Grading

The final score that you will receive for your programming assignment is generated in relation to the total points set in your programming assignment item—not the total point value in the nbgrader notebook. When calculating the final score shown to learners, the programming assignment takes the percentage of earned points vs. the total points provided by nbgrader and returns a score matching the equivalent percentage of the point value for the programming assignment.
DO NOT CHANGE VARIABLE OR METHOD SIGNATURES The autograder will not work properly if your change the variable or method signatures.

WARNING

Please refrain from using **print statements/anything that dumps large outputs(>500 lines) to STDOUT** to avoid running to into **memory issues**. Doing so requires your entire lab to be reset which may also result in loss of progress and you will be required to reach out to Coursera for assistance with this. This process usually takes time causing delays to your submission.

Validate Button

Please note that this assignment uses nbgrader to facilitate grading. You will see a **validate button** at the top of your Jupyter notebook. If you hit this button, it will run tests cases for the lab that aren't hidden. It is good to use the validate button before submitting the lab. Do know that the labs in the course contain hidden test cases. The validate button will not let you know whether these test cases pass. After submitting your lab, you can see more information about these hidden test cases in the Grader Output.
Cells with longer execution times will cause the validate button to time out and freeze. Please know that if you run into Validate time-outs, it will not affect the final submission grading.

Module 3: Logistic Regression

In [1]:
```
# importing all the required libraries

from math import exp
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
```

Binary classification with logistic regression.

Part A [5 points] : Your first task is to complete the function `gen_logistic` in the following cell so as to be able to generate the logistic function for a given input. The logistic function is a type of *sigmoid* function which has an 'S'-shape and 'squashes' its inputs to a value lying in the range [0,1]. Other sigmoid functions include the hyperbolic-tangent funcition (`tanh(x)`) and the error function (`erf(x)`). https://en.wikipedia.org/wiki/Sigmoid_function (https://en.wikipedia.org/wiki/Sigmoid_function). Calculate sigmoid using the below formulas.

In [2]:
```
def gen_logistic(x, w=1, b=0):
    """
    outputting the logistic output for an input x
    :param x: scalar or numpy array of shape (n_samples, n_features). If
only one feature, it must have the shape of (n_samples,1).
    :param w: weight(s); either scalar or numpy array of shape (1, n_feat
ures)
    :param b: bias; either scalar or numpy array of shape (1,)
    returns y of shape (n_samples,)
    """
    # TODO: Finish this function to return the output of applying the sig
moid
    # function to the input x (Please do not use external libraries) stor
e
    # the output in y and return y. Do not change the default parameter v
alues.
    # Hint: This function will be used in any input shape scalar (0d), 1d
vector, and 2d arrays. Please make sure it can handle all those. Followin
g reshaping codes might help.
    # Hint2: You may use design matrix using concatenation, but it is not
necesary.

    # Ensure x, w, and b are numpy arrays
    x = np.array(x)
    w = np.array(w)
    b = np.array(b)

    # Reshape x, w, and b appropriately
    if x.ndim == 0:
        x = x.reshape((1, 1))
    elif x.ndim == 1:
        x = x.reshape((-1, 1))

    if w.ndim == 0:
        w = w.reshape((1,))
    elif w.ndim == 1:
        w = w.reshape((1, -1))

    if b.ndim == 0:
        b = b.reshape((1,))

    # Calculate the logistic function
    z = np.dot(x, w.T) + b
    y = 1 / (1 + np.exp(-z))

    return y.ravel()
```

In [3]:
```
# Example usage:
print(gen_logistic(0))           # Should output 0.5
print(gen_logistic(np.array([0, 1, 2])))  # Should output array with logi
stic values
print(gen_logistic(np.array([[0], [1], [2]])))  # Should output array wit
h logistic values
```

```
[0.5]
[0.5        0.73105858 0.88079708]
[0.5        0.73105858 0.88079708]
```

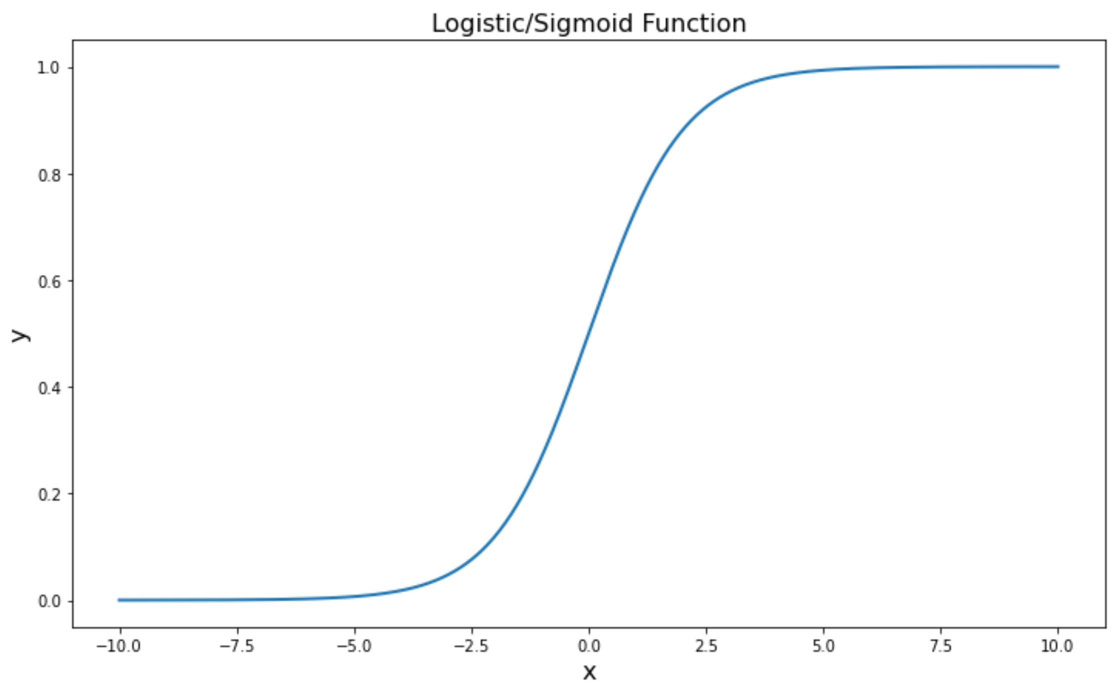
In [4]:
```
# Sample tests that gen_logistic function returns the output of applying the sigmoid function to the input x
# ouput is stored and returned in y
import pytest
assert pytest.approx(gen_logistic(np.array([[2],[0.2],[17]])), 0.001) == np.array([0.88079708, 0.549834, 0.99999996]), "Check the gen_logistic function."
```

In [5]:
```
# tests that gen_logistic function returns the output of applying the sigmoid function to the input x
# ouput is stored and returned in y
```

Part B [5 points, Peer Review]: Generate a vector x of length N with values lying between limits Xa and Xb (for this you will have to choose your own limits; play around with different values) and apply the `gen_logistic` function to this vector. Proceed to plot the output and verify the shape of the output. If your decision boundary value is about the center of your x range, you will see an S-shape. Complete the Peer Review section for this section.

In [6]:
```
# your code here

# TODO: change the values of N, a and b below to check how the output of
your function works
# Use a value for N greater than 1 and any limits a and b so that an S-sh
ape graph is generated

N = 1000  # Using a larger value for N to generate a smooth curve
Xa = -10  # Lower limit
Xb = 10   # Upper limit
w = 1     # Weight
b = 0     # Bias

x = np.expand_dims(np.linspace(Xa,Xb,N), axis=1)
y = gen_logistic(x, w, b)

fig, ax = plt.subplots(nrows=1,ncols=1,figsize=(12,7))
ax.plot(x,y, lw=2)
ax.set_xlabel("x", fontsize=16)
ax.set_ylabel("y", fontsize=16)
ax.set_title("Logistic/Sigmoid Function", fontsize=16)
```

Out[6]:
```
Text(0.5, 1.0, 'Logistic/Sigmoid Function')
```



1. Increasing w will make the curve transition sharply: [2 pts, True/False]

In [7]:
```
# uncomment sharp_transition and answer qustion 1. above
# replace string with 'True' or 'False'
# your code here

sharp_transition = 'True'
```

In [8]:
```
# this cell tests sharp_transition
```

2. If b increases by 1, then the decision boundary x decreases by 1: [3 pts, True/False]

In [9]:
```
# uncomment x_decreases_by_1 and answer question 2. above
# replace string with 'True' or 'False'
# your code here

x_decreases_by_1 = 'False'
```

In [10]:
```
# this cell tests x_decreases_by_1
```

PART C [10 pts, Peer Review]: Performing binary classification using logistic regression on the breast-cancer dataset. In this part you will be exposed to different methods within the scikit-learn LogisticRegression class so you can build a classifier.

Import breast cancer dataset from sklearn [5 pts]

In [11]:
```
# Importing the breast-cancer dataset from sklearn datasets

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score, roc_curve
import matplotlib.pyplot as plt

class BC_data:
    """
    class to import the breast cancer dataset from sklearn
    
    """
    
    def __init__(self):
        x, y = load_breast_cancer(return_X_y=True)
        self.x_train, self.x_test, self.y_train, self.y_test = train_test
_split(x, y, test_size=0.25, random_state=5)
        
        # TODO: Split the data into training and test data (use train_tes
t_split sklearn)
        # such that the test data size is 25% of total number of observat
ions
        # No need to rescale the data. Use the data as is.
        # Use random_state = 5
        
        # your code here
        
        
        
data = BC_data()
```

In [12]:
```
print(f"Training data shape: {data.x_train.shape}")
print(f"Test data shape: {data.x_test.shape}")
```

```
Training data shape: (426, 30)
Test data shape: (143, 30)
```

In [13]:
```
# tests that you properly split data into training and test data
# such that test dat size is 25% of the total number of observations
```

Build and Fit Logistic Regression Model [5 pts]

In [14]:
```
# TODO: Use the data object and then train the logistic regression model.
# 1. Change the code below to build the model called LogReg.
# Use the Logistic Regression function from Sklearn library
# and set up the logistic regression with the 'liblinear' solver.
# 2. Fit the model to the train data

LogReg = LogisticRegression(solver='liblinear')
LogReg.fit(data.x_train, data.y_train)

# your code here
```

Out[14]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=T
rue,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='liblinear', tol=0.0001, ve
rbose=0,
                   warm_start=False)
```

In [15]:
```
# weights
LogReg.coef_
```

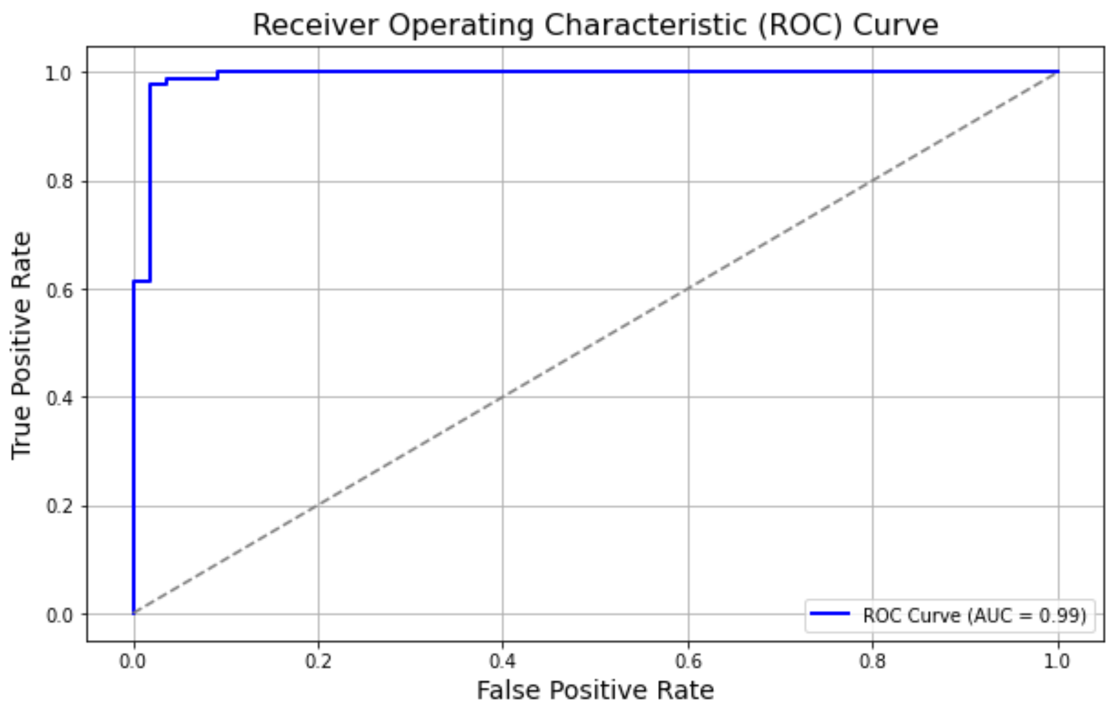
Out[15]:
```
array([[ 1.88231944e+00,  1.47142473e-01, -9.71531390e-02,
        -1.13599910e-03, -1.34134760e-01, -3.29140819e-01,
        -5.73217280e-01, -3.13930239e-01, -1.81409248e-01,
        -1.23090682e-02,  4.07858457e-02,  1.61999469e+00,
         1.21420271e-01, -9.83114672e-02, -1.74653538e-02,
         2.55209963e-02, -4.96490253e-02, -3.72305279e-02,
        -2.75600158e-02,  8.31209925e-03,  1.51867881e+00,
        -4.04910392e-01, -1.07027831e-01, -2.51574494e-02,
        -2.54528667e-01, -8.27922366e-01, -1.37276059e+00,
        -5.77660119e-01, -5.64539086e-01, -8.27950664e-02]])
```

In [16]:
```
# tests LogReg model
```

In the next cell, compute the ROC curve and the area under the curve and plot the ROC curve. Upload a copy or screenshot of the plot for this week's **Peer Review assignment**.

Hint: sklearn.metrics has a function to calculate area under the curve.

In [17]:
```
# TODO: compute the area under the curve and plot ROC curve
# Plot the ROC curve ( True positive rate v/s False positive rate) and in
dicate the AUC on the plot

# your code here
# Compute the ROC curve and the area under the curve (AUC)
y_pred_proba = LogReg.predict_proba(data.x_test)[:, 1]
fpr, tpr, thresholds = roc_curve(data.y_test, y_pred_proba)
auc = roc_auc_score(data.y_test, y_pred_proba)

# Plot the ROC curve
plt.figure(figsize=(10, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC Curve (AUC = {auc:.2
f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('False Positive Rate', fontsize=14)
plt.ylabel('True Positive Rate', fontsize=14)
plt.title('Receiver Operating Characteristic (ROC) Curve', fontsize=16)
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```



Part D [5 pts, Peer Review]: Here we will use the trained model coefficients and generate the `classification probabilities` using the `gen_logistic` function we built. The goal of this section is to make you understand how logistic regression classifies data points during and after training. Using the predictions from the generated probabilities, you will compute the precision and recall metrics (defined below).

In [18]:
```
def calculate_precision(y_true, y_pred, pos_label_value=1.0):
    '''
    This function accepts the labels and the predictions, then
    calculates precision for a binary classifier.

    Args
        y_true: np.ndarray
        y_pred: np.ndarray

        pos_label_value: (float) the number which represents the postiive
        label in the y_true and y_pred arrays. Other numbers will be take
n
        to be the non-positive class for the binary classifier.

    Returns precision as a floating point number between 0.0 and 1.0
    '''


    # your code here
    true_positives = np.sum((y_true == pos_label_value) & (y_pred == pos_
label_value))
    predicted_positives = np.sum(y_pred == pos_label_value)
    precision = true_positives / predicted_positives if predicted_positiv
es > 0 else 0.0
    return precision



def calculate_recall(y_true, y_pred, pos_label_value=1.0):
    '''
    This function accepts the labels and the predictions, then
    calculates recall for a binary classifier.

    Args
        y_true: np.ndarray
        y_pred: np.ndarray

        pos_label_value: (float) the number which represents the postiive
        label in the y_true and y_pred arrays. Other numbers will be take
n
        to be the non-positive class for the binary classifier.

    Returns precision as a floating point number between 0.0 and 1.0
    '''

    # your code here


    true_positives = np.sum((y_true == pos_label_value) & (y_pred == pos_
label_value))
    actual_positives = np.sum(y_true == pos_label_value)
    recall = true_positives / actual_positives if actual_positives > 0 el
se 0.0
    return recall
```

In [ ]:
```
```

```
In [19]: # Sample Test cell
         ut_true = np.array([1.0, 1.0, 0.0, 1.0, 1.0, 0.0])
         ut_pred = np.array([1.0, 1.0, 1.0, 1.0, 0.0, 1.0])
         prec = calculate_precision(ut_true, ut_pred, 1.0)
         recall = calculate_recall(ut_true, ut_pred, 1.0)
         assert prec == 0.6, "Check the precision value returned from your calcula
         te_precision function."
         assert recall == 0.75, "Check the recall value returned from your calcula
         te_recall function."
```

```
In [20]: print(prec)

         0.6
```

```
In [21]: print(recall)

         0.75
```

```
In [22]: # testing cell
```

In the next cell you will generate the predictions for the test data `data.x_test` and compute prediction and recall metrics by calling the functions you built above. Take a screenshot of your code to submit for your **Peer Review assignment**. Make sure that you use the ***gen_logistic function***.

In [23]:

```python
# TO-DO : Generate predicted y values using coefficients of the fit logis
tic regression model for data.x_test
# Then compute and print the precision and recall metrics


# Generate classification probabilities using the gen_logistic function
# Extract model coefficients and intercept
w = LogReg.coef_.flatten()
b = LogReg.intercept_

# Generate probabilities for the test set
y_prob = gen_logistic(data.x_test, w, b)
y_pred = (y_prob >= 0.5).astype(int)



precision = calculate_precision(data.y_test, y_pred)
recall = calculate_recall(data.y_test, y_pred)



print('Model Precision : %0.2f' % precision)
print('Model Recall : %0.2f' % recall)

print(y_pred)
```

```
Model Precision : 0.98
Model Recall : 0.99
[0 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1 0 0 0 0 0 1 1
0
 1 1 0 1 1 1 0 1 1 0 0 1 0 1 1 1 1 1 0 0 0 1 0 0 0 1 1 1 1 1 1 1 0 1 0 1
1
 1 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0 1 1 0 0 1 1 1 1 1 0 1 1 0 0 0 0 1 0
1
 0 1 0 1 1 0 1 0 1 1 1 0 1 0 1 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1]
```

In [ ]:

```python

```