# FinalProject

August 7, 2020

### 0.1 Final Project: Aya Eid

[1]:
```
# 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
from skimage import measure
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.axes_grid1.inset_locator import inset_axes

import warnings
import pylab
import numpy as np
import numpy.matlib
#import sklearn.preprocessing
import matplotlib.pyplot as plt
#import mpl_toolkits.mplot3d
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import StratifiedKFold

"""%matplotlib notebook"""
warnings.filterwarnings('ignore')
```

## 1 Model building helper functions:

These include loading data and manipulating it using various activation functions and feature transforms

[2]:
```
# load data
def load_data(filename):
    data = np.array(np.genfromtxt(filename, delimiter=','))
    x = data[:,:-1]# np.reshape(data[:,0:1],(np.size(data[:,0:1]),1))
    y = np.reshape(data[:,-1],(np.size(data[:,-1]),1))
    return x,y

def poly_features(x,D):
    """generates polynomial features with orders ranging from 0->D"""
    x.shape = (x.size,1)
    order = np.tile(np.arange(0,D+1), (x.size,1))
    F = np.power( np.tile(x, (1,D+1)), order) # []
    return F.T



def poly_features_multiDim(x,D):
    """generates polynomial features with orders ranging from 0->D"""
    #return x[:,0:D+1].T
   # F = np.zeros((D+1, x.shape[0]))
   # F[0,:] = np.ones((1,x.shape[0]))
   # F[1:,:] = x[:,0:D].T
   # print(D)

    #return F
    poly = PolynomialFeatures(D)
    F = poly.fit_transform(x)
    return F.T

def fourier_features(x,D):
    """takes fourier coefficients"""
    x.shape = (x.size,1)
    F = np.zeros((2*D+1, x.size))
    F[0,:] = np.ones((1,x.size))
    for i in np.arange(1,D+1):
        F[2*i-1,:] = np.cos(2*np.pi*x*i).flatten()
        F[2*i,:] = np.sin(2*np.pi*x*i).flatten()
    return F

def softmax_grad(F,y,w,nIter):
    alpha = 10**-2#10**-2
    for k in range(nIter):
        r = np.zeros((F.shape[1],1))
        for p in range(y.shape[0]):
            r = r + -1 * sigmoid(-y[p] * (F.T @ w)) * y[p]
        grad = F @ r
        w = w - alpha * grad
    return w

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


def partition_indices(length, n):
    indices = np.arange(length)
    np.random.shuffle(indices)
    q, r = divmod(length, n)
    stIndices = [q*i + min(i, r) for i in range(n+1)]
    return [indices[stIndices[i]:stIndices[i+1]] for i in range(n)]

def partition_indices_better(x, y, k):
    kf = StratifiedKFold(n_splits=k, random_state=None, shuffle=True)
    return [test_index for test_index, test_index in kf.split(x, y)]


def get_classification_res(yTrue, yPredicted):
    TP = np.logical_and(yTrue, yPredicted)
    TN = np.logical_and(np.logical_not(yTrue), np.logical_not(yPredicted))
    FP = np.logical_and(np.logical_not(yTrue), yPredicted)
    FN = np.logical_and(yTrue, np.logical_not(yPredicted))
    return TP, TN, FP, FN
```

## 2 Model visuation helper functions

Generalized for 2d and 3D data; plots data or the model performance

[4]:
```
# plot the polynomial
def plot_model_2D(w,D,y,ax,plot_color):
    # plot determined surface in 3d space
    s = np.arange(0, 1, .025)#.025)
    ns = s.size

    s.shape = (ns,1)
    sx, sy = np.meshgrid(s,s)
    sx2 = sx.flatten()
    sx2.shape = (sx2.size,1)
    sy2 = sy.flatten()
    sy2.shape = (sy2.size,1)
    s = np.concatenate((sx2,sy2),axis=1)

    f = poly_features_multiDim(s,D)
    z = np.dot(f.T,w)
    z = np.reshape(z,sx.shape)


    ax.contour(sx, sy, z, levels=[0])
    plt.axis([-.05,1.05,-.05,1.05])
    #plt.axis([-.1,2,-1,2])
    return ax

def plot_model_3D_fake(w,D,y,ax,plot_color):
    # plot determined surface in 3d space
    s = np.arange(0, 1, .025)#.025)
    ns = s.size

    s.shape = (ns,1)
    sx, sy, sz = np.meshgrid(s,s,s)
    sx2 = sx.flatten()
    sx2.shape = (sx2.size,1)
    sy2 = sy.flatten()
    sy2.shape = (sy2.size,1)
    sz2 = sz.flatten()
    sz2.shape = (sz2.size,1)
    s = np.concatenate((sx2,sy2,sz2),axis=1)

    f = poly_features_multiDim(s,D)
    z = np.dot(f.T,w)
    z = np.reshape(z,sx.shape)

    #print(z.shape)
    #ax.contour(np.mean(sx,axis=2), np.mean(sy,axis=2), np.mean(z,axis=2),␣
 ↪levels=[0])
  #  ax.contour(sx[:,:,1], sy[:,:,1], z[:,:,1], levels=[0])
  #  ax.contour(sx[:,:,20], sy[:,:,20], z[:,:,20], levels=[0])
  #  ax.contour(sx[:,:,-1], sy[:,:,-1], z[:,:,-1], levels=[0])
    plt.axis([-.05,1.05,-.05,1.05])
    #plt.axis([-.1,2,-1,2])
    return ax


def plot_model_3D(ax,w,D,y,plot_color):
    # plot determined surface in 3d space
    s = np.arange(0, 1, .025)
    ns = s.size

    s.shape = (ns,1)
    sx, sy, sz = np.meshgrid(s,s,s)
    sx2 = sx.flatten()
    sx2.shape = (sx2.size,1)
    sy2 = sy.flatten()
    sy2.shape = (sy2.size,1)
    sz2 = sz.flatten()
    sz2.shape = (sz2.size,1)
    s = np.concatenate((sx2,sy2,sz2),axis=1)

    f = poly_features_multiDim(s,D)
    z = np.dot(f.T,w)
    z = np.reshape(z,sx.shape)
    verts, faces, _, _ = measure.marching_cubes_lewiner(z, 0, spacing=(0.025, 0.
↪025, 0.025))#spacing=(0.1, 0.1, 0.1))


    ax.plot_trisurf(verts[:, 0], verts[:,1], faces, verts[:, 2],
                    color='r', lw=1, alpha = .5)

   # plt.axis([-.05,1.05,-.05,1.05])
    return ax

def plot_mse(ax, mses, deg, plot_color, legStr, titStr,xStr):
    plt.sca(ax)
    #plt.plot(np.arange(1,np.size(mses)+1),mses,'--', color = plot_color,␣
↪label=legStr)
    plt.plot(deg,mses,'--^', color = plot_color, label=legStr)
    plt.title(titStr, fontsize=28)
    plt.xlabel(xStr, fontsize=28)
    plt.ylabel('error', fontsize=28)
    plt.xlabel('degree D', fontsize=28)

# plot data
def plot_data_1D(x,y,i,plot_color, grid):
    plt.subplot(grid[int(np.floor(i/3)), (i % 3) ])
    plt.scatter(x,y,s = 30, color = plot_color)
    plt.axis([-.1,1.1,-.1,1.1])
    s = 'Hold out #' + str(i)
    plt.title(s, fontsize=15)

# plot data
def plot_data(x,i,plot_color, grid,sT=30, alphaT =1):
    plt.subplot(grid[int(np.floor(i/3)), (i % 3) ])
    plt.scatter(x[:,0],x[:,1],s = sT, color = plot_color, alpha=alphaT)
    plt.axis([-.1,1.1,-.1,1.1])
    s = 'Hold out #' + str(i)
    plt.title(s, fontsize=15)

# plot data 3D
def plot_data_3D(ax, x,plot_color,sT=30, alphaT =1):
  #  plt.subplot(grid[int(np.floor(i/3)), (i % 3) ], projection='3d')
    ax.scatter(x[:,0],x[:,1],x[:,2],s = sT, color = plot_color, alpha=alphaT)


# plot data
def plot_data_final(x,plot_color, grid):
    plt.subplot(grid[0,3])
    plt.scatter(x[:,0],x[:,1],s = 30, color = plot_color)
```

```
# plot data 3D
def plot_data_final_3D(ax, x,plot_color):
    #plt.subplot(grid[0,3], projection='3d')
    ax.scatter(x[:,0],x[:,1],x[:,2], color = plot_color, s = 5)
```

## 3 Fitting a polynomial feature model with varying degrees

While the codebase includes many options for features (eg. fourier features), as well as activation functions (sigmoid, softmax), we choose here the polynomial basis functions and choose the degree that is optimal for this data set.

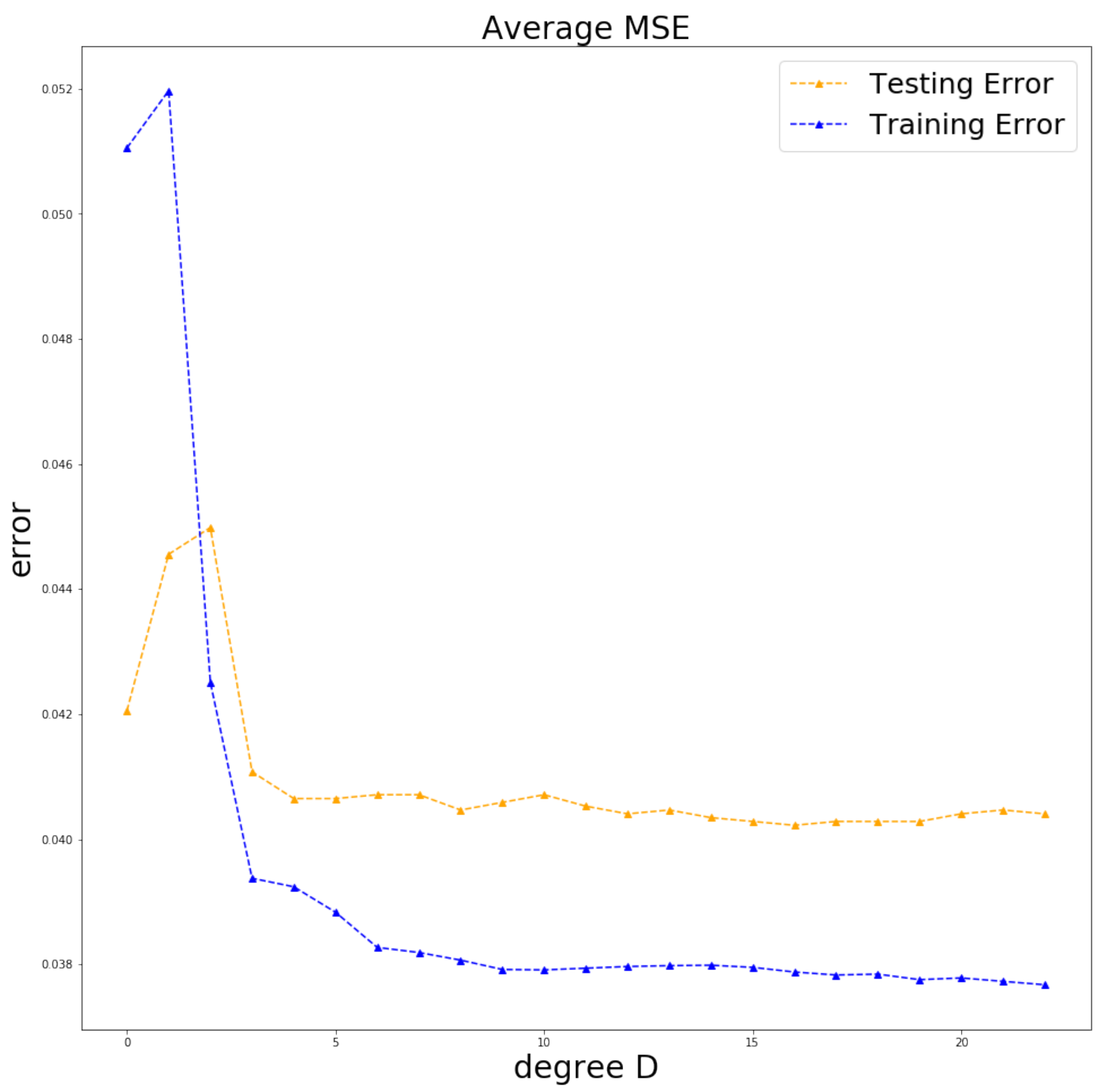
[5]:
```
# run over all the degrees, fit each models, and calculate errors
def hold_out_validation(x, y, k, deg_range, is_train):
    nP = x.shape[0]
    is_in = y == 1
    is_out = np.logical_not(is_in)


    indices = partition_indices_better(x,y,k)

    color_test = 'orange'
    color_train = 'blue'
    fig = plt.figure(figsize = (16,16))#20))
    ax = fig.add_subplot(111)

    mses_test = np.zeros((deg_range.size,1))
    mses_train = np.zeros((deg_range.size,1))


    is_test = np.logical_not(is_train)

    yTrain = y[is_train]
    yTest  = y[is_test]

    for D in np.arange(0,np.size(deg_range)):
        print(D/np.size(deg_range))
        # First we find the optimal number of features for this training set

        F_train = poly_features_multiDim(x[is_train,:], deg_range[D])
        F_test  = poly_features_multiDim(x[is_test,:], deg_range[D])

        temp = np.linalg.pinv(np.dot(F_train, F_train.T))
        w = np.dot(np.dot(temp, F_train),yTrain)

        trainMSEtemp = np.sign(-yTrain * np.dot(F_train.T, w))
        trainMSEtemp[trainMSEtemp < 0] = 0
        testMSEtemp = np.sign(-yTest * np.dot(F_test.T, w))
        testMSEtemp[testMSEtemp < 0] = 0

        mses_train[D] = np.sum(trainMSEtemp)/trainMSEtemp.shape[0]
        mses_test[D] = np.sum(testMSEtemp)/testMSEtemp.shape[0]

    # choose best case number of features and plot the final model
    deg_final = deg_range[np.argmin(mses_test)]
    F_train = poly_features_multiDim(x[is_train,:], deg_final)
    F_test  = poly_features_multiDim(x[is_test,:], deg_final)
    F = poly_features_multiDim(x, deg_final)
    temp = np.linalg.pinv(np.dot(F_train, F_train.T))
    w_Final = np.dot(np.dot(temp, F_train), yTrain)

    plot_mse(ax, mses_test, deg_range, color_test, 'Testing Error','','')
    plot_mse(ax, mses_train, deg_range, color_train, 'Training Error', 'Average␣
↪MSE','degree D')
    plt.legend(fontsize=25)

    mse_all_final = np.sign(-y * np.dot(F.T, w_Final))
    mse_all_final[mse_all_final < 0] = 0



    return  mse_all_final
```

## 4 Load data, concatenate and compute optimal number of polynomial degrees

Data at this point has been downsampled and chosen to ensure a fairly balanced set.

[8]:
```
# load data and defined degree range
import random
x, y = load_data('data_dn4_3d\DATA_W1_downsampled.csv')
nImg = 1;

imgNum = np.arange(2,11);
random.shuffle(imgNum)

imgNums = np.ones((y.shape));

for i in imgNum:#range(2,11):#5):#11):
    nImg += 1
    xT, yT = load_data('data_dn4_3d\DATA_W' + str(i) + '_downsampled.csv')
    imgNums = np.concatenate((imgNums, i * np.ones((yT.shape))),axis=0);
    x = np.concatenate((x,xT),axis=0);
    y =  np.concatenate((y,yT),axis=0)


for i in range(x.shape[1]):
    x[:,i] = (x[:,i] - np.min(x[:,i]))/np.ptp(x[:,i])

is_train = np.zeros(x.shape[0], dtype=bool)
is_train[0:-np.int(x.shape[0]/nImg)] = True
is_test = np.logical_not(is_train)

y[y==0] = -1;

# hold out validation
k = 10

# degrees to test
deg_range = np.arange(0,23)   # for 10

final_est = hold_out_validation(x,y,k,deg_range, is_train)
```

```
0.0
0.043478260869565216
0.08695652173913043
0.13043478260869565
0.17391304347826086
0.21739130434782608
0.2608695652173913
0.30434782608695654
0.34782608695652173
0.391304347826087
0.43478260869565216
0.4782608695652174
0.5217391304347826
0.5652173913043478
0.6086956521739131
0.6521739130434783
0.6956521739130435
0.7391304347826086
0.782608695652174
0.8260869565217391
0.8695652173913043
0.9130434782608695
0.9565217391304348
```

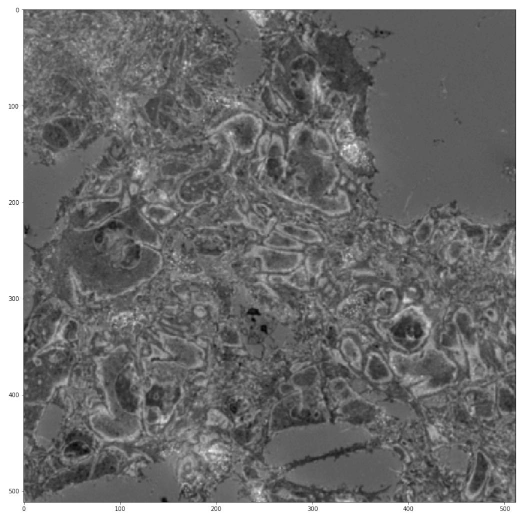


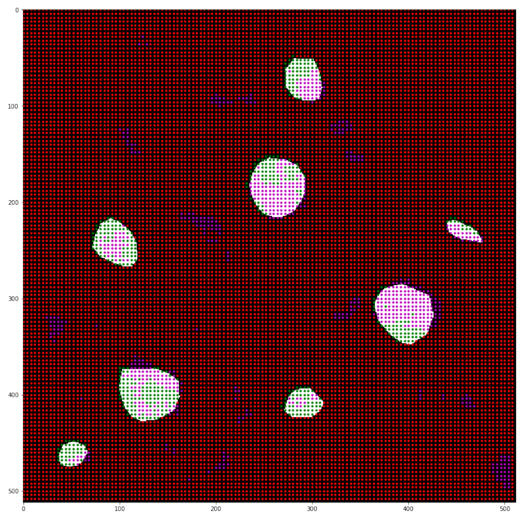
## 5 Build model, compute metrics and visualize results

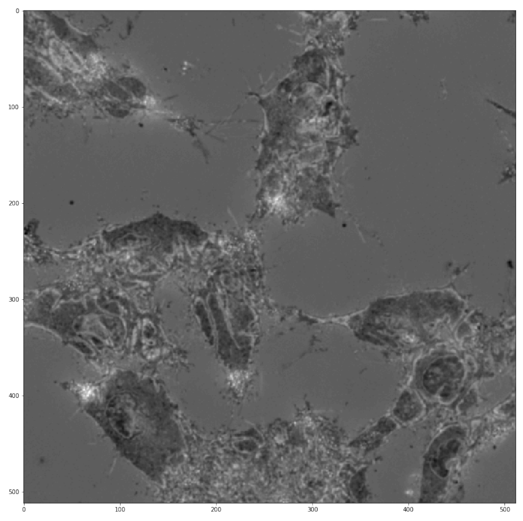
[19]:
```python
#from IPython.display import Image
#from IPython.core.display import HTML
import matplotlib.image as mpimg
PATH = "data_dn4_3d/"



sx, sy = np.meshgrid(np.arange(0,512,4),np.arange(0,512,4))
sx = sx.flatten()
sy = sy.flatten()
```

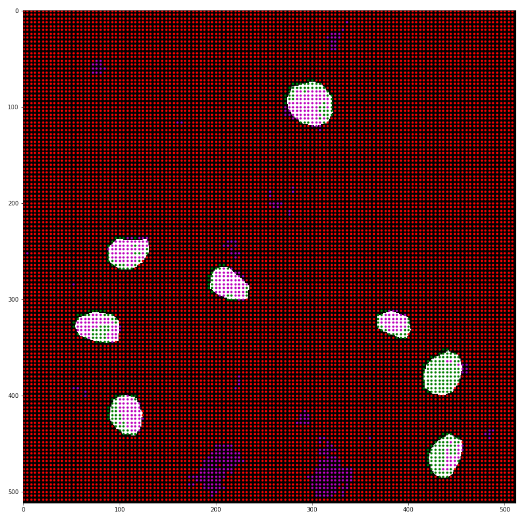
```
cellTrainToView = 6;
img1 = mpimg.imread(PATH + "Cell" + str(cellTrainToView) +"_imbd.png")
img2 = mpimg.imread(PATH + "Cell" + str(cellTrainToView) +"_bw.png")

_, yTrained = load_data('data_dn4_3d\DATA_W' + str(cellTrainToView) +␣
 ↪'_downsampled.csv')
yTrained_Model = final_est[imgNums==cellTrainToView]
yTrained = yTrained ==1;  yTrained = yTrained.reshape((-1,))
yTrained_Model = yTrained_Model == 1;

TP1, TN1, FP1, FN1 = get_classification_res(yTrained, yTrained_Model)


img3 = mpimg.imread(PATH + "Cell" + str(imgNum[-1]) + "_imbd.png")
img4 = mpimg.imread(PATH + "Cell" + str(imgNum[-1]) + "_bw.png")
_, yTested = load_data('data_dn4_3d\DATA_W' + str(imgNum[-1]) + '_downsampled.
 ↪csv')
yTested_Model = final_est[imgNums==imgNum[-1]]
yTested = yTested ==1;  yTested = yTested.reshape((-1,))
yTested_Model = yTested_Model == 1;




plt.figure(figsize = (34,34))
plt.subplot(221)
plt.imshow(img1);

plt.subplot(222)
plt.imshow(img2)
plt.plot( sy[TP1],sx[TP1],'g.')
plt.plot( sy[TN1],sx[TN1],'r.')
plt.plot( sy[FP1],sx[FP1],'m.')
plt.plot( sy[FN1],sx[FN1],'m.')


plt.subplot(223)
plt.imshow(img3);

plt.subplot(224)
plt.imshow(img4)
plt.plot(sy[TP2],sx[TP2], 'g.')
plt.plot(sy[TN2],sx[TN2], 'r.')
plt.plot( sy[FP2],sx[FP2],'m.')
plt.plot( sy[FN2],sx[FN2],'m.')

plt.show()
```

```
TP2, TN2, FP2, FN2 = get_classification_res(yTested, yTested_Model)
print('TP-test',np.sum(TP2))
print('TN-test',np.sum(TN2))
print('FP-test',np.sum(FP2))
print('FN-test',np.sum(FN2))
print('TP-test',np.sum(TP2)/TP2.shape[0])
print('TN-test',np.sum(TN2)/TP2.shape[0])
print('FP-test',np.sum(FP2)/TP2.shape[0])
print('FN-test',np.sum(FN2)/TP2.shape[0])
```









```
TP-test 361
TN-test 15397
FP-test 298
FN-test 328
TP-test 0.02203369140625
TN-test 0.93975830078125
FP-test 0.0181884765625
FN-test 0.02001953125
```