In [1]:

```
#  Ebnable HTML/CSS
from IPython.core.display import HTML
HTML("<link href='https://fonts.googleapis.com/css?family=Passion+One' rel='stylesheet' type='text/css'><style>div.attn { font-family: 'Helvetica Neue'; font-size: 30px; line-height: 40px; color: #FFFFFF; text-align: center; marg
```

Out[1]:

Enter Team Member Names here (double click to edit):

- Name 1: Ben Goodwin
- Name 2: Andre Mauldin

In Class Assignment One

In the following assignment you will be asked to fill in python code and derivations for a number of different problems. Please read all instructions carefully and turn in the rendered notebook (or HTML of the rendered notebook) before the end of class (or right after class). The initial portion of this notebook is given before class and the remainder is given during class. Please answer the initial questions before class. Once class has started you may rework your answers as a team for the initial part of the assignment.

Contents

Back to Top

Loading the Data

Please run the following code to read in the "diabetes" dataset from sklearn's data loading module.

This will load the data into the variable `ds`. `ds` is a dictionary object with fields like `ds.data`, which is a matrix of the continuous features in the dataset. The object is not a pandas dataframe. It is a numpy matrix. Each row is a set of observed instances, each column is a different feature. It also has a field called `ds.target` that is a continuous value we are trying to predict. Each entry in `ds.target` is a label for each row of the `ds.data` matrix.

In [181…

```
from sklearn.datasets import load_diabetes
import numpy as np
from __future__ import print_function


ds = load_diabetes()

# this holds the continuous feature data
# because ds.data is a matrix, there are some special properties we can access (like 'shape')
print('features shape:', ds.data.shape, 'format is:', ('rows','columns')) # there are 442 instances and 10 features per instance
print('range of target:', np.min(ds.target),np.max(ds.target))
```

```
features shape: (442, 10) format is: ('rows', 'columns')
range of target: 25.0 346.0
```

In [182…

```
from pprint import pprint

# we can set the fields inside of ds and set them to new variables in python
pprint(ds.data) # prints out elements of the matrix
pprint(ds.target) # prints the vector (all 442 items)
```

```
array([[ 0.03807591,  0.05068012,  0.06169621, ..., -0.00259226,
         0.01990842, -0.01764613],
       [-0.00188202, -0.04464164, -0.05147406, ..., -0.03949338,
        -0.06832974, -0.09220405],
       [ 0.08529891,  0.05068012,  0.04445121, ..., -0.00259226,
         0.00286377, -0.02593034],
       ...,
       [ 0.04170844,  0.05068012, -0.01590626, ..., -0.01107952,
        -0.04687948,  0.01549073],
       [-0.04547248, -0.04464164,  0.03906215, ...,  0.02655962,
         0.04452837, -0.02593034],
       [-0.04547248, -0.04464164, -0.0730303 , ..., -0.03949338,
        -0.00421986,  0.00306441]])
array([151.,  75., 141., 206., 135.,  97., 138.,  63., 110., 310., 101.,
        69., 179., 185., 118., 171., 166., 144.,  97., 168.,  68.,  49.,
        68., 245., 184., 202., 137.,  85., 131., 283., 129.,  59., 341.,
        87.,  65., 102., 265., 276., 252.,  90., 100.,  55.,  61.,  92.,
       259.,  53., 190., 142.,  75., 142., 155., 225.,  59., 104., 182.,
       128.,  52.,  37., 170., 170.,  61., 144.,  52., 128.,  71., 163.,
       150.,  97., 160., 178.,  48., 270., 202., 111.,  85.,  42., 170.,
       200., 252., 113., 143.,  51.,  52., 210.,  65., 141.,  55., 134.,
        42., 111.,  98., 164.,  48.,  96.,  90., 162., 150., 279.,  92.,
        83., 128., 102., 302., 198.,  95.,  53., 134., 144., 232.,  81.,
       104.,  59., 246., 297., 258., 229., 275., 281., 179., 200., 200.,
       173., 180.,  84., 121., 161.,  99., 109., 115., 268., 274., 158.,
       107.,  83., 103., 272.,  85., 280., 336., 281., 118., 317., 235.,
        60., 174., 259., 178., 128.,  96., 126., 288.,  88., 292.,  71.,
       197., 186.,  25.,  84.,  96., 195.,  53., 217., 172., 131., 214.,
        59.,  70., 220., 268., 152.,  47.,  74., 295., 101., 151., 127.,
       237., 225.,  81., 151., 107.,  64., 138., 185., 265., 101., 137.,
       143., 141.,  79., 292., 178.,  91., 116.,  86., 122.,  72., 129.,
       142.,  90., 158.,  39., 196., 222., 277.,  99., 196., 202., 155.,
        77., 191.,  70.,  73.,  49.,  65., 263., 248., 296., 214., 185.,
        78.,  93., 252., 150.,  77., 208.,  77., 108., 160.,  53., 220.,
       154., 259.,  90., 246., 124.,  67.,  72., 257., 262., 275., 177.,
        71.,  47., 187., 125.,  78.,  51., 258., 215., 303., 243.,  91.,
       150., 310., 153., 346.,  63.,  89.,  50.,  39., 103., 308., 116.,
       145.,  74.,  45., 115., 264.,  87., 202., 127., 182., 241.,  66.,
        94., 283.,  64., 102., 200., 265.,  94., 230., 181., 156., 233.,
        60., 219.,  80.,  68., 332., 248.,  84., 200.,  55.,  85.,  89.,
        31., 129.,  83., 275.,  65., 198., 236., 253., 124.,  44., 172.,
       114., 142., 109., 180., 144., 163., 147.,  97., 220., 190., 109.,
       191., 122., 230., 242., 248., 249., 192., 131., 237.,  78., 135.,
       244., 199., 270., 164.,  72.,  96., 306.,  91., 214.,  95., 216.,
       263., 178., 113., 200., 139., 139.,  88., 148.,  88., 243.,  71.,
        77., 109., 272.,  60.,  54., 221.,  90., 311., 281., 182., 321.,
        58., 262., 206., 233., 242., 123., 167.,  63., 197.,  71., 168.,
       140., 217., 121., 235., 245.,  40.,  52., 104., 132.,  88.,  69.,
       219.,  72., 201., 110.,  51., 277.,  63., 118.,  69., 273., 258.,
        43., 198., 242., 232., 175.,  93., 168., 275., 293., 281.,  72.,
       140., 189., 181., 209., 136., 261., 113., 131., 174., 257.,  55.,
        84.,  42., 146., 212., 233.,  91., 111., 152., 120.,  67., 310.,
        94., 183.,  66., 173.,  72.,  49.,  64.,  48., 178., 104., 132.,
       220.,  57.])
```

Back to Top

Using Linear Regression

In the videos, we derived the formula for calculating the optimal values of the regression weights (you must be connected to the internet for this equation to show up properly):

$$w = (X^T X)^{-1} X^T y$$

where X is the matrix of values with a bias column of ones appended onto it. For the diabetes dataset one could construct this X matrix by stacking a column of ones onto the `ds.data` matrix.

$$X = \begin{bmatrix} & \vdots & & 1 \\ \cdots & \text{ds.data} & \cdots & \vdots \\ & \vdots & & 1 \end{bmatrix}$$

Question 1: For the diabetes dataset, how many elements will the vector w contain?

In [4]:

```
# Enter your answer here (or write code to calculate it)

# Enter your answer here (or write code to calculate it)

#We have 10 predictors, and will append 1 bias column
#Answer: 11

#
```

The answer is: 11 elements are in w

Exercise 1: In the following empty cell, use this equation and numpy matrix operations to find the values of the vector w. You will need to be sure X and y are created like the instructor talked about in the video. Don't forget to include any modifications to X to account for the bias term in w. You might be interested in the following functions:

- `np.hstack((mat1,mat2))` stack two matrices horizontally, to create a new matrix
- `np.ones((rows,cols))` create a matrix full of ones
- `my_mat.T` takes transpose of numpy matrix named `my_mat`
- `np.dot(mat1,mat2)` is matrix multiplication for two matrices
- `np.linalg.inv(mat)` gets the inverse of the variable `mat`

In [183…

```
# Write you code here, print the values of the regression weights using the 'print()' function in python
# Write you code here, print the values of the regression weights using the 'print()' function in python
rows = ds.data.shape[0]             #<- grab all of the data for X
ones_col = np.ones([rows,1])        #<- create a ones column for the bias term
X = np.hstack( (ones_col, ds.data) )  #<- stack the ones column onto ds.data using np.haystack(), litterally appends the ones column to X (hstack == stack horizontally)

y = ds.target


w = np.dot( np.dot( np.linalg.inv( np.dot(X.T, X) ) , X.T ) , y)
print(w)
```

```
[ 152.13348416  -10.01219782 -239.81908937  519.83978679  324.39042769
 -792.18416163  476.74583782  101.04457032  177.06417623  751.27932109
   67.62538639]
```

Back to Top

Start of Live Session Coding

Exercise 2: Scikit-learn also has a linear regression fitting implementation. Look at the scikit learn API and learn to use the linear regression method. The API is here:

- API Reference: http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

Use the sklearn `LinearRegression` module to check your results from the previous question.

Question 2: Did you get the same parameters?

In [184…

```
from sklearn.linear_model import LinearRegression

# write your code here, print the values of model by accessing
# its properties that you looked up from the API

regressor = LinearRegression()
regressor.fit(ds.data, ds.target)
intercept = regressor.intercept_
regressors= regressor.coef_


print('model coefficients are:', regressors)
print('model intercept is', intercept)
print('Answer to question is', 'Yes, we got the same parameters')
```

```
model coefficients are: [ -10.01219782 -239.81908937  519.83978679  324.39042769 -792.18416163
  476.74583782  101.04457032  177.06417623  751.27932109   67.62538639]
model intercept is 152.1334841628965
Answer to question is Yes, we got the same parameters
```

Recall that to predict the output from our model, $\hat{y}$, from w and X we need to use the following formula:

- $\hat{y} = w^T X^T$

Where X is a matrix with example instances in *each row* of the matrix.

Exercise 3:

- *Part A:* Use matrix multiplication to predict output using numpy, $\hat{y}_{numpy}$ and also using the sklearn regression object, $\hat{y}_{sklearn}$.
  - **Note**: you may need to make the regression weights a column vector using the following code: `w = w.reshape((len(w),1))` This assumes your weights vector is assigned to the variable named `w`.
- *Part B:* Calculate the mean squared error between your prediction from numpy and the target, $\sum_i (y - \hat{y}_{numpy})^2$.
- *Part C:* Calculate the mean squared error between your sklearn prediction and the target, $\sum_i (y - \hat{y}_{sklearn})^2$.

In [185…

```
# Use this block to answer the questions
from sklearn.metrics import mean_squared_error


##########Sklearm part###########
#Do predictions with sklearn
y_pred_sklearn = regressor.predict(ds.data)

#Print results
#print(y_pred_sklearn)

#MSE Calculations
mseSkLearn=(mean_squared_error(ds.target, y_pred_sklearn))



##########Numpy part###########
#Do predictions with numpy
w = w.reshape((len(w),1)) # make w a column vector
y_pred_numpy=np.dot(w.T,X.T)

#Print Results
#print(y_pred_numpy)

#MSE Calculations
mse = ((y_pred_numpy -ds.target )**2).mean(axis=1)
#print(mse)


##########Results#############
print('MSE Sklearn is:', mseSkLearn)
print('MSE Numpy is:', mse)
```

```
MSE Sklearn is: 2859.6903987680657
MSE Numpy is: [2859.69039877]
```

Back to Top

Using Linear Classification

Now lets use the code you created to make a classifier with linear boundaries. Run the following code in order to load the iris dataset.

In [186…

```
from sklearn.datasets import load_iris
import numpy as np

# this will overwrite the diabetes dataset
ds = load_iris()
print('features shape:', ds.data.shape) # there are 150 instances and 4 features per instance
print('original number of classes:', len(np.unique(ds.target)))

# now let's make this a binary classification task
ds.target = ds.target>1
print ('new number of classes:', len(np.unique(ds.target)))
```

```
features shape: (150, 4)
original number of classes: 3
new number of classes: 2
```

Exercise 4: Now use linear regression to come up with a set of weights, `w`, that predict the class value. This is exactly like you did before for the *diabetes* dataset. However, instead of regressing to continuous values, you are just regressing to the integer value of the class (0 or 1), like we talked about in the video. Remember to account for the bias term when constructing the feature matrix, `X`. Print the weights of the linear classifier.

In [187…

```
# write your code here and print the values of the weights
rows = ds.data.shape[0]             #<- grab all of the data for X
ones_col = np.ones([rows,1])        #<- create a ones column for the bias term
X = np.hstack( (ones_col, ds.data) )  #<- stack the ones column onto ds.data using np.haystack(), litterally appends the ones column to X (hstack == stack horizontally)

y = ds.target


w = np.dot( np.dot( np.linalg.inv( np.dot(X.T, X) ) , X.T ) , y)
print(w)

#Check again
regressor = LinearRegression()
regressor.fit(ds.data, ds.target)
#print(regressor.intercept_)
#print(regressor.coef_)

#Do predictions with sklearn
#y_pred_sklearn = regressor.predict(ds.data)
#print(y_pred_sklearn)
```

```
[-0.69528186 -0.04587608  0.20276839  0.00398791  0.55177932]
```

Out[187…

```
LinearRegression()
```

Exercise 5: Finally, use a hard decision function on the output of the linear regression to make this a binary classifier. This is just like we talked about in the video, where the output of the linear regression passes through a function:

- $\hat{y} = g(w^T X^T)$ where
  - $g(w^T X^T)$ for $w^T X^T < \alpha$ maps the predicted class to `0`
  - $g(w^T X^T)$ for $w^T X^T \geq \alpha$ maps the predicted class to `1`.

Here, alpha is a threshold for deciding the class.

Question 3: What value for α makes the most sense? What is the accuracy of the classifier given the α you chose?

Question 3 Answer I think that 0.5 is a good decision function. And based on this level of α we have a 92.66% accuracy.

Note: You can calculate the accuracy with the following code: `accuracy = float(sum(yhat==y)) / len(y)` assuming you choose variable names `y` and `yhat` for the target and prediction, respectively.

In [180…

```
# use this box to predict the classification output
y_pred_sklearn= np.dot(w.T,X.T)

y_pred_sklearn=y_pred_sklearn>0.5


#Test situation
#b =  y_pred_sklearn > 0.5
#print(b)
#print(y_pred_sklearn)

accuracy = float(sum(y_pred_sklearn==ds.target)) / len(ds.target)

print('Percentage accuracy:', accuracy*100)
```

```
Percentage accuracy: 92.66666666666666
```

That's all! Please **save (make sure you saved!!!) and upload your rendered notebook** and please include **team member names** in the notebook submission.

In [ ]: