdots \frac{\partial}{\partial w_M} g \quad \frac{\partial}{\partial c_1} g \cdots \frac{\partial}{\partial c_M} g \cdots \nabla_{v_1}^T g \cdots \nabla_{v_M}^T g \right]^T$$

$$g(\tilde{w}) g = \sum_{p=1}^{P} \log\left(1 + e^{-y_p \tilde{f}_p^T \tilde{w}}\right)$$

$$\cancel{= \sum_{p=1}^{P} \log\left(1 + e^{-y_p \tilde{f}_p^T (a(c_m + x_p^T v_m)) \tilde{w}}\right)}$$

$$= \sum_{p=1}^{P} \log\left(1 + e^{-y_p (b + \sum_{m=1}^{M} a(c_m + x_p^T v_m)\tilde{w})}\right)$$

$$\frac{\partial g}{\partial b} = \sum_{p=1}^{P} \frac{+1}{1 + e^{-y_p(b + \sum_{m=1}^{M} a(c_m + x_p^T v_m)\tilde{w}_n)}} e^{-y_p(b + \sum_{m=1}^{M} a(c_m + x_p^T v_m)\tilde{w}_n)} - y_p$$

$$= \sum_{p=1}^{P} \frac{1}{1 + \frac{1}{e^{y_p(b + \sum_{m=1}^{M} a(c_m + x_p^T v_m)\tilde{w}_n)}}} - y_p \, e^{-y_p(b + \sum_{m=1}^{M} a(c_m + x_p^T v_m)\tilde{w}_n)}$$

$$= \sum_{p=1}^{P} \frac{e^{y_p(b + a(c_m + x_p^T v_m)\tilde{w}_n)}}{e^{y_p(b + \sum_{m=1}^{M} a(c_m + x_p^T v_m)\tilde{w}_n)} + 1} - y_p \, e^{-y_p(b + \sum_{m=1}^{M} a(c_m + x_p^T v_m)\tilde{w}_n)}$$

$$\frac{\partial g}{\partial b} = \sum_{p=1}^{P} -\sigma\left(-y_p(b + a(c_m + x_p^T v_m)\tilde{w}_n\right) y_p$$

$$g = \sum_{p=1}^{P} \log\left(1 + e^{-y_p\left(b + \sum_{m=1}^{M} a(c_m + x_p^T v_m)\tilde{w}_n\right)}\right)$$

$$\frac{\partial g}{\partial w_n} = \frac{1}{1 + e^{-y_p\left(b + \sum_{m=1}^{M} a(c_m + x_p^T v_m)\tilde{w}_n\right)}} \, e^{-y_p\left(b + \sum_{m=1}^{M} a(c_m + x_p^T v_m)\tilde{w}_n\right)} \; -y_p \sum_{m=1}^{M} a'(c_m + x_p^T v_m)\, g$$

$$= -\sigma\left(-y_p\left(b + \sum_{m=1}^{M} a(c_m + x_p^T v_m)\tilde{w}_n\right)\right) a'(c_n + x_p^T v_n)\, y_p$$

$$\frac{\partial g}{\partial c_n} = \sum_{p=1}^{P} \frac{1}{1 + e^{-y_p\left(b + \sum_{m=1}^{M} a(c_m + x_p^T v_m)\tilde{w}_n\right)}} \, e^{-y_p\left(b + \sum_{m=1}^{M} a(c_m + x_p^T v_m)\tilde{w}_n\right)} \; -y_p\, a'(c_n + x_p^T v_n)\tilde{W}_n$$

$$= \sum_{p=1}^{P} \sigma\left(-y_p\left(b + \sum_{m=1}^{M} a(c_m + x_p^T v_m)\tilde{w}_n\right)\right) y_p\, a'(c_n + x_p^T v_n)\, w_n$$

$$\nabla_{v_n} g = \frac{1}{1 + e^{-y_p\left(b + \sum_{m=1}^{M} a(c_m + x_p^T v_m)\tilde{w}_n\right)}} \, e^{-y_p\left(b + \sum_{m=1}^{M} a(c_m + x_p^T v_m)\tilde{w}_n\right)} \; a'(c_n + x_p^T v_n)\, x_p \hat{W}_n\, y_p$$

$$= \sum_{p=1}^{P} \sigma\left(-y_p\left(b + \sum_{m=1}^{N} a(c_m + x_p^T v_m)\tilde{w}_n\right)\right) a'(c_n + x_p^T v_n)\, x_p^T W_n\, y_p$$

Scanned by CamScanner

Exercise 6.4 b

let $a = \tanh(\cdot)$
$a' = \mathrm{sech}^2(\cdot)$

To prove : $\frac{\partial}{\partial b} g = -1^T q \odot y$

$$\frac{\partial}{\partial w_n} g = -1^T (q \odot t_n \odot y)$$

$$\frac{\partial}{\partial c_n} g = -1^T (q \odot s_n \odot y) w_n$$

$$\nabla_{v_n} g = -X \cdot q \odot s_n \odot y w_n$$

$$q_p = \sigma\left(-y_p \left(b + \sum_{m=1}^{M} w_m \tanh(c_m + x_p^T v_m)\right)\right) \quad ①$$

$$t_{np} = \tanh(c_n + x_p^T v_n) \quad ②$$

$$s_{np} = \mathrm{sech}^2(c_n + x_p^T v_n) \quad ③$$

$$\frac{\partial g}{\partial b} = -\sum_{p=1}^{P} \sigma\left(-y_p \left(b + \sum_{m=1}^{M} w_m a(c_m + x_p^T v_m)\right)\right) y_p$$

$$= -\sum_{p=1}^{P} q_p y_p \quad \text{from } ①$$

$$= -1_{P\times 1}^T q \odot y.$$

$$\frac{\partial}{\partial w_n} g = -\sum_{p=1}^{P} \sigma\left(-y_p \left(b + \sum_{m=1}^{M} w_m a(c_m + x_p^T v_m)\right)\right) a(c_n + x_p^T v_n) y_p$$

$$= -\sum_{p=1}^{P} q_p \cancel{y_p} t_{np} y_p$$

$$= -1_{P\times 1}^T (q \odot t_n \odot y)$$

Scanned by CamScanner

$$\frac{\partial}{\partial c_n} g = -\sum_{p=1}^{P} \sigma\left(-y_p\left(b + \sum_{m=1}^{M} w_m\, a\left(c_m + x_p^T v_m\right)\right)\right) a'\left(c_n + x_p^T v_n\right) w_n y_p$$

$$= -\sum_{p=1}^{P} q_p s_{np} w_n y_p$$

$$= -1_{P\times 1}^{T}\left(q \odot s_n \odot y\right) w_n$$

$$\nabla_{v_n} g = -\sum_{p=1}^{P} \sigma\left(-y_p\left(b + \sum_{m=1}^{M} w_m\, a\left(c_m + x_p^T v_m\right)\right)\right) a'\left(c_n + x_p^T v_n\right) x_p w_n y_p$$

$$= -\sum_{p=1}^{P} q_p s_{np} \cdot \bar{x}_p w_n y_p$$

$$= -X\left(\bar{q} \odot \bar{s}_n \odot \bar{y}\right) w_n$$

$$= -X \cdot q \odot s_n \odot y\, w_n$$

Exercise 6.5

```
# This file is associated with the book
# "Machine Learning Refined", Cambridge University Press, 2016.
# by Jeremy Watt, Reza Borhani, and Aggelos Katsaggelos.
from __future__ import division
import numpy as np
import numpy.matlib as nm
import matplotlib.pyplot as plt
from sympy import *
import math

def single_layer_classification_hw():

    # load data

    global M
    M=[4,6,7,8,10,15]
    for i in M:
        #M = 4 # number of hidden units
        X, y = load_data()
        M=int(i)
        print("for M",i)
        b,w,c,V = tanh_softmax(X.T,y,M)
        plot_separator(b,w,c,V,X,y)
        plt.show()


### gradient descent for single layer tanh nn ###
def tanh_softmax(X,y,M):
```

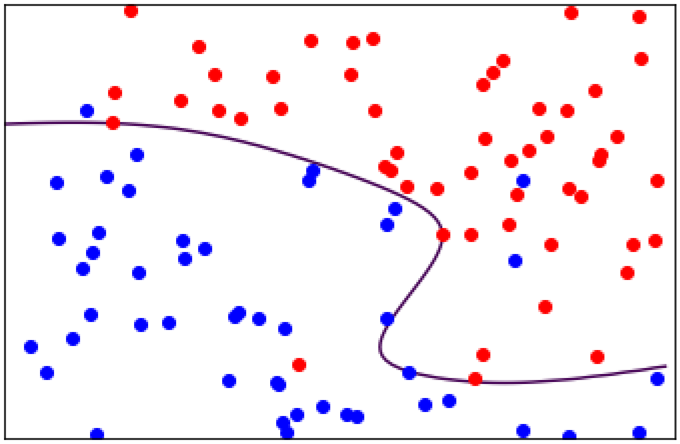
```
y = np.reshape(y,(np.size(y),1))

# initializations
N = np.shape(X)[0]
P = np.shape(X)[1]

b = np.random.randn()
w = np.random.randn(M,1)
c = np.random.randn(M,1)
V = np.random.randn(N,M)
l_P = np.ones((P,1))
# stoppers
max_its = 10000
grad = 1
count = 1

### main ###
while (count <= max_its) & (np.linalg.norm(grad) > 1e-5):
    F = obj(c, V, X)
    q = sigmoid(-y * (b * l_P + np.dot(F.T, w)))
    new_F = obj1(c, V, X)
    q1 = nm.repmat(q, 1, M)
    y1 = nm.repmat(y, 1, M)
    w1 = nm.repmat(w.T, N, 1)
    t1 = F.T
    s1 = new_F.T

    grad_b = - np.dot(l_P.T,(q*y))
    grad_w = - np.dot(l_P.T,q1*F.T*y1).T
    grad_c = - (np.dot(l_P.T,q1*s1*y1)).T * w
```

```
        grad_V = np.dot(-X, (q1 * s1 * y1)) * w1



        # determine steplength
        alpha = 1e-2

        # take gradient steps
        b = b - alpha*grad_b
        w = w - alpha*grad_w
        c = c - alpha*grad_c
        V = V - alpha*grad_V

        # update stoppers
        count = count + 1

    return b, w, c, V



### load data
def load_data():

    data = np.array(np.genfromtxt('genreg_data.csv', delimiter=','))
    A = data[:,0:-1]
    b = data[:,-1]

    # plot data
    ind = np.nonzero(b==1)[0]
    plt.plot(A[ind,0],A[ind,1],'ro')
    ind = np.nonzero(b==-1)[0]
    plt.plot(A[ind,0],A[ind,1],'bo')
```

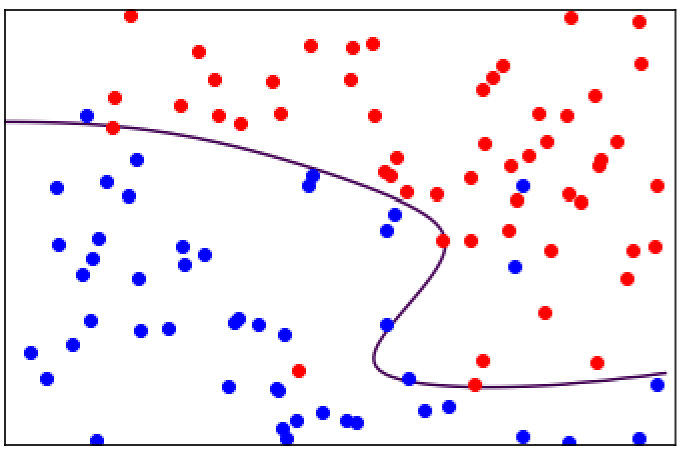
```
    plt.hold(True)
    return A,b



def sigmoid(z):
    return 1/(1+np.exp(-z))


# plot the seprator + surface
def plot_separator(b,w,c,V,X,y):

    s = np.arange(-1,1,.01)
    s1, s2 = np.meshgrid(s,s)

    s1 = np.reshape(s1,(np.size(s1),1))
    s2 = np.reshape(s2,(np.size(s2),1))
    g = np.zeros((np.size(s1),1))

    t = np.zeros((2,1))
    for i in np.arange(0,np.size(s1)):
        t[0] = s1[i]
        t[1] = s2[i]
        F = obj(c,V,t)
        g[i] = np.tanh(b + np.dot(F.T,w))

    s1 = np.reshape(s1,(np.size(s),np.size(s)))
    s2 = np.reshape(s2,(np.size(s),np.size(s)))
    g = np.reshape(g,(np.size(s),np.size(s)))
```

```
    # plot contour in original space
    plt.contour(s1,s2,g,1,color = 'k')
    plt.gca().xaxis.set_major_locator(plt.NullLocator())
    plt.gca().yaxis.set_major_locator(plt.NullLocator())
    plt.xlim(0,1)
    plt.ylim(0,1)
    plt.hold(true)



def obj(z,H,A):
    F = np.zeros((M,np.shape(A)[1]))
    for p in np.arange(0,np.shape(A)[1]):
        F[:,p] = np.ravel(np.tanh(z + np.dot(H.T,np.reshape(A[:,p],(np.shape(A)[0],1)))))

    return F


def obj1(z,H,A):
    F = np.zeros((M,np.shape(A)[1]))
    for p in np.arange(0,np.shape(A)[1]):
        F[:,p] = np.ravel(1/np.cosh(z + np.dot(H.T,np.reshape(A[:,p],(np.shape(A)[0],1))))**2)

    return F


single_layer_classification_hw()
```

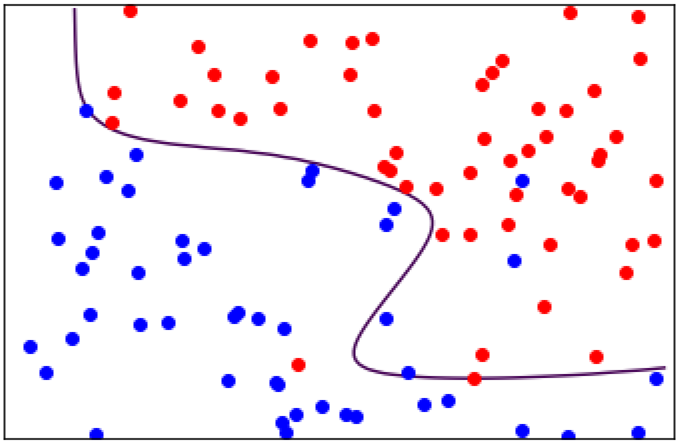
Diagram – M=4



for M 6

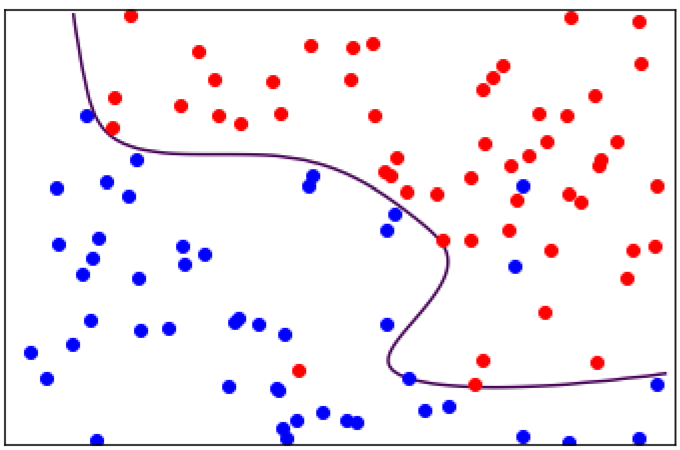


for M 7

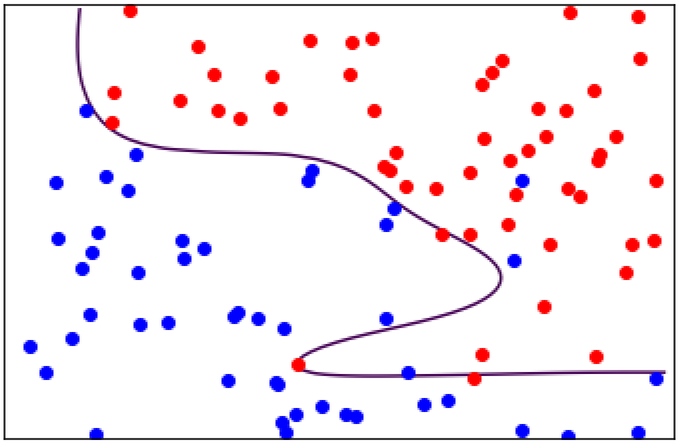


-

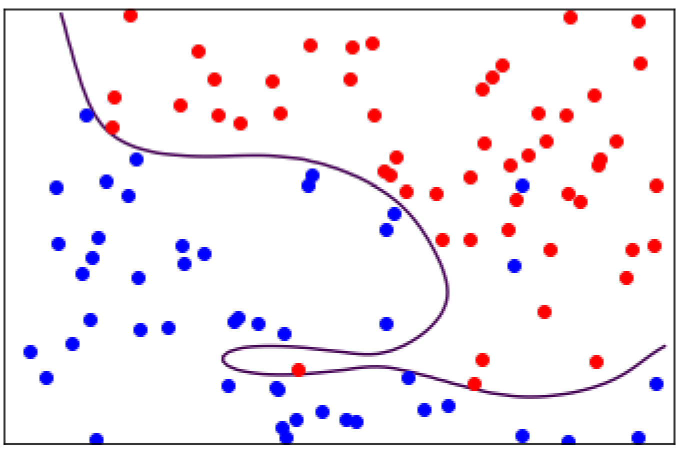
for M 8



for M 10



for M 15



As we can see from above plot diagrams for M=7 is better fit than M=4. Also, there is overfitting of data when M increases.

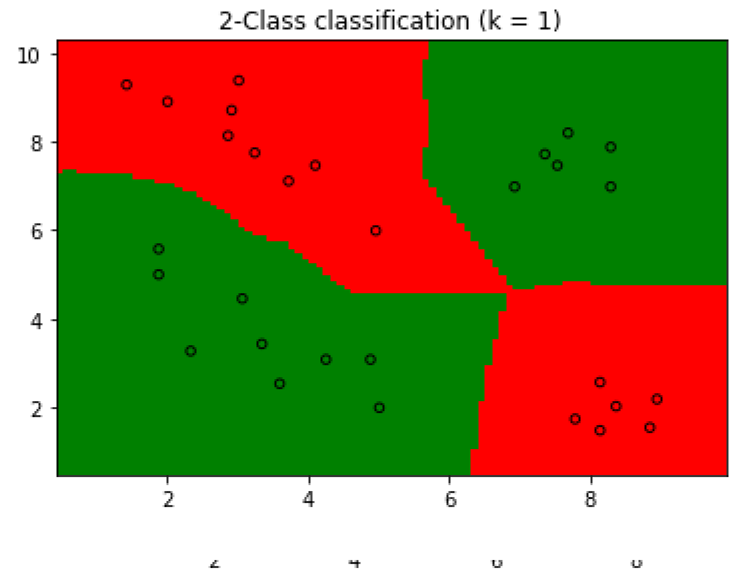
Exercise 6.6

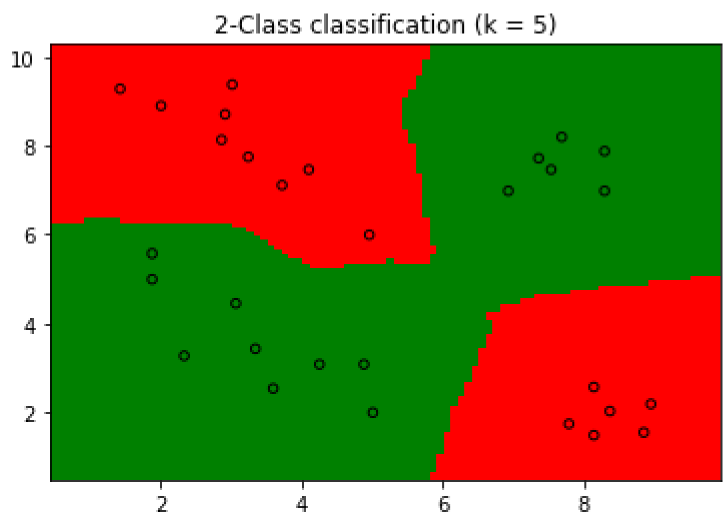
Referred Sklearn website for plotting boundary decisions

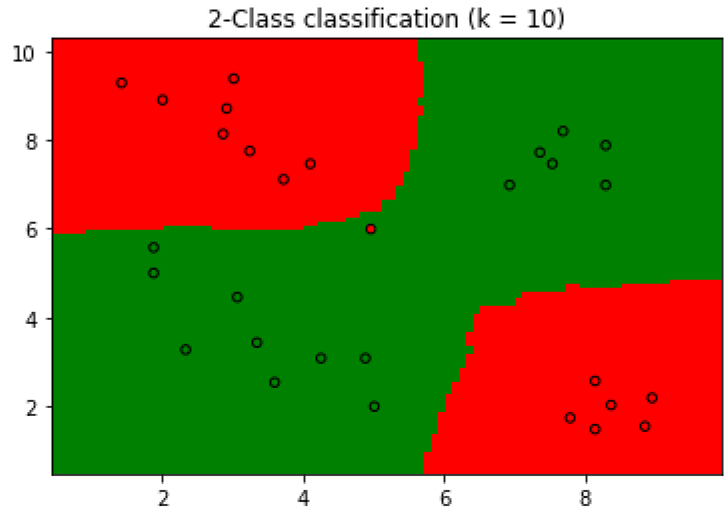
(http://scikit-learn.org/stable/auto_examples/neighbors/plot_classification.html)

```
import numpy as np

import numpy.matlib

import matplotlib.pyplot as plt

from __future__ import division

import pylab

from sklearn.model_selection import KFold

from sklearn.cross_validation import train_test_split

from sklearn.neighbors import KNeighborsClassifier

import matplotlib.pyplot as plt

from matplotlib.colors import ListedColormap

from sklearn import neighbors, datasets


### load data ###

def load_data():

    # load data from file

    data = np.array(np.genfromtxt('knn_data.csv', delimiter=','))

    X = data[:,0:-1]

    y = data[:,-1]


    return X,y


X,y = load_data()

h = .1  # step size in the mesh


# Create color maps

light = ListedColormap(['green', 'red'])

bold = ListedColormap(['green', 'red'])
```

```
a=[1,5,10]
for n_neighbors in a:
    # we create an instance of Neighbours Classifier and fit the data.
    k_nn= neighbors.KNeighborsClassifier(n_neighbors)
    k_nn.fit(X, y)

    # Plot the decision boundary. For that, we will assign a color to each
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),np.arange(y_min, y_max, h))
    Z = k_nn.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.figure()
    plt.pcolormesh(xx, yy, Z, cmap=light)

    # Plot also the training points
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=bold,edgecolor='k', s=20)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.title("2-Class classification (k = %i)"
              % (n_neighbors))

    plt.show()
```


2-Class classification (k = 1)
10
8
6
4
2
2
4
6
8


2-Class classification (k = 5)
10
8
6
4
2
2
4
6
8



Exercise 6.9

```
# This file is associated with the book
# "Machine Learning Refined", Cambridge University Press, 2016.
# by Jeremy Watt, Reza Borhani, and Aggelos Katsaggelos.

import numpy as np
import numpy.matlib
import matplotlib.pyplot as plt
from __future__ import division
import pylab
from sklearn.model_selection import KFold
from sklearn.cross_validation import train_test_split


### load data ###
def load_data():
    # load data from file
    data = np.array(np.genfromtxt('2eggs_data.csv', delimiter=','))
```

```
    X = data[:,0:-1]
    y = data[:,-1]

    return X,y

def sigmoid(z):
    return 1/(1+np.exp(-z))

def poly_features(X,D):
    o = np.ones((np.shape(X)[0],1))
    F = o
    for i in range(1,D+1):
        temp = np.array((X[:,0]**0)*(X[:,1]**(i-0)))
        for j in range(1,i+1):
            temp2 = np.array((X[:,0]**j)*(X[:,1]**(i-j)))
            temp = np.column_stack((temp,temp2))
        F = np.column_stack((F,temp))
    return F

def poly_features1(X,D):
    o = np.ones((np.shape(X)[0],1))
    F = o
    for i in range(1,D+1):
        temp = np.array((X[:,0]**0)*(X[:,1]**(i-0)))
        for j in range(1,i+1):
            temp2 = np.array((X[:,0]**j)*(X[:,1]**(i-j)))
            temp = np.column_stack((temp,temp2))
        F = np.column_stack((F,temp))
```

```
    return F

### plots learned model ###
def plot_poly(w,deg):
    # Generate poly seperator
    o = np.arange(-2,10,.01)
    s, t = np.meshgrid(o,o)
    s = np.reshape(s,(np.size(s),1))
    t = np.reshape(t,(np.size(t),1))
    f = np.zeros((np.size(s),1))
    count = 0

    for n in np.arange(0,deg+1):
        for m in np.arange(0,deg+1):
            if (n + m <= deg):
                f = f + w[count]*((s**n)*(t**m))
                count = count + 1

    s = np.reshape(s,(np.size(o),np.size(o)))
    t = np.reshape(t,(np.size(o),np.size(o)))
    f = np.reshape(f,(np.size(o),np.size(o)))

    # plot contour in original space
    plt.contour(s,t,f,1, colors =color)

    plt.gca().xaxis.set_major_locator(plt.NullLocator())
    plt.gca().yaxis.set_major_locator(plt.NullLocator())
    plt.axis('equal')
```

```
def plot_data(A,b,deg):
    for i in np.arange(1,9):
        plt.subplot(2,4,i)

        # plot data
        ind = np.nonzero(b==1)[0]
        plt.plot(A[ind,0],A[ind,1],'ro')
        ind = np.nonzero(b==-1)[0]
        plt.plot(A[ind,0],A[ind,1],'bo')
        plt.hold(True)

        # graph info labels
        s = 'D = ' + str(deg[i-1])
        plt.title(s, fontsize=15)
        plt.axis('off')

# plot mse's over all D tested
def plot_mse(mses,mses1,deg):
    plt.plot(np.arange(1,np.size(mses)+1),mses,'ro--')
    plt.plot(np.arange(1,np.size(mses1)+1),mses1,'.-')
    plt.title('MSE on entire dataset in D', fontsize=18)
    plt.xlabel('degree D', fontsize=18)
    plt.ylabel('MSE       ', fontsize=18)




# run over all the degrees, fit each models, and calculate errors
def try_all_degs(A,b,deg_range):
    colors = ['m','b','r','c']
    # plot datapoints - one panel for each deg in deg_range
```

```
fig = plt.figure(figsize = (5,5))
plot_data(x,y,deg_range)
global M
M = 4     # number of hidden units


A,b=load_data()

N = np.shape(A)[0]
P = np.shape(A)[1]
c = np.random.randn(M,1)
V = (np.random.randn(N,M))
A=x
b=y
colors = ['m','b','r','c']


# generate nonlinear feature


kf = KFold(n_splits=3)
KFold(n_splits=3)
k=3

mse_Train=[]
mse1_Test=[]
mse_Train = np.zeros((np.size(deg_range)))
mse1_Test = np.zeros((np.size(deg_range)))

for D in np.arange(0,len(deg_range)):
```

```
# generate poly feature transformation
    for train_index, test_index in kf.split(x):
        X_train, X_test = x[train_index], x[test_index]
        Y_train, Y_test = y[train_index], y[test_index]

        F = poly_features(X_train,deg_range[D])

        F1 = poly_features1(X_test,deg_range[D])

        temp = np.linalg.pinv(np.dot(F,F.T))
        w = np.dot(np.dot(temp,F).T,Y_train)
        mse = np.linalg.norm(np.dot(F,w)-Y_train)/(np.size(Y_train))
        mse_Train[D]+=mse



        temp1 = np.linalg.pinv(np.dot(F1,F1.T))

        mse1 = np.linalg.norm(np.dot(F1,w)-Y_test)/(np.size(Y_test))

        mse1_Test[D] +=mse1



        # plot fit to data
        plt.subplot(2,4,D+1)
        #b,w,c,V = tanh_softmax(X.T,y,M)
        #plot_poly(w,deg_range[D])
```

```
        mse_Train[D]=mse_Train[D]/k

        mse1_Test[D]=mse1_Test[D]/k



    fig = plt.figure(figsize = (5,5))

    plot_mse(mse_Train,mse1_Test,deg_range)

    plt.show()


x,y=load_data()

deg_range = [1,2,3,4,5,6,7,8]         # degree polys to try

try_all_degs(x,y,deg_range)
```