# California State University, Fullerton


**Data Mining Project #1: Regression & Classification**

Joseph Porter
Noe Rojas
Christian Chavez
Zhi Hong
Kevin Ochoa

**Data Set**

For our first data set we used publicly available research on Facebook performance metrics. The data is from 2014 and contains 500 instances. Each instance contains 19 attributes. 7 of the features are based on information prior to post. The other 12 attributes were for post impact. The 19 attributes included are:

Page total likes;Type;Category;Post Month;Post Weekday;Post Hour;Paid;Lifetime Post Total Reach;Lifetime Post Total Impressions;Lifetime Engaged Users;Lifetime Post Consumers;Lifetime Post Consumptions;Lifetime Post Impressions by people who have liked your Page;Lifetime Post reach by people who like your Page;Lifetime People who have liked your Page and engaged with your post;comment;like;share;Total Interactions.

(Moro et al., 2016) S. Moro, P. Rita and B. Vala. Predicting social media performance metrics and evaluation of the impact on brand building: A data mining approach. Journal of Business Research, Elsevier, In press.

**Task**

Our task was to use the Facebook performance metrics to predict the total amount of interactions for a post using regression.

**Preprocessing**

```
def preprocessFacebook(fileIn, fileOut):
    Photo = '0'
    Status = '1'
    Link = '2'
    Video = '3'
    with open(fileIn, 'r') as fin:
        outF = open(fileOut, 'w')
        for data in fin:
            if data.find(';;') == -1:
                if data.find('Photo') > -1:
                    outF.write(data.replace('Photo', Photo).replace('\r', ''))
                elif data.find('Status') > -1:
                    outF.write(data.replace('Status', Status).replace('\r', ''))
                elif data.find('Link') > -1:
                    outF.write(data.replace('Link', Link).replace('\r', ''))
                elif data.find('Video') > -1:
                    outF.write(data.replace('Video', Video).replace('\r', ''))

        outF.close()
```

In order to use the data set we need to clean up the information. The Facebook dataset used entries that were not numeric and had attributes that were empty. In the preprocessing step, we replaced the second row entry labeled Photo, Status, Link, or Video with the values 0-3. Where photo::0, status::1, link::2, and video::3. The entries that contained empty values were removed.

**Normalization**

```
preprocessFacebook('dataset_Facebook.csv', 'facebook.csv')
data = np.loadtxt('facebook.csv', delimiter = ';')
data = data-data.mean(axis = 0)
imax = np.concatenate((data.max(axis=0)*np.ones((1,19)),np.abs(data.min(axis=0)*np.ones((1,19)))),axis=0).max(axis=0)
data = data/imax
```

Next, we normalized the inputs using the maximum. The second column, which contained the post type, was not normalized.

**Training, Validation, Test Sets**

```
train = data[::2,:]
trainTarget = data[::2,18].reshape((np.shape(train)[0]),1)
valid = data[1::4,:]
validTarget = data[1::4,18].reshape((np.shape(valid)[0]),1)
test = data[3::4,:]
testTarget = data[3::4,18].reshape((np.shape(test)[0]),1)
```

The training, validation and, test sets are split  50-25-25. We did not have to randomize the data since it did not come with any predefined pattern.

**Multi-Layered Perceptron Design**

```
net = mlp.mlp(train, trainTarget, 30, outtype = 'linear')
net.earlystopping(train, trainTarget, valid, validTarget, 0.4, 500)

test = np.concatenate((test,-np.ones((np.shape(test)[0],1))),axis=1)
testout = net.mlpfwd(test)
```

```
pl.figure()
pl.plot(np.arange(np.shape(test)[0]),testout,'.')
pl.plot(np.arange(np.shape(test)[0]),testTarget,'x')
pl.legend(('Predictions','Targets'))
pl.show()
```

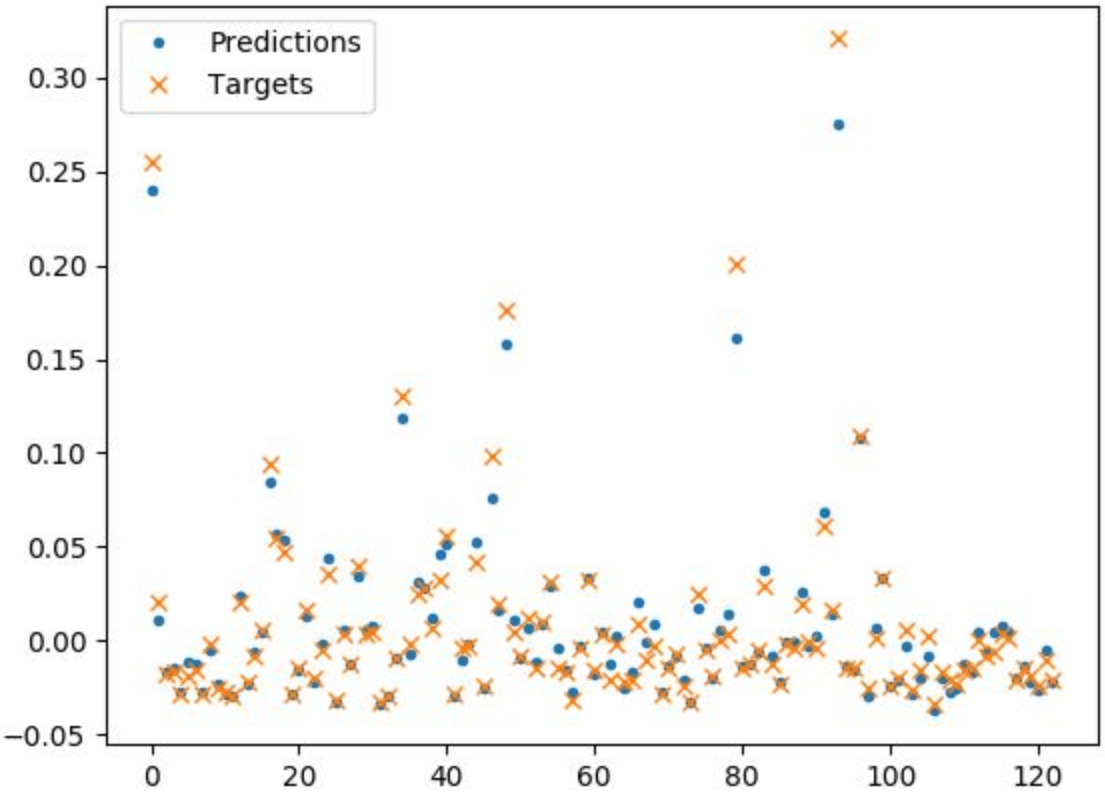
The design for our MLP contains 30 nodes in the hidden layer. Since we are performing regression we use a linear output type. Our learning rate was set to 0.4 for 500 iterations. We generated a graph of our predictions in relation to our targets.

**Results**

Console Output showing the error for each iteration and the the 3 errors at the time it stopped

```
1
Iteration:  0  Error:  18.9701514659
2
Iteration:  0  Error:  0.00581873886823
3
Iteration:  0  Error:  0.00331312003154
4
Iteration:  0  Error:  0.00261037117019
Stopped 0.00238053406667 0.00281945977451 0.00344300671786
```

This graph shows our predictions in relation to our targets. X-axis is the instances and the Y-axis is the normalized interactions per post.



**Data Mining Project #1: Classification**

**Data Set**

For our second data set we used wine recognition data. The data is the chemical analysis of wine grown in the same location in Italy but 3 different cultivars/plant types/classes. There are 178 instances. Each instance contains 13 attributes. Attributes included are:

Alcohol; Malic acid; Ash; Alcalinity of ash; Magnesium; Total phenols; Flavanoids; Nonflavanoid phenols; Proanthocyanins; Color intensity; Hue; OD280/OD315 of diluted wines; Proline

S. Aeberhard, D. Coomans and O. de Vel,"THE CLASSIFICATION PERFORMANCE OF RDA" Tech. Rep. no. 92-01, (1992), Dept. of Computer Science and Dept. of Mathematics and Statistics, James Cook University of North Queensland. (Also submitted to Journal of Chemometrics).

**Task**

Our task was to use the wine recognition data to classify determine the cultivar of wine using classification.

**Normalization**

```
10  #normolizing
11  wine[:,1:] = wine[:,1:]-wine[:,1:].mean(axis=0)
12  imax = np.concatenate((wine.max(axis=0)*np.ones((1,14)),np.abs(wine.min(axis=0)*np.ones((1,14)))),axis=0).max(axis=0)
13  wine[:,1:] = wine[:,1:]/imax[1:]
14
```

Next, we normalized the inputs using the maximum. Unlike our regression example, every column in our dataset was normalized.

**Convert to 1-of-N Encoding**

```
14
15  target = np.zeros((np.shape(wine)[0],3));
16  indices = np.where(wine[:,0]==1)
17  target[indices,0] = 1
18  indices = np.where(wine[:,0]==2)
19  target[indices,1] = 1
20  indices = np.where(wine[:,0]==3)
21  target[indices,2] = 1
22
23  # Randomly order the data
```

We change it to 1-of-N encodings so that we can perform softmax classification.

**Randomize the Order of the Input**

```
23  order = range(np.shape(wine)[0])
24  np.random.shuffle(order)
25  wine = wine[order,:]
26  target = target[order,:]
```

Since our data come predefined within its classes, we have to randomize the data to prevent overfitting for a certain class.

**Training, Test, Validation**

```
29  train = wine[::2,1:]
30  traint = target[::2]
31  valid = wine[1::4,1:]
32  validt = target[1::4]
33  test = wine[3::4,1:]
34  testt = target[3::4]
35
```

The training, validation and, test sets are split  50-25-25.

**Multi-Layered Perceptron Design**

```
36
37  net = mlp.mlp(train,traint,13,outtype='softmax')
38  net.earlystopping(train,traint,valid,validt,0.1)
39  net.confmat(test,testt)
40
```

The design for our MLP contains 13 nodes in the hidden layer. Since we are performing classification we use a softmax output type. Our learning rate was set to 0.1.

**Results**

```
Confusion matrix is:
[[ 14.   1.   0.]
 [  0.  16.   0.]
 [  0.   0.  13.]]
Percentage Correct:  97.7272727273
```

**Slides Link**

https://docs.google.com/presentation/d/1QpcDccw803EFcr7sNP1194BNWSWhbuiwq0t1b7BxvbM/edit?usp=sharing