LAB5 Logistic Regression Part1

March 4, 2021

1 LAB5 Logistic Regression Part1

1.0.1 Part 1: Work on the python code for chapter 7 from "Python Machine Learning" by Wei-Meng Lee.

1.0.2 Import libraries

[1]:
```
import pandas as pd
import numpy as np
import matplotlib as plt
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from sklearn.datasets import load_breast_cancer
from sklearn import linear_model
from mpl_toolkits.mplot3d import Axes3D
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, auc
```

L = ln* $\frac{P}{1-P}$ The formula for the logit function

[2]:
```
%matplotlib inline
```

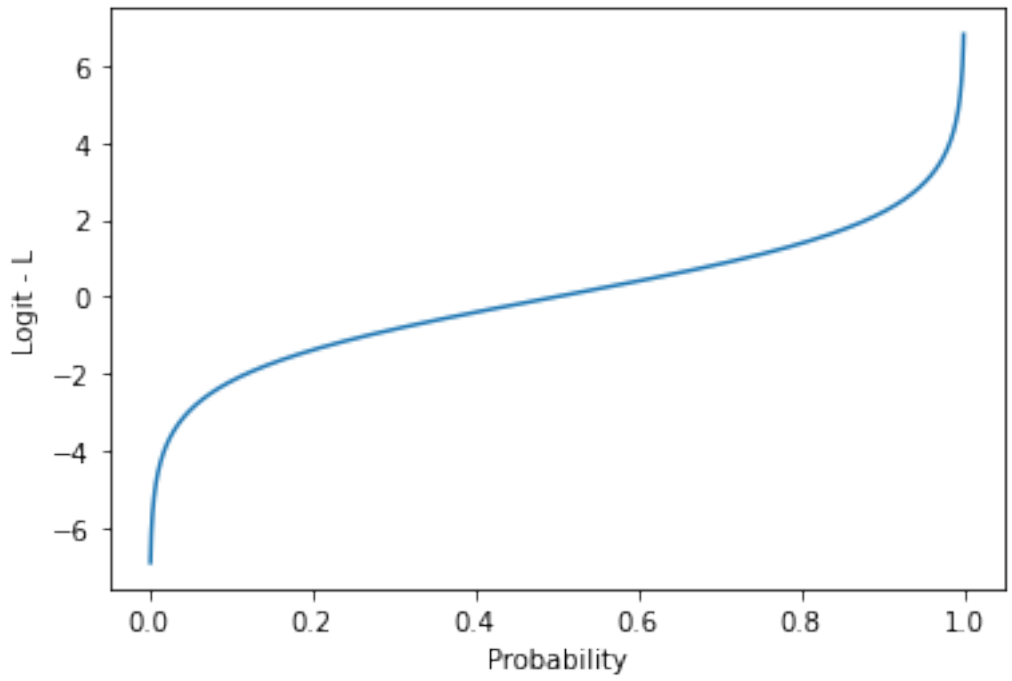
import %matplotlib only once

[3]:
```
# logit function
def logit(x):
    return np.log(x / (1-x))
```

1.0.3 Logit Curve

[4]:
```
x = np.arange(0.001, 0.999, 0.0001)
y = [logit(n) for n in x]  # put all x values into logit func.
plt.plot(x,y)
plt.xlabel("Probability")
plt.ylabel("Logit - L")
```

[4]:
```
Text(0, 0.5, 'Logit - L')
```



Shows the logit curve plotted using the preceding code snippet. 0

1.0.4 Sigmoid Curve

0 1 1

1.0.5 Inverse of Logit function. Flip the axes.

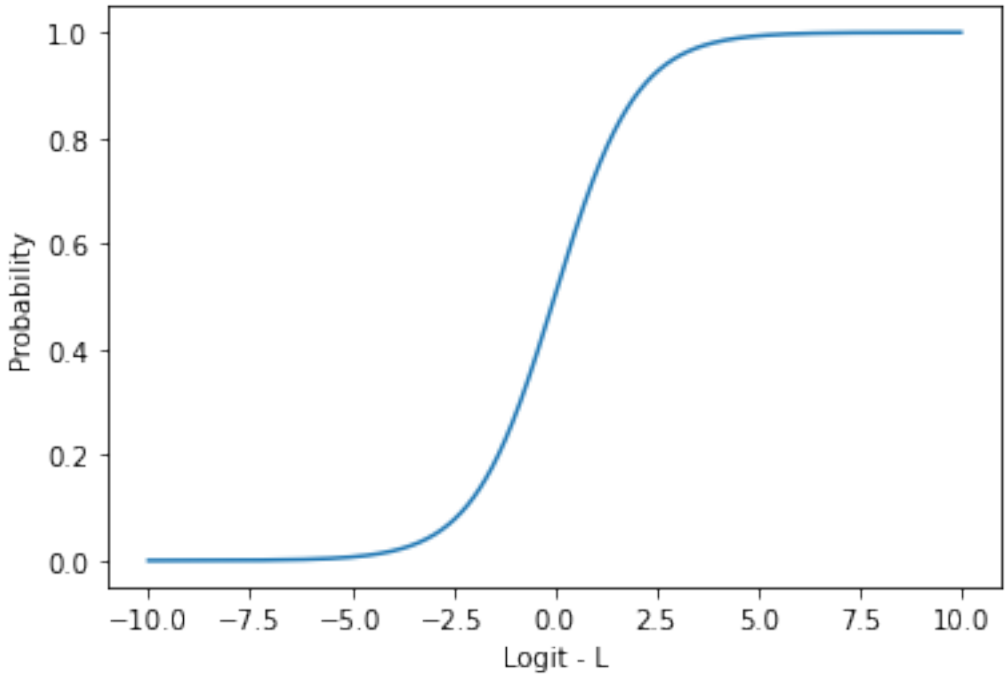
$P = \frac{1}{e^{-}(L)}$

1.0.6 Plot Sigmoid Curve

[5]:
```
def sigmoid(x):
    return (1/(1+np.exp(-x)))

x = np.arange(-10, 10, 0.0001)
y = [sigmoid(n) for n in x]
plt.plot(x,y)
plt.xlabel("Logit - L")
plt.ylabel("Probability")
```

[5]:
```
Text(0, 0.5, 'Probability')
```



Shows the plot of Sigmoid
We use sigmoid as an activation layer in neural networks

1.0.7 Read the datasets from database

[6]:
```
# from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
```

[7]:
```
#print(cancer)
```

[ ]:
```

```

[8]:
```
print(cancer.keys())
```

```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names',
'filename'])
```

1.0.8 How to convert this data files to dataframe?

def converts():

```
data = numpy.c_[cancer.data, cancer.target]
colums = numpy.append(cancer.feature_names,[target])
return pandas.DataFrame(data, colums=colums)
```

df = converts() assert frame.shape ==(len(cancer.tagert),31)

[ ]:

1.0.9 Plotting the Features in 2D

[9]:
```
# from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer()
```

[10]:
```
cancer
# 'data' means whole data
```

[10]:
```
{'data': array([[1.799e+01, 1.038e+01, 1.228e+02, …, 2.654e-01, 4.601e-01,
         1.189e-01],
        [2.057e+01, 1.777e+01, 1.329e+02, …, 1.860e-01, 2.750e-01,
         8.902e-02],
        [1.969e+01, 2.125e+01, 1.300e+02, …, 2.430e-01, 3.613e-01,
         8.758e-02],
        …,
        [1.660e+01, 2.808e+01, 1.083e+02, …, 1.418e-01, 2.218e-01,
         7.820e-02],
        [2.060e+01, 2.933e+01, 1.401e+02, …, 2.650e-01, 4.087e-01,
         1.240e-01],
        [7.760e+00, 2.454e+01, 4.792e+01, …, 0.000e+00, 2.871e-01,
         7.039e-02]]),
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
1,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
        1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
        1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
        1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
        0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
        1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
        0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
        1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
        0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
        1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
        1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
        1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
        1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
```

```
        1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
        1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1]),
 'frame': None,
 'target_names': array(['malignant', 'benign'], dtype='<U9'),
 'DESCR': '.. _breast_cancer_dataset:\n\nBreast cancer wisconsin (diagnostic)
dataset\n--------------------------------------------\n\n**Data Set
Characteristics:**\n\n    :Number of Instances: 569\n\n    :Number of
Attributes: 30 numeric, predictive attributes and the class\n\n    :Attribute
Information:\n        - radius (mean of distances from center to points on the
perimeter)\n        - texture (standard deviation of gray-scale values)\n
- perimeter\n        - area\n        - smoothness (local variation in radius
lengths)\n        - compactness (perimeter^2 / area - 1.0)\n        - concavity
(severity of concave portions of the contour)\n        - concave points (number
of concave portions of the contour)\n        - symmetry\n        - fractal
dimension ("coastline approximation" - 1)\n\n        The mean, standard error,
and "worst" or largest (mean of the three\n        worst/largest values) of
these features were computed for each image,\n        resulting in 30 features.
For instance, field 0 is Mean Radius, field\n        10 is Radius SE, field 20
is Worst Radius.\n\n        - class:\n                - WDBC-Malignant\n
- WDBC-Benign\n\n    :Summary Statistics:\n\n
===================================== ====== ======\n
Min    Max\n    ===================================== ====== ======\n    radius
(mean):                         6.981  28.11\n    texture (mean):
9.71   39.28\n    perimeter (mean):                     43.79  188.5\n    area
(mean):                           143.5  2501.0\n    smoothness (mean):
0.053  0.163\n    compactness (mean):                   0.019  0.345\n
concavity (mean):                     0.0    0.427\n    concave points (mean):
0.0    0.201\n    symmetry (mean):                       0.106  0.304\n
fractal dimension (mean):             0.05   0.097\n    radius (standard error):
0.112  2.873\n    texture (standard error):             0.36   4.885\n
perimeter (standard error):           0.757  21.98\n    area (standard error):
6.802  542.2\n    smoothness (standard error):          0.002  0.031\n
compactness (standard error):         0.002  0.135\n    concavity (standard
error):           0.0    0.396\n    concave points (standard error):      0.0
0.053\n    symmetry (standard error):            0.008  0.079\n    fractal
dimension (standard error):   0.001  0.03\n    radius (worst):
7.93   36.04\n    texture (worst):                      12.02  49.54\n
perimeter (worst):                    50.41  251.2\n    area (worst):
185.2  4254.0\n    smoothness (worst):                   0.071  0.223\n
compactness (worst):                  0.027  1.058\n    concavity (worst):
0.0    1.252\n    concave points (worst):               0.0    0.291\n
symmetry (worst):                     0.156  0.664\n    fractal dimension
(worst):            0.055  0.208\n    =====================================
====== ======\n\n    :Missing Attribute Values: None\n\n    :Class Distribution:
212 - Malignant, 357 - Benign\n\n    :Creator:  Dr. William H. Wolberg, W. Nick
```

```
Street, Olvi L. Mangasarian\n\n    :Donor: Nick Street\n\n    :Date: November,
1995\n\nThis is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic)
datasets.\nhttps://goo.gl/U2Uwz2\n\nFeatures are computed from a digitized image
of a fine needle\naspirate (FNA) of a breast mass.  They
describe\ncharacteristics of the cell nuclei present in the image.\n\nSeparating
plane described above was obtained using\nMultisurface Method-Tree (MSM-T) [K.
P. Bennett, "Decision Tree\nConstruction Via Linear Programming." Proceedings of
the 4th\nMidwest Artificial Intelligence and Cognitive Science Society,\npp.
97-101, 1992], a classification method which uses linear\nprogramming to
construct a decision tree.  Relevant features\nwere selected using an exhaustive
search in the space of 1-4\nfeatures and 1-3 separating planes.\n\nThe actual
linear program used to obtain the separating plane\nin the 3-dimensional space
is that described in:\n[K. P. Bennett and O. L. Mangasarian: "Robust
Linear\nProgramming Discrimination of Two Linearly Inseparable
Sets",\nOptimization Methods and Software 1, 1992, 23-34].\n\nThis database is
also available through the UW CS ftp server:\n\nftp ftp.cs.wisc.edu\ncd math-
prog/cpo-dataset/machine-learn/WDBC/\n\n.. topic:: References\n\n   - W.N.
Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction \n     for
breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on \n
Electronic Imaging: Science and Technology, volume 1905, pages 861-870,\n
San Jose, CA, 1993.\n   - O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast
cancer diagnosis and \n     prognosis via linear programming. Operations
Research, 43(4), pages 570-577, \n     July-August 1995.\n   - W.H. Wolberg,
W.N. Street, and O.L. Mangasarian. Machine learning techniques\n     to diagnose
breast cancer from fine-needle aspirates. Cancer Letters 77 (1994) \n
163-171.',
 'feature_names': array(['mean radius', 'mean texture', 'mean perimeter', 'mean
area',
        'mean smoothness', 'mean compactness', 'mean concavity',
        'mean concave points', 'mean symmetry', 'mean fractal dimension',
        'radius error', 'texture error', 'perimeter error', 'area error',
        'smoothness error', 'compactness error', 'concavity error',
        'concave points error', 'symmetry error',
        'fractal dimension error', 'worst radius', 'worst texture',
        'worst perimeter', 'worst area', 'worst smoothness',
        'worst compactness', 'worst concavity', 'worst concave points',
        'worst symmetry', 'worst fractal dimension'], dtype='<U23'),
 'filename': '/Users/hidenaka/opt/anaconda3/lib/python3.8/site-
packages/sklearn/datasets/data/breast_cancer.csv'}
```

[11]:
```
cancer.target
```

[11]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
```

```
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
       0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1])
```

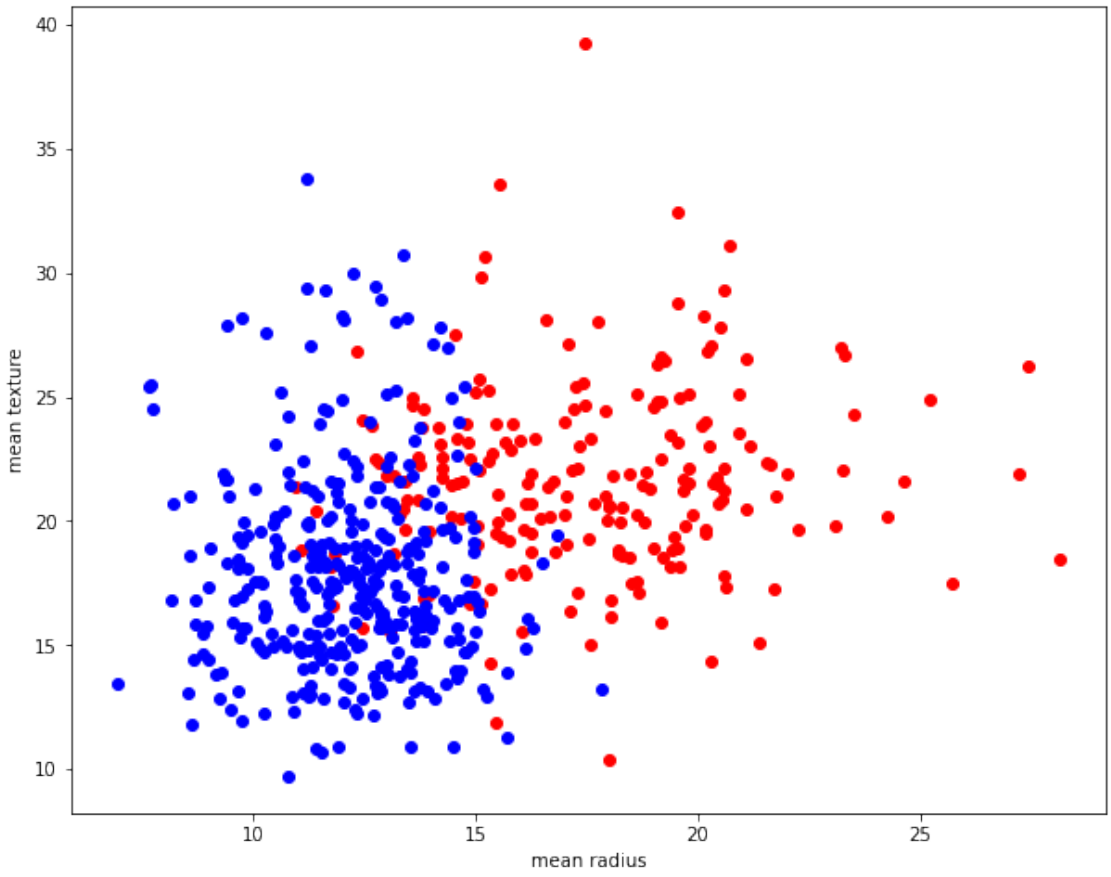
1.0.10 Copy from dataset into a 2D list

[12]:
```
x = []
for target in range(2):
    x.append([[], []])
    for i in range(len(cancer.data)):
        if cancer.target[i] == target:
            x[target][0].append(cancer.data[i][0])
            x[target][1].append(cancer.data[i][1])

colours = ("r", "b")  # r = malignant( ), b = benign( )
fig = plt.figure(figsize=(10,8))  #10,8 for default size
ax = fig.add_subplot(111)
for target in range(2):
    ax.scatter(x[target][0],
               x[target][1],
               c=colours[target])

ax.set_xlabel("mean radius")
ax.set_ylabel("mean texture")

plt.show() #shows the scatter plot of mean radius vs mean texture
```



[13]:
```
cancer.data
```

[13]:
```
array([[1.799e+01, 1.038e+01, 1.228e+02, …, 2.654e-01, 4.601e-01,
        1.189e-01],
       [2.057e+01, 1.777e+01, 1.329e+02, …, 1.860e-01, 2.750e-01,
        8.902e-02],
       [1.969e+01, 2.125e+01, 1.300e+02, …, 2.430e-01, 3.613e-01,
        8.758e-02],
       …,
       [1.660e+01, 2.808e+01, 1.083e+02, …, 1.418e-01, 2.218e-01,
        7.820e-02],
       [2.060e+01, 2.933e+01, 1.401e+02, …, 2.650e-01, 4.087e-01,
        1.240e-01],
       [7.760e+00, 2.454e+01, 4.792e+01, …, 0.000e+00, 2.871e-01,
        7.039e-02]])
```

1.0.11 Plotting in 3D

[14]:
```
# from mpl_toolkits.mplot3d import Axes3D
# from sklearn.datasets import load_breast_cancer
```

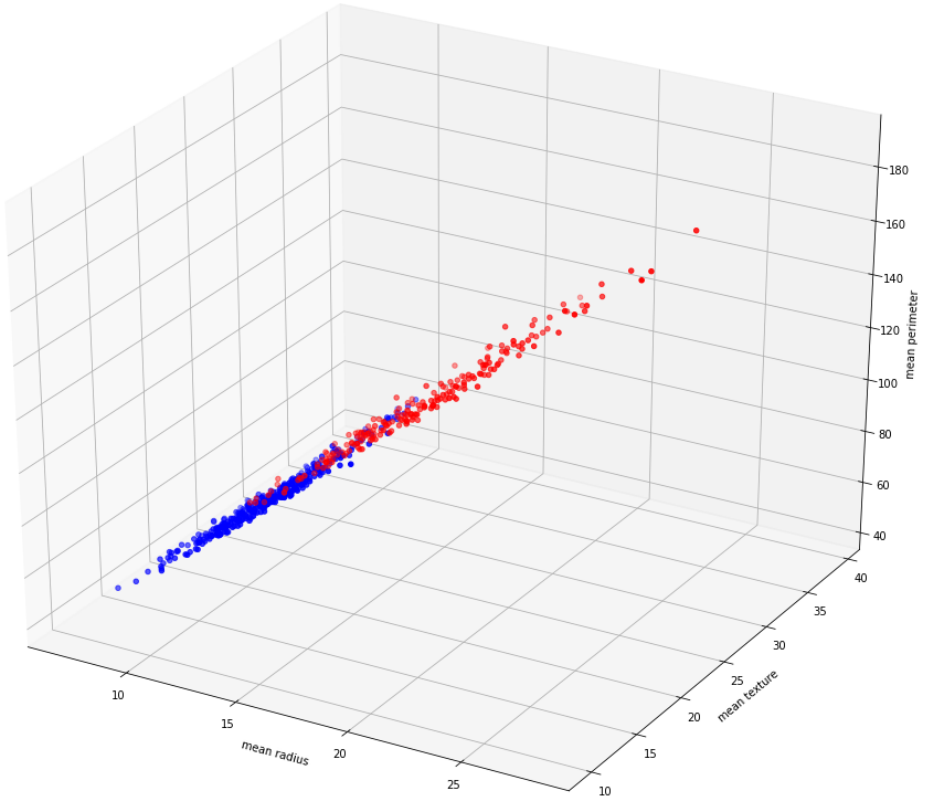
[15]:
```
cancer = load_breast_cancer()
```

[16]:
```
# x[1][0]
```

[17]:
```
x = []
for target in range(2):
    x.append([[], [], []])
    for i in range(len(cancer.data)):
        if cancer.target[i] == target:
            x[target][0].append(cancer.data[i][0])
            x[target][1].append(cancer.data[i][1])
            x[target][2].append(cancer.data[i][2])

colours = ("r", "b")
fig = plt.figure(figsize=(18,15))
ax = fig.add_subplot(111, projection='3d')
for target in range(2):
    ax.scatter(x[target][0],
               x[target][1],
               x[target][2],
               c = colours[target])

ax.set_xlabel("mean radius")
ax.set_ylabel("mean texture")
ax.set_zlabel("mean perimeter")
plt.show()
```



1.0.12 Training Using One Feature

Let's now use logistic regression to try to predict if a tumor is cancerous. To get started, let's use only the first feature of the dataset: mean radius. The following code snippet plots a scatter plot showing if a tumor is malignant or benign based on the mean radius:

Import libraries

[18]:
```
%matplotlib inline
# import matplotlib.patches as mpatches
```

Load data set

[19]:
```
# from sklearn.datasets import load_breast_cancer
```

[20]:
```
cancer = load_breast_cancer()     # Load dataset
x= cancer.data[:,0]               # mean radius
y=cancer.target                   # 0: malignant,1:benign
colors = {0:'red', 1:'blue'}      # 0: malignant,1:benign
```

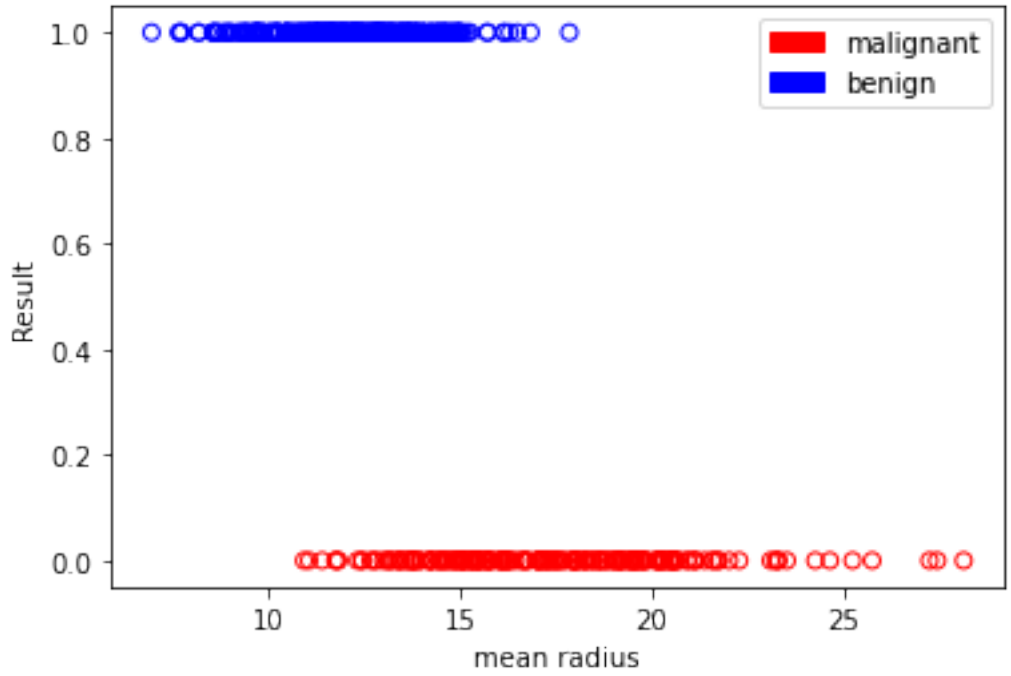
```
plt.scatter(x,y,
            facecolors ='none',
            edgecolors = pd.DataFrame(cancer.target)[0].apply(lambda x:
colors[x]),
            cmap = colors)

plt.xlabel("mean radius")
plt.ylabel("Result")

red = mpatches.Patch(color='red', label='malignant')
blue= mpatches.Patch(color='blue', label='benign')

plt.legend(handles=[red, blue], loc=1)
```

[20]: <matplotlib.legend.Legend at 0x7fa1b55d1f70>



Plotting a scatter plot based on one feature

[ ]:

1.0.13 Finding the Intercept and Coefficient

Scikit-learn comes with the LogisticRegression class that allows you to apply logistic regression to train model. Thus, in this example, you are going to train a model using the first featureof the dataset:

[21]:
```
# from sklearn import linear_model
# import numpy as np
```

[22]:
```
log_regress = linear_model.LogisticRegression()

# -- train the model --

log_regress.fit(X = np.array(x).reshape(len(x),1),  # first X is Cap letter,␣
 ↪second is small
                y = y)

# -- print trained model intercept --
print(log_regress.intercept_)

# --print trained model coefficients --
print(log_regress.coef_)
```

```
[15.120902]
[[-1.02475609]]
```

Once the model is trained, what we are most interested in at this point is the intercept and coefficient. The intercept is β_0 and the coefficient is xβ. Knowing these two values allows us to plot the sigmoid cirve that tries to fit the points on the chart

[ ]:
```

```

1.0.14 Plotting the Sigmoid Curve

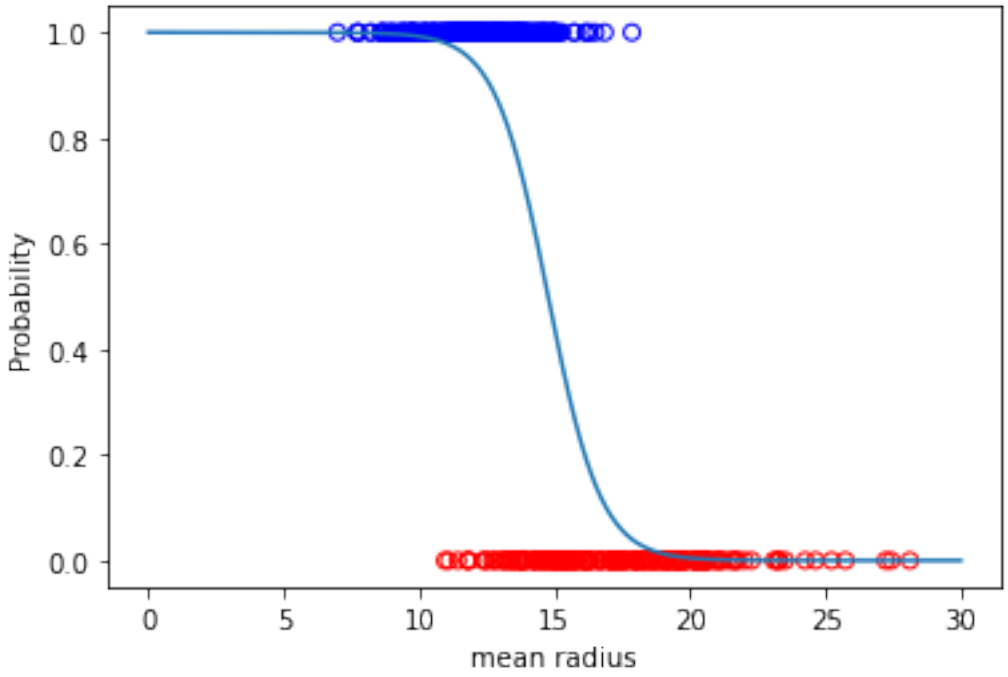
[23]:
```
def sigmoid(x):
    return(1/ (1+ np.exp(-(log_regress.intercept_[0] + (log_regress.coef_[0][0]␣
 ↪*x)))))

x1 = np.arange(0, 30, 0.01)
y1 = [sigmoid(n) for n in x1]

plt.scatter(x,y,
            facecolor='none',
            edgecolors=pd.DataFrame(cancer.target)[0].apply(lambda x:␣
 ↪colors[x]),
                                                                        ␣
 ↪cmap=colors)

plt.plot(x1,y1)
plt.xlabel("mean radius")
plt.ylabel("Probability")
```

[23]:
```
Text(0, 0.5, 'Probability')
```



The sigmoid curve fitting to the two sets of points

1.0.15 Making Predictions

Using the trained model, let's try to make some predictions. Let's try to predict the result if the mean radius is 20:

[24]:
```
print(log_regress.predict_proba([[20]]))
print(log_regress.predict([[20]])[0])
```

```
[[0.99538685 0.00461315]]
0
```

As you can see from the output, the predict_proba() function in the first statement returns a two-dimentional array. The result of 0.93489354 indicates the probability that the prediction is 0(malignant) while the result of 0.06510646 indicates the probability that prediction is 1. Based on the default threshold of 0.5, the prediction is that the tumor is galignant(value of 0),since its predicted probability(0.93489354) of 0 is more than 0.4. The predict() function in the second statement returns the class that the result lies in (which in this case can be a 0 or 1). The result of 0 indicates that the prediction is that the tumor is malignant.

Try another example with the mean radius of 8 this time:

[25]:
```
print(log_regress.predict_proba([[8]]))
print(log_regress.predict([[8]])[0])
```

```
[[9.84046071e-04 9.99015954e-01]]
1
```

[26]:
```
# double bracket for 2D array
```

1.0.16 Traning the Model Using All Features

[27]:
```
# from sklearn.datasets import load_breast_cancer
# cancer = load_breast_cancer()
```

Instead of training the model using all of the rows in the dataset, you are going to split it into two sets, one for training and one for testing. To do so, you use the train_test_split() function. This function allows you to split your data into random train and test subsets. The following code snippet splits the dataset into a 75 percent training and 25 percent testing set:

[28]:
```
# from sklearn.model_selection import train_test_split
train_set, test_set, train_labels, test_labels = train_test_split(
                            cancer.data,              #features
                            cancer.target,            # labels
                            test_size = 0.25,         # labels
                            random_state = 1,         # set random seed
                            stratify = cancer.target)  # randomize based on␣
 ↪labels
```

using " sklearn_semection import train_test_split " will give an error, so use other module

[29]:
```
# from sklearn import linear_model

x = train_set[:,0:30]     # mean radius
y = train_labels          # 0: malignant, 1: benign

log_regress = linear_model.LogisticRegression()
log_regress.fit(X = x,
                y = y)
```

```
/Users/hidenaka/opt/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:762: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[29]:
```
LogisticRegression()
```

[47]:
```
print(log_regress.intercept_)
print(log_regress.coef_)
```

```
[0.10275241]
[[ 0.58423081  0.52596032  0.51030382 -0.03114384 -0.01613523 -0.09086357
  -0.12877866 -0.0509992  -0.02960892 -0.00431452  0.02720266  0.13741625
   0.02784736 -0.11096659 -0.00142723 -0.02147716 -0.02940127 -0.00683337
  -0.00556348 -0.00205294  0.67652264 -0.57200189 -0.30713434 -0.00673701
  -0.03260984 -0.30745061 -0.38697836 -0.10427861 -0.08810453 -0.02873485]]
```

1.0.17 Testing the Model

[31]:
```
# get a predicted probabilities and convert into a dataframe
preds_prob = pd.DataFrame(log_regress.predict_proba(X=test_set))
```

[32]:
```
# assign column names to prediction
preds_prob.columns = ["Malignant", "Benign"]
```

[33]:
```
# get the predicted class labels
preds = log_regress.predict(X=test_set)
preds_class = pd.DataFrame(preds)
preds_class.columns = ["Prediction"]
```

[34]:
```
# actual diagnosis
original_result = pd.DataFrame(test_labels)
original_result.columns = ["Original Result"]
```

[35]:
```
# Merge the three dataframes into one
result = pd.concat([preds_prob, preds_class, original_result], axis =1)
print(result.head())
```

```
   Malignant        Benign  Prediction  Original Result
0   0.999838  1.624571e-04           0                0
1   0.999750  2.503299e-04           0                0
2   0.174584  8.254156e-01           1                1
3   1.000000  5.795674e-09           0                0
4   0.088078  9.119217e-01           1                0
```

The result of the predictions are then printed out.
The predictions and original diagnosis are displayed side by side for rasy comparison.

1.0.18 Getting the Confusion Matrix

[36]:
```
# generate table of predictions vs actual
print("--- confusion Matrix ---")
print(pd.crosstab(preds, test_labels))
```

```
--- confusion Matrix ---
col_0   0   1
row_0
0      48   4
1       5  86
```

(0 for malignant and 1 for benign)
- True Positive(TP): The model correctly predicts the outcome as positive. In this example, the number of TP(87) indicates the number of correct predictions that a tumor is benign. - True Negative(TN): The model correctly predicts the outcome as negative. In this example, tumors were correctly predicted to be malignant. - False Positibe(FP): The model is incorrectly predicted the outcome as positibe, but the actual result is negatibe. In this example, it means that the tumor is actually malignant, but the model predicted the tumor to be benign. - False Negative(FN): The model is incorrectly predicted the outcomes as negative, but the actual result is positive. In this example, it means that the tumor is actually, benign, but the model predicted the tumor to be malignant.

Besides using the crosstab() function, you can also use the confusion _matrix() function to print out confusion to print out the confusion matrix:

[37]:
```
# from sklearn import metrics
# --- view the confusion matrix --
print(metrics.confusion_matrix(y_true = test_labels,
                               y_pred = preds))
```

```
[[48  5]
 [ 4 86]]
```

1.0.19 Get The Accuracy of the Prediction

[38]:
```
#--- get the accuracy of the prediction ----
print("--- Accuracy ----")
print(log_regress.score(X = test_set,
                        y = test_labels))
```

```
--- Accuracy ----
0.9370629370629371
```

To get the precision, recall, and F1-score of the model, use the classification_report() function of the metrics module:

[39]:
```
# View summary of common classfication metrics
print("--- Metrices----")

print(metrics.classification_report(
      y_true = test_labels,
      y_pred = preds))
```

```
--- Metrices----
              precision    recall  f1-score   support
```

```
              0       0.92      0.91      0.91        53
              1       0.95      0.96      0.95        90

    accuracy                            0.94       143
   macro avg       0.93      0.93      0.93       143
weighted avg       0.94      0.94      0.94       143
```

1.0.20 Receiving Operating Characteristic (ROC) Curve

With so many metrics available, what is an easy way to examine the effectiveness of an algorithm? One way would be to plot a curve known as the Receiver Operating Characteristic(ROC) curve. The ROC curve is created by plotting the TPR against the FPR at various threshold settings.

Default threshold of 0.5 (meaning that all of those predicted probabilities less than or equal to 0.5 belong to one class, while those greater than 0.5 belong to another class).

[40]:
```
# from sklearn.metrics import roc_curve, auc
```

[41]:
```
# find the predicted probabilities using the test set

probs = log_regress.predict_proba(test_set)
preds = probs[:,1]

# find the FPR, TPR, and threshold
fpr, tpr, threshold  = roc_curve(test_labels, preds)
```

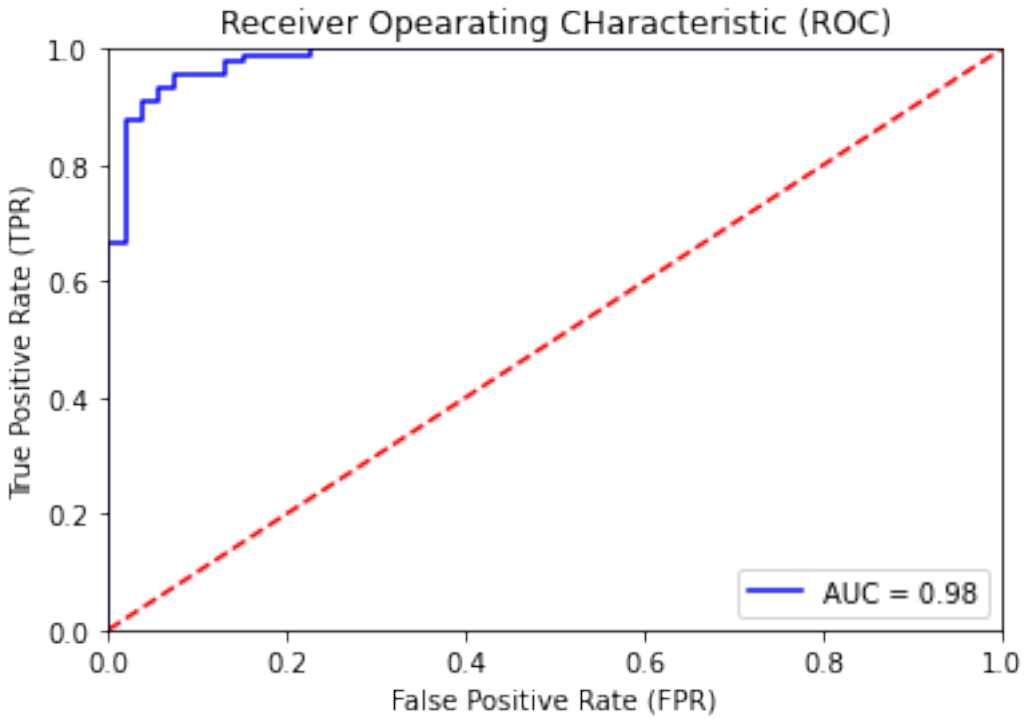
[42]:
```
print(fpr)
print(tpr)
print(threshold)
```

```
[0.         0.         0.         0.01886792 0.01886792 0.03773585
 0.03773585 0.05660377 0.05660377 0.0754717  0.0754717  0.13207547
 0.13207547 0.1509434  0.1509434  0.22641509 0.22641509 1.        ]
[0.         0.01111111 0.66666667 0.66666667 0.87777778 0.87777778
 0.91111111 0.91111111 0.93333333 0.93333333 0.95555556 0.95555556
 0.97777778 0.97777778 0.98888889 0.98888889 1.         1.        ]
[1.99994044e+00 9.99940440e-01 9.80321221e-01 9.79957392e-01
 9.16897493e-01 9.11921727e-01 8.65643726e-01 8.34315530e-01
 8.25415569e-01 7.68102382e-01 6.94031723e-01 2.94993542e-01
 2.57568912e-01 2.36390045e-01 7.56560102e-02 3.09178515e-02
 1.08408102e-02 1.76984420e-20]
```

1.0.21 Plotting the ROC and Finding the Area Under the Curve (AUC)

[43]:
```
# find the area under the curve
roc_auc = auc(fpr, tpr)
```

[44]:
```
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.plot([0,1], [0,1], 'r--')
plt.xlim([0,1])
plt.ylim([0,1])
plt.ylabel('True Positive Rate (TPR)')
plt.xlabel('False Positive Rate (FPR)')
plt.title('Receiver Opearating CHaracteristic (ROC)')
plt.legend(loc = 'lower right')
plt.show()
```



The area under an ROC curve is a measure of the usefulness of a test in general, where a greater are means a more useful test and the areas under ROC curves are used to compare the usefulness if tests. Generally, aim for the algorithm with the highest AUC.

[ ]:

[ ]: