In [1]:
```
"""
Landon Buell
Marek Petrik
CS 750.01
11 Feb 2020
"""

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Problem 1 [25%]

Suppose that collected data for a group of machine learning students from last year. For each student,I have a feature:

X1 = hours studied for the class every week, X2 = overall GPA Y = whether the student receives an A.

We fit a logistic regression model and produce estimated coefficients

$\hat{\beta}_0 = -6$, $\hat{\beta}_1 = -0.1$ and $\hat{\beta}_2 = 1.0$

1. Estimate the probability of getting an A for a student who studies for 40h and has an undergrad GPA of 2.0

In [2]:
```
b0,b1,b2 = -6.0,-0.1,+1.0
X1,X2 = 40,2.0

def logistic_regression (beta0,beta1,beta2,X1,X2):
    """ Compute  output given input & polynomial coefficients """
    exp = np.exp(beta0 + (beta1*X1) + (beta2*X2))
    return exp/(1+exp)

probA = logistic_regression(b0,b1,b2,X1,X2)
print("Probability of getting an A:",probA*100,"% chance")
```

```
Probability of getting an A: 0.03353501304664781 % chance
```

2. By how much would the student in part 1 need to improve their GPA or adjust time studied to have a 90% chance of getting an A in the class? Is that likely?

In [3]:
```
probA = logistic_regression(b0,b1,b2,X1,12.2)
print("Probability of getting an A:",probA*100,"% chance")
print("A student would have to improve their GPS from a 2.0 to a ~12.2 to ha
```

```
Probability of getting an A: 90.02495108803149 % chance
A student would have to improve their GPS from a 2.0 to a ~12.2 to have a 9
0% chance of getting an A
```

Problem 2 [25%]

Consider a classification problem with two classes T (true) and F (false). Then, suppose that you have the following four prediction models:

•T: The classifier predicts T for each instance (always)

• F: The classifier predicts F for each instance (always)

• C: The classifier predicts the correct label always (100% accuracy)

• W: The classifier predicts the wrong label always (0% accuracy)

You also have a test set with 60% instances labeled T and 40%instances labeled F. Now, compute the following statistics for each one of your algorithms:

In [4]:
```
import sklearn.metrics as metrics

# create fake data sets
targets = np.array([1,1,1,1,1,1,0,0,0,0])
Cls_T = np.ones((1,10),dtype=int).transpose()
Cls_F = np.zeros((1,10),dtype=int).transpose()
Cls_C = np.copy(targets)
Cls_W = np.array([0,0,0,0,0,0,1,1,1,1])

# Compute vals & assemble into frame
models = [Cls_T,Cls_F,Cls_C,Cls_W]
```

In [5]:
```
# Compute recall scores w/ sklearn
recalls = np.array([metrics.recall_score(targets,X) for X in models])
truepos = np.array([metrics.confusion_matrix(targets,X)[1][1] for X in model
falsepos = np.array([metrics.confusion_matrix(targets,X)[0][1] for X in mode
trueneg = np.array([metrics.confusion_matrix(targets,X)[0][0] for X in model
specificity = np.array([metrics.confusion_matrix(targets,X)[1][0] for X in m
precisions = np.array([metrics.precision_score(targets,X) for X in models])

data = np.array([recalls,truepos,falsepos,trueneg,specificity,precisions])
frame = pd.DataFrame(data=data,columns=['Cls. T','Cls. F','Cls. C','Cls. W']
frame
```

```
C:\Users\Landon\Anaconda3\lib\site-packages\sklearn\metrics\classification.
py:1143: UndefinedMetricWarning: Precision is ill-defined and being set to
0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
```

Out[5]:

	Cls. T	Cls. F	Cls. C	Cls. W
0	1	0	1	0
1	6	0	6	0
2	4	0	0	4
3	0	4	4	0
4	0	6	0	6
5	0	0	1	0

Problem 3 [25%]

In this problem, you will derive the bias-variance decomposition of MSE as described in Eq. (2.7) in ISL. Let f be the true model, $\hat{f}$ be the estimated model. Consider fixed instance x_0 with the label $y_0 = f(x_0)$. For simplicity, assume that Var[ϵ]=0, in which case the decomposition becomes:

$$E\left[(y_0 - \hat{f}(x_0))^2\right] = Var\left[\hat{f}(x_0)\right] + \left(E\left[f(x_0) - \hat{f}(x_0)\right]\right)^2$$

Prove that the equality holds

We start by expanding the variance out:

$$E\left[(y_0 - \hat{f}(x_0))^2\right] = E\left[\hat{f}(x_0)^2\right] + E\left[\hat{f}(x_0)\right]^2 + \left(E\left[y_0 - \hat{f}(x_0)\right]\right)^2$$

Given the linearity of the expecation value operator, we can also expand out the left side of the equation such that we know read:

$$E\left[y^2\right] - 2E\left[y\hat{f}(x_0)\right] + E\left[\hat{f}(x_0)^2\right] = E\left[\hat{f}(x_0)^2\right] + E\left[\hat{f}(x_0)\right]^2 + \left(E\left[y_0 - \hat{f}(x_0)\right]\right)^2$$

Remove redundant terms from both sides:

$$E\left[y^2\right] - 2E\left[y\hat{f}(x_0)\right] = E\left[\hat{f}(x_0)\right]^2 + \left(E\left[y_0 - \hat{f}(x_0)\right]\right)^2$$

We can expand out the Bias^2 term to read:

$$\left(E\left[y_0 - \hat{f}(x_0)\right]\right)^2 = E\left[y_0 = \hat{f}(x_0)\right]E\left[y_0 = \hat{f}(x_0)\right] = E\left[y_0\right]^2 - 2E\left[y_0\right]E\left[\hat{f}(x_0)\right]$$

Which can be inserted back into the equation from the previous cell:

$$E\left[y_0^2\right] - 2E\left[y\hat{f}(x_0)\right] = E\left[\hat{f}(x_0)\right]^2 + E\left[y_0\right]^2 - 2E\left[y_0\right]E\left[\hat{f}(x_0)\right]$$

Finally, cancelling terms can be removed, and the remaining terms are:

$$-2E\left[y\hat{f}(x_0)\right] = -2E\left[y_0\right]E\left[\hat{f}(x_0)\right]$$

Which is true for independent variables y_0 and $\hat{f}(x_0)$. Thus the MSE is equal to the sum of the variance and the Bias- squared.

Problem 4 [25%]

I wrote the following code that computes the MSE, bias, and variance for a test point.

The problem lies in the 2nd to last line of the program. The sample variance of a dataset is given by:

$$\sigma^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

However, as per the R documentation pages, R computes the variance of a datatset as:

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

Which is actaully the population variance of a data set and produces a different value. So, the relationship between MSE, bias, and variance no longer holds as computed by R.