*You are currently looking at **version 1.3** of this notebook. To download notebooks and datafiles, as well as get help on Jupyter notebooks in the Coursera platform, visit the Jupyter Notebook FAQ (https://www.coursera.org/learn/python-machine-learning/resources/bANLa) course resource.*

Assignment 1 - Introduction to Machine Learning

For this assignment, you will be using the Breast Cancer Wisconsin (Diagnostic) Database to create a classifier that can help diagnose patients. First, read through the description of the dataset (below).

In [12]:

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer()

#print(cancer.DESCR) # Print the data set description
```

The object returned by `load_breast_cancer()` is a scikit-learn Bunch object, which is similar to a dictionary.

In [13]:

```
cancer.keys()
```

Out[13]:

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

Question 0 (Example)

How many features does the breast cancer dataset have?

This function should return an integer.

In [14]:

```
# You should write your whole answer within the function provided. The autograder will call
# this function and compare the return value against the correct solution value
def answer_zero():
    # This function returns the number of features of the breast cancer dataset, which is a
    # The assignment question description will tell you the general format the autograder i
    return len(cancer['feature_names'])

# You can examine what your function returns by calling it in the cell. If you have questio
# about the assignment formats, check out the discussion forums for any FAQs
answer_zero()
```

Out[14]:

```
30
```

Question 1

Scikit-learn works with lists, numpy arrays, scipy-sparse matrices, and pandas DataFrames, so converting the dataset to a DataFrame is not necessary for training this model. Using a DataFrame does however help make many things easier such as munging data, so let's practice creating a classifier with a pandas DataFrame.

Convert the sklearn.dataset `cancer` to a DataFrame.

This function should return a `(569, 31)` DataFrame with

columns =

```
['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',
'target']
```

and index =

```
RangeIndex(start=0, stop=569, step=1)
```

In [15]:

```
def answer_one():

    df = pd.DataFrame(cancer.data, columns=cancer.feature_names)
    df['target'] = cancer.target

    return df


answer_one()
```

Out[15]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points
0	17.990	10.38	122.80	1001.0	0.11840	0.27760	0.300100	0.147100
1	20.570	17.77	132.90	1326.0	0.08474	0.07864	0.086900	0.070170
2	19.690	21.25	130.00	1203.0	0.10960	0.15990	0.197400	0.127900
3	11.420	20.38	77.58	386.1	0.14250	0.28390	0.241400	0.105200
4	20.290	14.34	135.10	1297.0	0.10030	0.13280	0.198000	0.104300
5	12.450	15.70	82.57	477.1	0.12780	0.17000	0.157800	0.080890
6	18.250	19.98	119.60	1040.0	0.09463	0.10900	0.112700	0.074000
7	13.710	20.83	90.20	577.9	0.11890	0.16450	0.093660	0.059850
8	13.000	21.82	87.50	519.8	0.12730	0.19320	0.185900	0.093530
9	12.460	24.04	83.97	475.9	0.11860	0.23960	0.227300	0.085430
10	16.020	23.24	102.70	797.8	0.08206	0.06669	0.032990	0.033230
11	15.780	17.89	103.60	781.0	0.09710	0.12920	0.099540	0.066060
12	19.170	24.80	132.40	1123.0	0.09740	0.24580	0.206500	0.111800
13	15.850	23.95	103.70	782.7	0.08401	0.10020	0.099380	0.053640
14	13.730	22.61	93.60	578.3	0.11310	0.22930	0.212800	0.080250
15	14.540	27.54	96.73	658.8	0.11390	0.15950	0.163900	0.073640
16	14.680	20.13	94.74	684.5	0.09867	0.07200	0.073950	0.052590
17	16.130	20.68	108.10	798.8	0.11700	0.20220	0.172200	0.102800
18	19.810	22.15	130.00	1260.0	0.09831	0.10270	0.147900	0.094980
19	13.540	14.36	87.46	566.3	0.09779	0.08129	0.066640	0.047810
20	13.080	15.71	85.63	520.0	0.10750	0.12700	0.045680	0.031100
21	9.504	12.44	60.34	273.9	0.10240	0.06492	0.029560	0.020760
22	15.340	14.26	102.50	704.4	0.10730	0.21350	0.207700	0.097560
23	21.160	23.04	137.20	1404.0	0.09428	0.10220	0.109700	0.086320
24	16.650	21.38	110.00	904.6	0.11210	0.14570	0.152500	0.091700

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points
25	17.140	16.40	116.00	912.7	0.11860	0.22760	0.222900	0.140100
26	14.580	21.53	97.41	644.8	0.10540	0.18680	0.142500	0.087830
27	18.610	20.25	122.10	1094.0	0.09440	0.10660	0.149000	0.077310
28	15.300	25.27	102.40	732.4	0.10820	0.16970	0.168300	0.087510
29	17.570	15.05	115.00	955.1	0.09847	0.11570	0.098750	0.079530
...	...	...	...	...	...	...	...	...
539	7.691	25.44	48.34	170.4	0.08668	0.11990	0.092520	0.013640
540	11.540	14.44	74.65	402.9	0.09984	0.11200	0.067370	0.025940
541	14.470	24.99	95.81	656.4	0.08837	0.12300	0.100900	0.038900
542	14.740	25.42	94.70	668.6	0.08275	0.07214	0.041050	0.030270
543	13.210	28.06	84.88	538.4	0.08671	0.06877	0.029870	0.032750
544	13.870	20.70	89.77	584.8	0.09578	0.10180	0.036880	0.023690
545	13.620	23.23	87.19	573.2	0.09246	0.06747	0.029740	0.024430
546	10.320	16.35	65.31	324.9	0.09434	0.04994	0.010120	0.005495
547	10.260	16.58	65.85	320.8	0.08877	0.08066	0.043580	0.024380
548	9.683	19.34	61.05	285.7	0.08491	0.05030	0.023370	0.009615
549	10.820	24.21	68.89	361.6	0.08192	0.06602	0.015480	0.008160
550	10.860	21.48	68.51	360.5	0.07431	0.04227	0.000000	0.000000
551	11.130	22.44	71.49	378.4	0.09566	0.08194	0.048240	0.022570
552	12.770	29.43	81.35	507.9	0.08276	0.04234	0.019970	0.014990
553	9.333	21.94	59.01	264.0	0.09240	0.05605	0.039960	0.012820
554	12.880	28.92	82.50	514.3	0.08123	0.05824	0.061950	0.023430
555	10.290	27.61	65.67	321.4	0.09030	0.07658	0.059990	0.027380
556	10.160	19.59	64.73	311.7	0.10030	0.07504	0.005025	0.011160
557	9.423	27.88	59.26	271.3	0.08123	0.04971	0.000000	0.000000
558	14.590	22.68	96.39	657.1	0.08473	0.13300	0.102900	0.037360
559	11.510	23.93	74.52	403.5	0.09261	0.10210	0.111200	0.041050
560	14.050	27.15	91.38	600.4	0.09929	0.11260	0.044620	0.043040
561	11.200	29.37	70.67	386.0	0.07449	0.03558	0.000000	0.000000
562	15.220	30.62	103.40	716.9	0.10480	0.20870	0.255000	0.094290
563	20.920	25.09	143.00	1347.0	0.10990	0.22360	0.317400	0.147400
564	21.560	22.39	142.00	1479.0	0.11100	0.11590	0.243900	0.138900
565	20.130	28.25	131.20	1261.0	0.09780	0.10340	0.144000	0.097910

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points
566	16.600	28.08	108.30	858.1	0.08455	0.10230	0.092510	0.053020
567	20.600	29.33	140.10	1265.0	0.11780	0.27700	0.351400	0.152000
568	7.760	24.54	47.92	181.0	0.05263	0.04362	0.000000	0.000000

569 rows × 31 columns

Question 2

What is the class distribution? (i.e. how many instances of `malignant` (encoded 0) and how many `benign` (encoded 1)?)

This function should return a Series named `target` of length 2 with integer values and index = `['malignant', 'benign']`

In [16]:

```
def answer_two():
    cancerdf = answer_one()
    
    Distribution = pd.Series([len(cancerdf[cancerdf['target']==0]), len(cancerdf[cancerdf['
    
    return Distribution


answer_two()
```

Out[16]:

```
malignant    212
benign       357
Name: target, dtype: int64
```

Question 3

Split the DataFrame into `X` (the data) and `y` (the labels).

This function should return a tuple of length 2: `(X, y)`*, where*

- `X`*, a pandas DataFrame, has shape* `(569, 30)`
- `y`*, a pandas Series, has shape* `(569,)`.

In [20]:

```
def answer_three():
    cancerdf = answer_one()

    X = cancerdf[cancer['feature_names']]
    y = cancerdf['target']

    return X, y

answer_three()
```

```
0.3596
 15            0.65770          0.70260              0.17120
0.4218
 16            0.18710          0.29140              0.16090
0.3029
 17            0.42330          0.47840              0.20730

0.3706
 18            0.31500          0.53720              0.23880
0.2768
 19            0.17730          0.23900              0.12880
0.2977
 20            0.27760          0.18900              0.07283
0.3184
 21            0.11480          0.08867              0.06227
0.2450
 22            0.59540          0.63050              0.23930
0.4667
 23            0.26000          0.31550              0.20090
0.2822
```

Question 4

Using `train_test_split`, split `X` and `y` into training and test sets (`X_train, X_test, y_train, and y_test`).

Set the random number generator state to 0 using `random_state=0` to make sure your results match the autograder!

This function should return a tuple of length 4: `(X_train, X_test, y_train, y_test)`, *where*

- `X_train` *has shape* `(426, 30)`
- `X_test` *has shape* `(143, 30)`
- `y_train` *has shape* `(426,)`
- `y_test` *has shape* `(143,)`

```
In [19]:
```

```python
from sklearn.model_selection import train_test_split

def answer_four():
    X, y = answer_three()


    X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

    return X_train, X_test, y_train, y_test

answer_four()
```

```
 332                 0.06522
 565                 0.06637
 278                 0.06263
 489                 0.07623
 346                 0.08083
 357                 0.07113
 355                 0.07188
 112                 0.10820
 68                  0.11750
 526                 0.08665
 206                 0.07380
 65                  0.08911
 437                 0.07234
 126                 0.07900
 429                 0.06025
 392                 0.10190
 343                 0.07918
 334                 0.07207
 440                 0.09532
 441                 0.07944
```

Question 5

Using KNeighborsClassifier, fit a k-nearest neighbors (knn) classifier with `X_train`, `y_train` and using one nearest neighbor (`n_neighbors = 1`).

This function should return a `sklearn.neighbors.classification.KNeighborsClassifier`.

In [24]:

```
from sklearn.neighbors import KNeighborsClassifier

def answer_five():
    X_train, X_test, y_train, y_test = answer_four()

    knn = KNeighborsClassifier(n_neighbors = 1)
    knn.fit(X_train, y_train)

    return knn

answer_five()
```

Out[24]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=1, p=2,
           weights='uniform')
```

Question 6

Using your knn classifier, predict the class label using the mean value for each feature.

Hint: You can use `cancerdf.mean()[:-1].values.reshape(1, -1)` which gets the mean value for each feature, ignores the target column, and reshapes the data from 1 dimension to 2 (necessary for the precict method of KNeighborsClassifier).

This function should return a numpy array either `array([ 0.])` or `array([ 1.])`

In [29]:

```
def answer_six():
    cancerdf = answer_one()
    means = cancerdf.mean()[:-1].values.reshape(1, -1)
    knn = answer_five()
    answer = knn.predict(means)

    return answer
answer_six()
```

Out[29]:

```
array([1])
```

Question 7

Using your knn classifier, predict the class labels for the test set `X_test`.

This function should return a numpy array with shape `(143,)` and values either `0.0` or `1.0`.

In [30]:

```
def answer_seven():
    X_train, X_test, y_train, y_test = answer_four()
    knn = answer_five()

    ansTest = knn.predict(X_test)

    return ansTest
answer_seven()
```

Out[30]:

```
array([1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1,
       1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0,
       1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0,
       1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0,
       1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0,
       0, 1, 1, 1, 0])
```

Question 8

Find the score (mean accuracy) of your knn classifier using `X_test` and `y_test`.

This function should return a float between 0 and 1

In [31]:

```
def answer_eight():
    X_train, X_test, y_train, y_test = answer_four()
    knn = answer_five()

    meanAccuracy = knn.score(X_test, y_test)

    return meanAccuracy
answer_eight()
```

Out[31]:

```
0.916083916083916O4
```

Optional plot

Try using the plotting function below to visualize the differet predicition scores between training and test sets, as well as malignant and benign cells.

In [33]:

```
def accuracy_plot():
    import matplotlib.pyplot as plt

    %matplotlib notebook

    X_train, X_test, y_train, y_test = answer_four()

    # Find the training and testing accuracies by target value (i.e. malignant, benign)
    mal_train_X = X_train[y_train==0]
    mal_train_y = y_train[y_train==0]
    ben_train_X = X_train[y_train==1]
    ben_train_y = y_train[y_train==1]

    mal_test_X = X_test[y_test==0]
    mal_test_y = y_test[y_test==0]
    ben_test_X = X_test[y_test==1]
    ben_test_y = y_test[y_test==1]

    knn = answer_five()

    scores = [knn.score(mal_train_X, mal_train_y), knn.score(ben_train_X, ben_train_y),
              knn.score(mal_test_X, mal_test_y), knn.score(ben_test_X, ben_test_y)]


    plt.figure()

    # Plot the scores as a bar chart
    bars = plt.bar(np.arange(4), scores, color=['#4c72b0','#4c72b0','#55a868','#55a868'])

    # directly label the score onto the bars
    for bar in bars:
        height = bar.get_height()
        plt.gca().text(bar.get_x() + bar.get_width()/2, height*.90, '{0:.{1}f}'.format(heig
                     ha='center', color='w', fontsize=11)

    # remove all the ticks (both axes), and tick labels on the Y axis
    plt.tick_params(top='off', bottom='off', left='off', right='off', labelleft='off', labe

    # remove the frame of the chart
    for spine in plt.gca().spines.values():
        spine.set_visible(False)

    plt.xticks([0,1,2,3], ['Malignant\nTraining', 'Benign\nTraining', 'Malignant\nTest', 'E
    plt.title('Training and Test Accuracies for Malignant and Benign Cells', alpha=0.8)
```

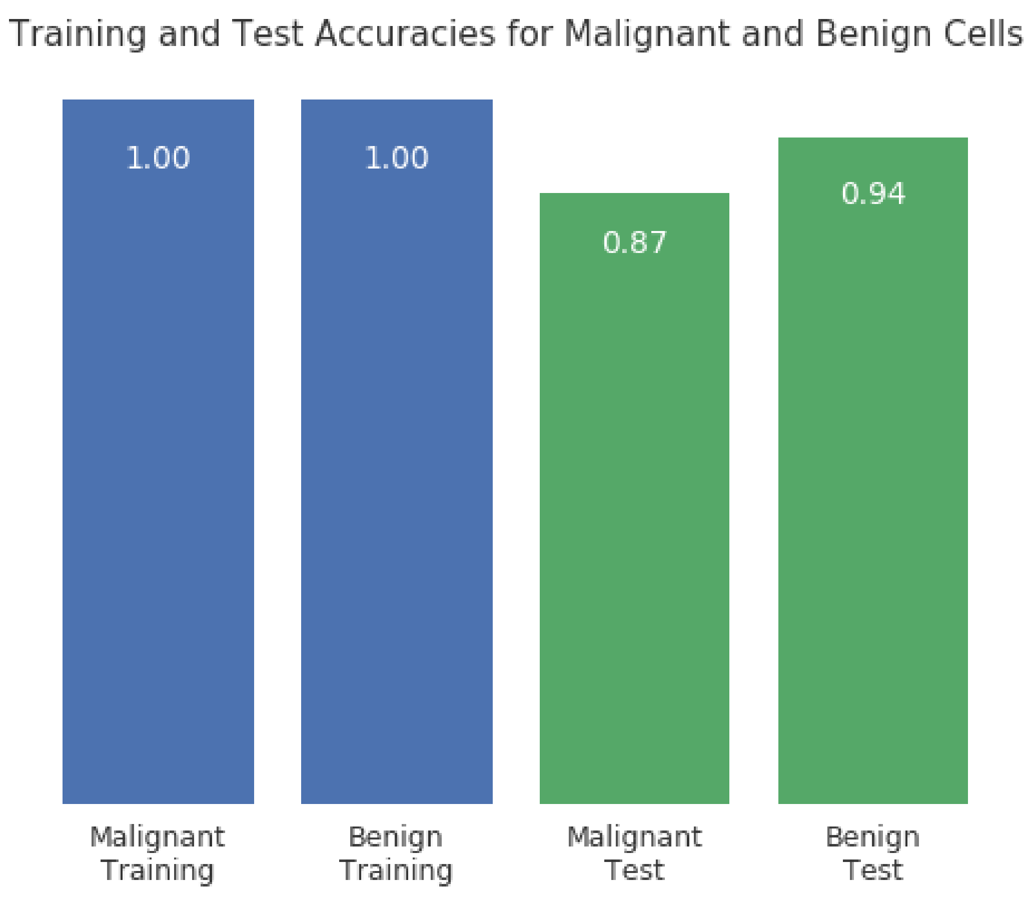
Uncomment the plotting function to see the visualization.

Comment out the plotting function when submitting your notebook for grading.

In [34]:

```
accuracy_plot()
```

Figure 1



In [ ]: