Yashshree Patil
yap14

ML Project Breast Cancer Dataset

December 19, 2020

ML PROJECT ON BREAST CANCER DATASET

```
[1]: import pandas as pd
     import csv
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
     from collections import Counter
     df = pd.read_csv ('bcw.data')
     df.head(10)
```

```
[1]:   id number  Clump Thickness  Uniformity of Cell Size  \
    0    1000025                5                        1
    1    1002945                5                        4
    2    1015425                3                        1
    3    1016277                6                        8
    4    1017023                4                        1
    5    1017122                8                       10
    6    1018099                1                        1
    7    1018561                2                        1
    8    1033078                2                        1
    9    1033078                4                        2

       Uniformity of Cell Shape  Marginal Adhesion  Single Epithelial Cell Size  \
    0                         1                  1                            2
    1                         4                  5                            7
    2                         1                  1                            2
    3                         8                  1                            3
    4                         1                  3                            2
    5                        10                  8                            7
    6                         1                  1                            2
    7                         2                  1                            2
    8                         1                  1                            2
    9                         1                  1                            2

       Bare Nuclei  Bland Chromatin  Normal Nucleoli  Mitoses  Class
    0          1.0                3                1        1      2
```

```
1         10.0                  3                2        1      2
2          2.0                  3                1        1      2
3          4.0                  3                7        1      2
4          1.0                  3                1        1      2
5         10.0                  9                7        1      4
6         10.0                  3                1        1      2
7          1.0                  3                1        1      2
8          1.0                  1                1        5      2
9          1.0                  2                1        1      2
```

[2]:
```
df.info()
df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 699 entries, 0 to 698
Data columns (total 11 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   id number                    699 non-null    int64
 1   Clump Thickness              699 non-null    int64
 2   Uniformity of Cell Size      699 non-null    int64
 3   Uniformity of Cell Shape     699 non-null    int64
 4   Marginal Adhesion            699 non-null    int64
 5   Single Epithelial Cell Size  699 non-null    int64
 6   Bare Nuclei                  683 non-null    float64
 7   Bland Chromatin              699 non-null    int64
 8   Normal Nucleoli              699 non-null    int64
 9   Mitoses                      699 non-null    int64
 10  Class                        699 non-null    int64
dtypes: float64(1), int64(10)
memory usage: 60.2 KB
```

[2]:
```
          id number  Clump Thickness  Uniformity of Cell Size  \
count  6.990000e+02       699.000000               699.000000
mean   1.071704e+06         4.417740                 3.134478
std    6.170957e+05         2.815741                 3.051459
min    6.163400e+04         1.000000                 1.000000
25%    8.706885e+05         2.000000                 1.000000
50%    1.171710e+06         4.000000                 1.000000
75%    1.238298e+06         6.000000                 5.000000
max    1.345435e+07        10.000000                10.000000

       Uniformity of Cell Shape  Marginal Adhesion  \
count                699.000000         699.000000
mean                   3.207439           2.806867
std                    2.971913           2.855379
min                    1.000000           1.000000
```

```
25%                      1.000000            1.000000
50%                      1.000000            1.000000
75%                      5.000000            4.000000
max                     10.000000           10.000000

       Single Epithelial Cell Size  Bare Nuclei  Bland Chromatin  \
count                   699.000000   683.000000       699.000000
mean                      3.216023     3.544656         3.437768
std                       2.214300     3.643857         2.438364
min                       1.000000     1.000000         1.000000
25%                       2.000000     1.000000         2.000000
50%                       2.000000     1.000000         3.000000
75%                       4.000000     6.000000         5.000000
max                      10.000000    10.000000        10.000000

       Normal Nucleoli     Mitoses       Class
count       699.000000  699.000000  699.000000
mean          2.866953    1.589413    2.689557
std           3.053634    1.715078    0.951273
min           1.000000    1.000000    2.000000
25%           1.000000    1.000000    2.000000
50%           1.000000    1.000000    2.000000
75%           4.000000    1.000000    4.000000
max          10.000000   10.000000    4.000000
```

[3]:
```
#Converting Bare Nuclei Feature to integer
df['Bare Nuclei'] = df['Bare Nuclei'].astype("Float32").astype("Int32")
```

1. There are 699 datapoints in total.
2. Missing attribute values: 16 There are 16 instances in Groups 1 to 6 that contain a single missing (i.e., unavailable) attribute value, now denoted by "?".
3. Class distribution:

   2 = Benign: 458 (65.5%) 4 = Malignant: 241 (34.5%)

The ID number consist of unique values and hence hold no meaning in prediction, hence can be dropped.

[4]:
```
df = df.drop(['id number'], axis = 1)
```

[5]:
```
df.head(10)
```

[5]:
```
   Clump Thickness  Uniformity of Cell Size  Uniformity of Cell Shape  \
0                5                        1                          1
1                5                        4                          4
2                3                        1                          1
3                6                        8                          8
```

```
4               4                       1                          1
5               8                      10                         10
6               1                       1                          1
7               2                       1                          2
8               2                       1                          1
9               4                       2                          1

   Marginal Adhesion  Single Epithelial Cell Size  Bare Nuclei  \
0                  1                            2            1
1                  5                            7           10
2                  1                            2            2
3                  1                            3            4
4                  3                            2            1
5                  8                            7           10
6                  1                            2           10
7                  1                            2            1
8                  1                            2            1
9                  1                            2            1

   Bland Chromatin  Normal Nucleoli  Mitoses  Class
0                3                1        1      2
1                3                2        1      2
2                3                1        1      2
3                3                7        1      2
4                3                1        1      2
5                9                7        1      4
6                3                1        1      2
7                3                1        1      2
8                1                1        5      2
9                2                1        1      2
```

DATA PREPROCESSING

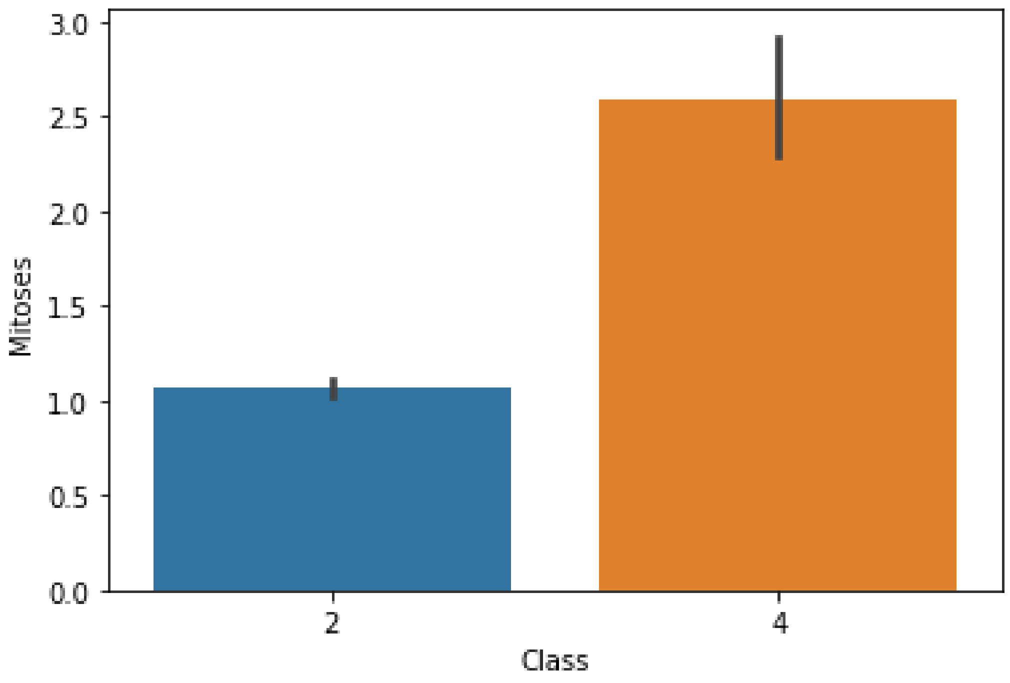
To check which features are more relevant than the others, and does our model correctly predicts it?

Lets start with filling the missing values.

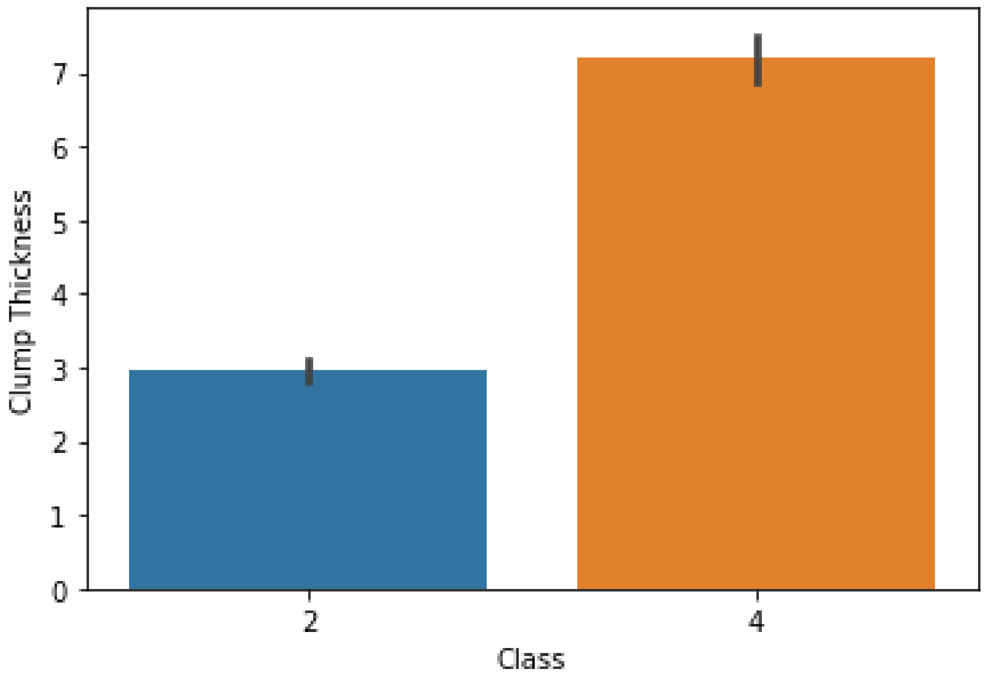
[9]:
```
# As all the Features have discrete values I am using mode of the features to␣
 ↪fill the missing values.
for x in df.keys():
    a = np.array(df[x])
    b = Counter(a)
    df[x] = df[x].fillna(np.argmax(b.most_common(1)))

df.head(10)
```

```
[9]:    Clump Thickness  Uniformity of Cell Size  Uniformity of Cell Shape  \
     0                5                        1                         1
     1                5                        4                         4
     2                3                        1                         1
     3                6                        8                         8
     4                4                        1                         1
     5                8                       10                        10
     6                1                        1                         1
     7                2                        1                         2
     8                2                        1                         1
     9                4                        2                         1

        Marginal Adhesion  Single Epithelial Cell Size  Bare Nuclei  \
     0                  1                            2            1
     1                  5                            7           10
     2                  1                            2            2
     3                  1                            3            4
     4                  3                            2            1
     5                  8                            7           10
     6                  1                            2           10
     7                  1                            2            1
     8                  1                            2            1
     9                  1                            2            1

        Bland Chromatin  Normal Nucleoli  Mitoses  Class
     0                3                1        1      2
     1                3                2        1      2
     2                3                1        1      2
     3                3                7        1      2
     4                3                1        1      2
     5                9                7        1      4
     6                3                1        1      2
     7                3                1        1      2
     8                1                1        5      2
     9                2                1        1      2
```

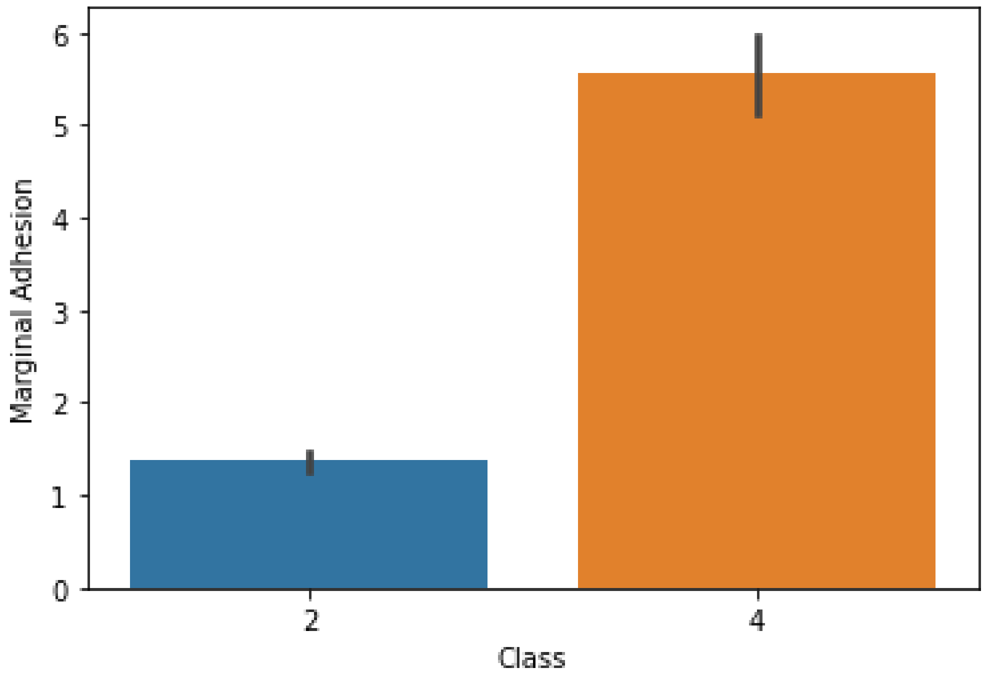
```
[10]: sns.barplot(x="Class", y="Mitoses", data=df)
      plt.show()
```



[11]:
```
sns.barplot(x="Class", y="Clump Thickness", data=df)
plt.show()
```



[12]:
```
sns.barplot(x="Class", y="Marginal Adhesion", data=df)
plt.show()
```

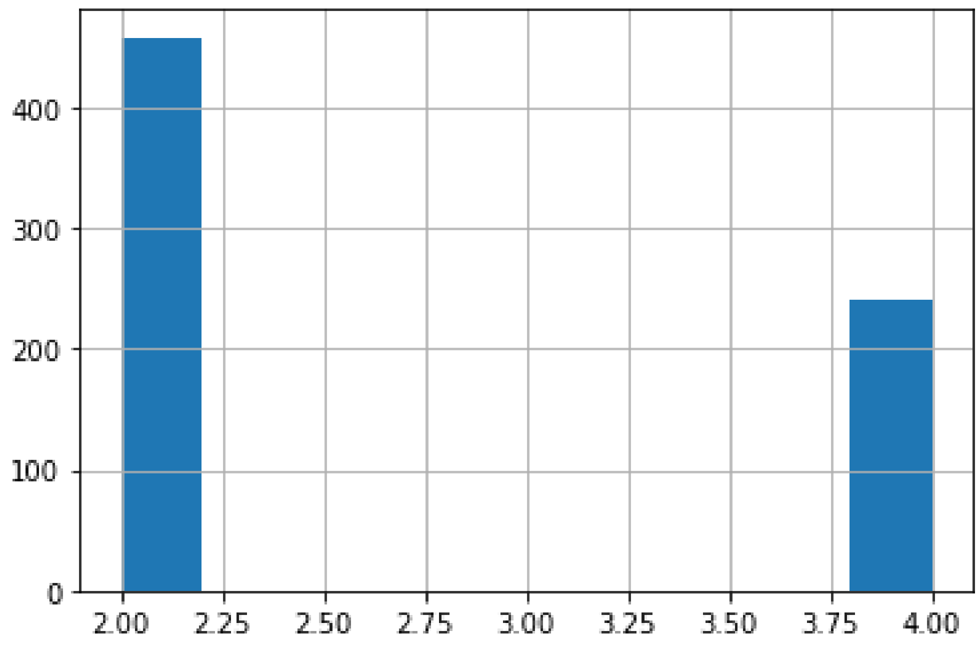


From the above graph we can say that if the value of Mitosis, Clump Thickness and Marginal Adhesion is less than 5 then the Class is Benign and if the values are greater than that of 5 then the class is Malignant. Using this Information we can also turn this categorical data to Binary data, By setting all the values <= 5 to 0 and values > 5 to 1

And for the Class Label we can set the Benign class to 0 Malignant class to 1 But, for now lets play some more with the data and its tables Because from the above Graphs we cannot determine that which feature is important and which is not. (that is relevance of the feature is not clear yet)

[17]:
```
df["Class"].hist()
```

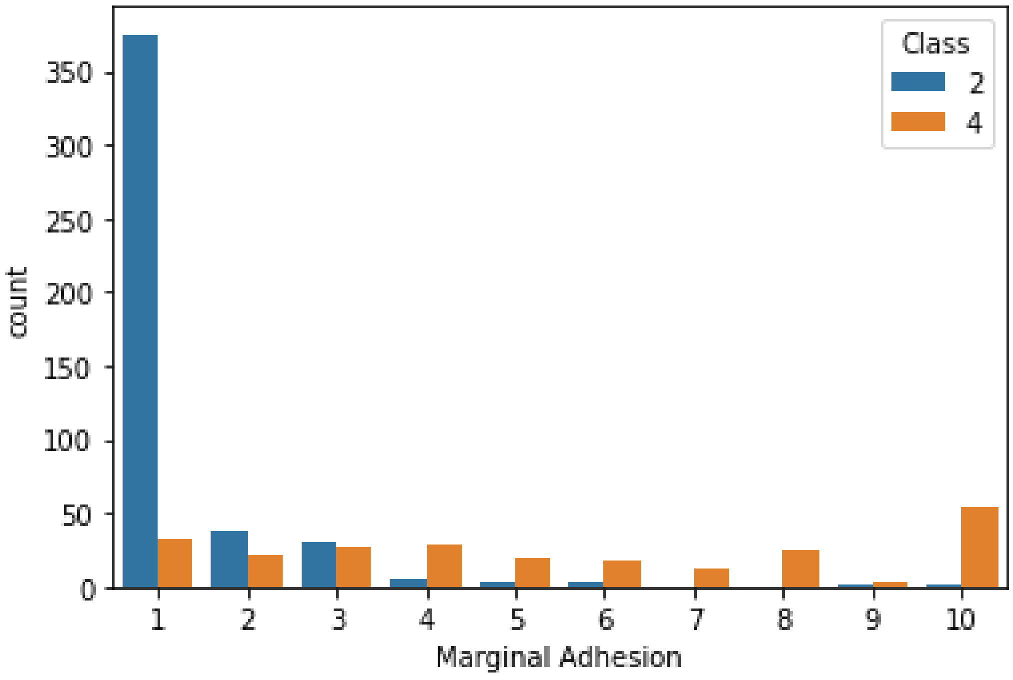
[17]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8f27f6a7f0>
```



As state earlier total Benign entries = 458 (65.5%) total Malignant entries = 241 (34.5%)

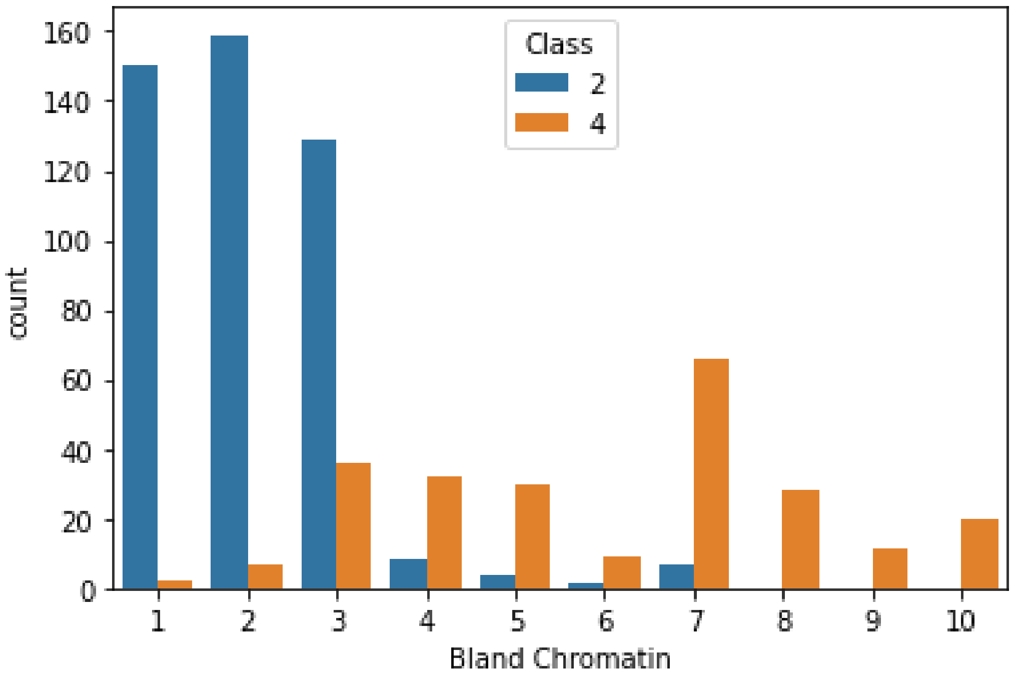
[16]:
```
sns.countplot(x="Marginal Adhesion", hue="Class", data=df)
```

[16]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8f27a782b0>
```



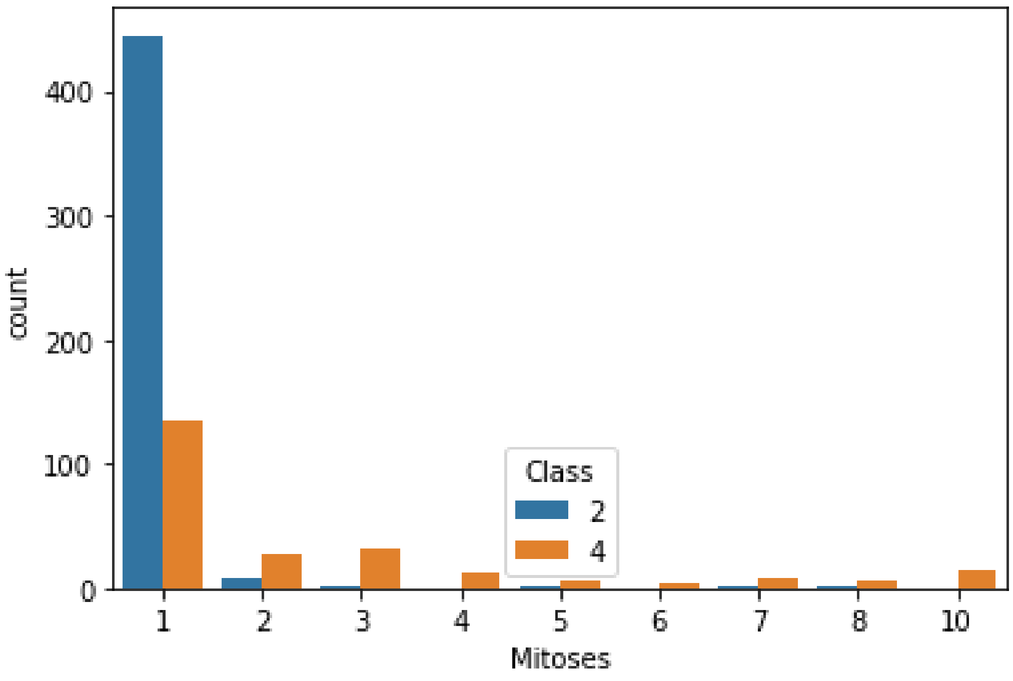
[18]:
```
sns.countplot(x="Bland Chromatin", hue="Class", data=df)
```

[18]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8f280939a0>
```



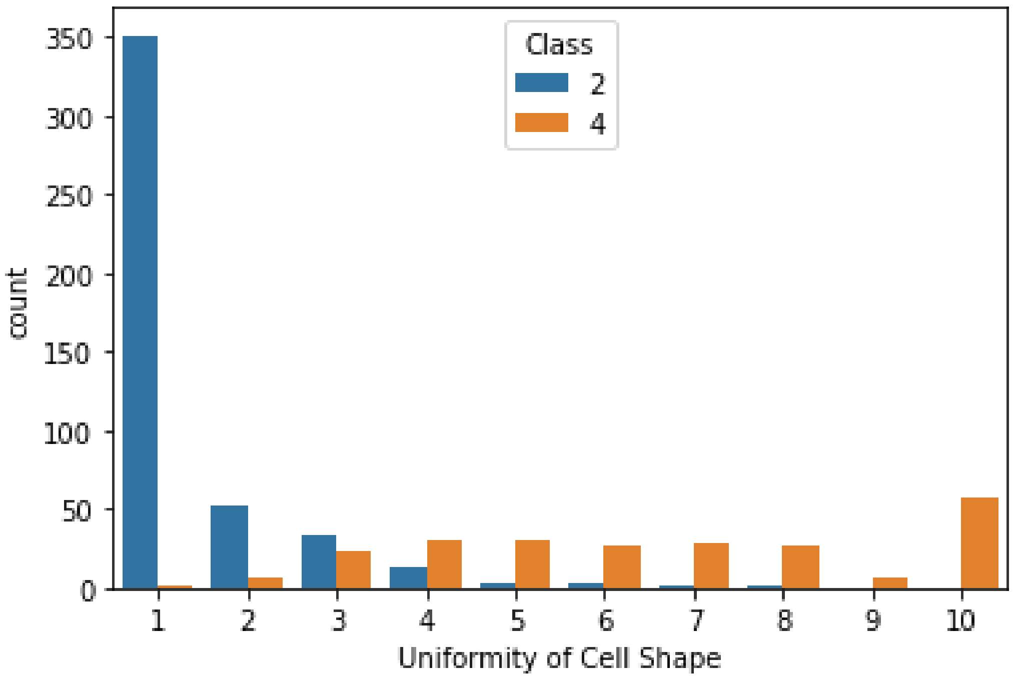
[19]:
```
sns.countplot(x="Mitoses", hue="Class", data=df)
```

[19]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8f281c1370>
```



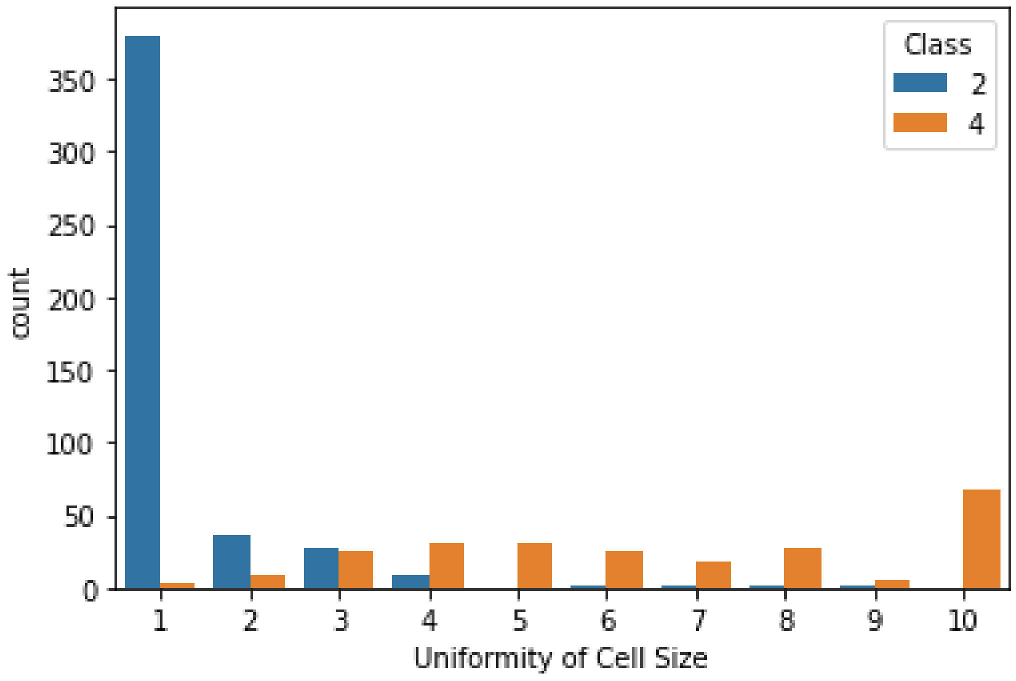
```
[20]: sns.countplot(x="Uniformity of Cell Shape", hue="Class", data=df)
```

```
[20]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8f27a6b370>
```



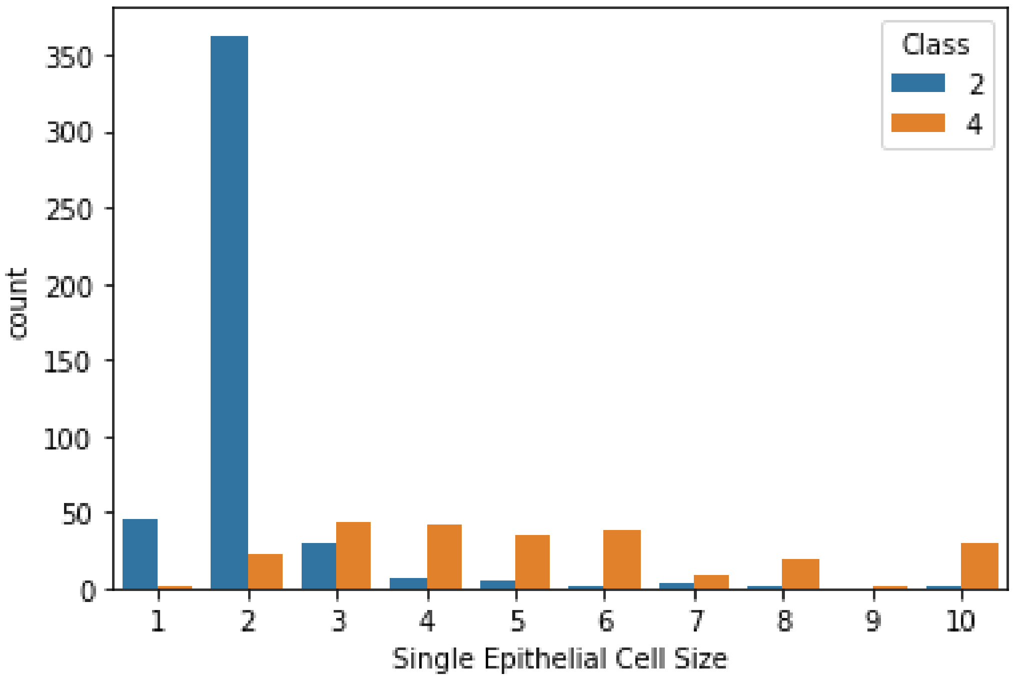
[21]:
```
sns.countplot(x="Uniformity of Cell Size", hue="Class", data=df)
```

[21]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8f276ba7f0>
```



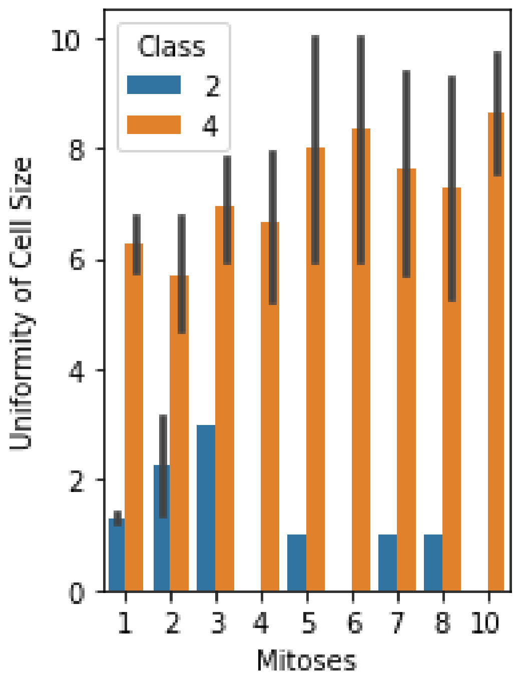
[22]:
```
sns.countplot(x="Single Epithelial Cell Size", hue="Class", data=df)
```

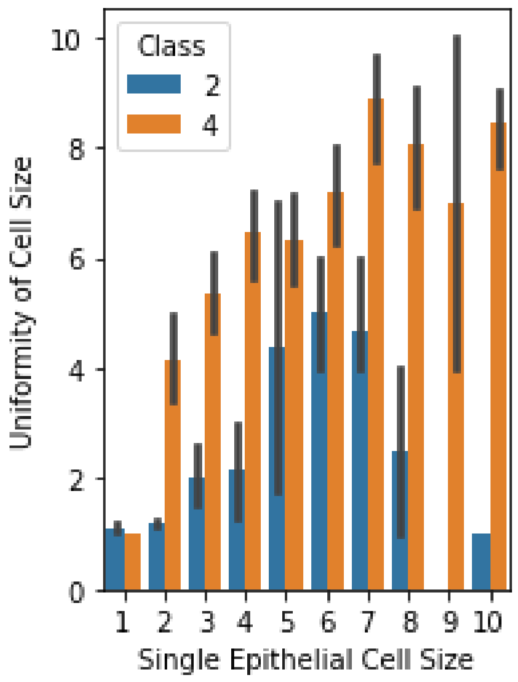
[22]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8f278d9940>
```

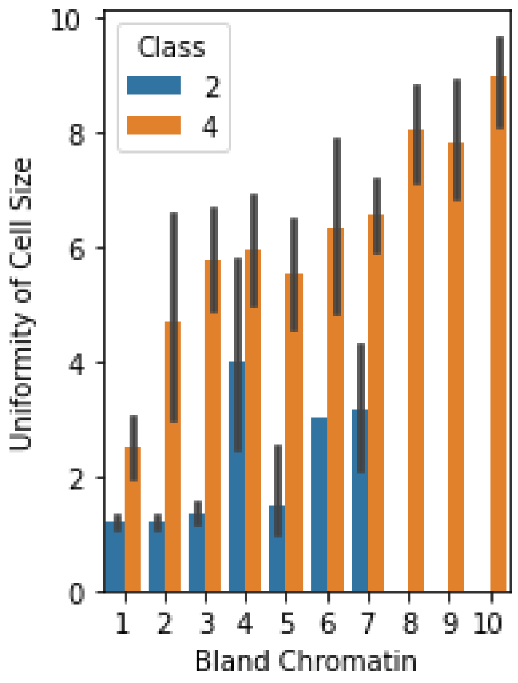


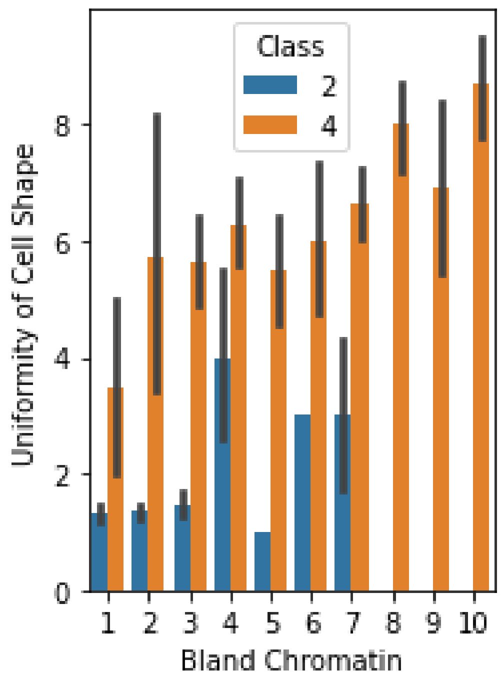
From the above graphs we can say that Uniformity of cell size, Uniformity of cell shape and Bland Chromatin appears to be the most important features so far. Because there is a huge difference in count of Classes at a single value of x axis. For example let us take uniformity of the cell size by looking at the garph we can say that if the Uniformity of the cell size is 1 then for sure the class is going to be benign and if the uniformity of cell size is 10 then for sure the class is going to be Malignant

[27]:
```
plt.subplot(122)
sns.barplot(x="Mitoses", y="Uniformity of Cell Size", hue="Class", data=df)
plt.show()
plt.subplot(122)
sns.barplot(x="Single Epithelial Cell Size", y="Uniformity of Cell Size",␣
 ↪hue="Class", data=df)
plt.show()
plt.subplot(122)
sns.barplot(x="Bland Chromatin", y="Uniformity of Cell Size", hue="Class",␣
 ↪data=df)
plt.show()
plt.subplot(122)
sns.barplot(x="Bland Chromatin", y="Uniformity of Cell Shape", hue="Class",␣
 ↪data=df)
plt.show()
```


Class
2
4
Uniformity of Cell Size
Mitoses
0
2
4
6
8
10
1
2
3
4
5
6
7
8
10


Class
2
4
Uniformity of Cell Size
Single Epithelial Cell Size
0
2
4
6
8
10
1
2
3
4
5
6
7
8
9
10

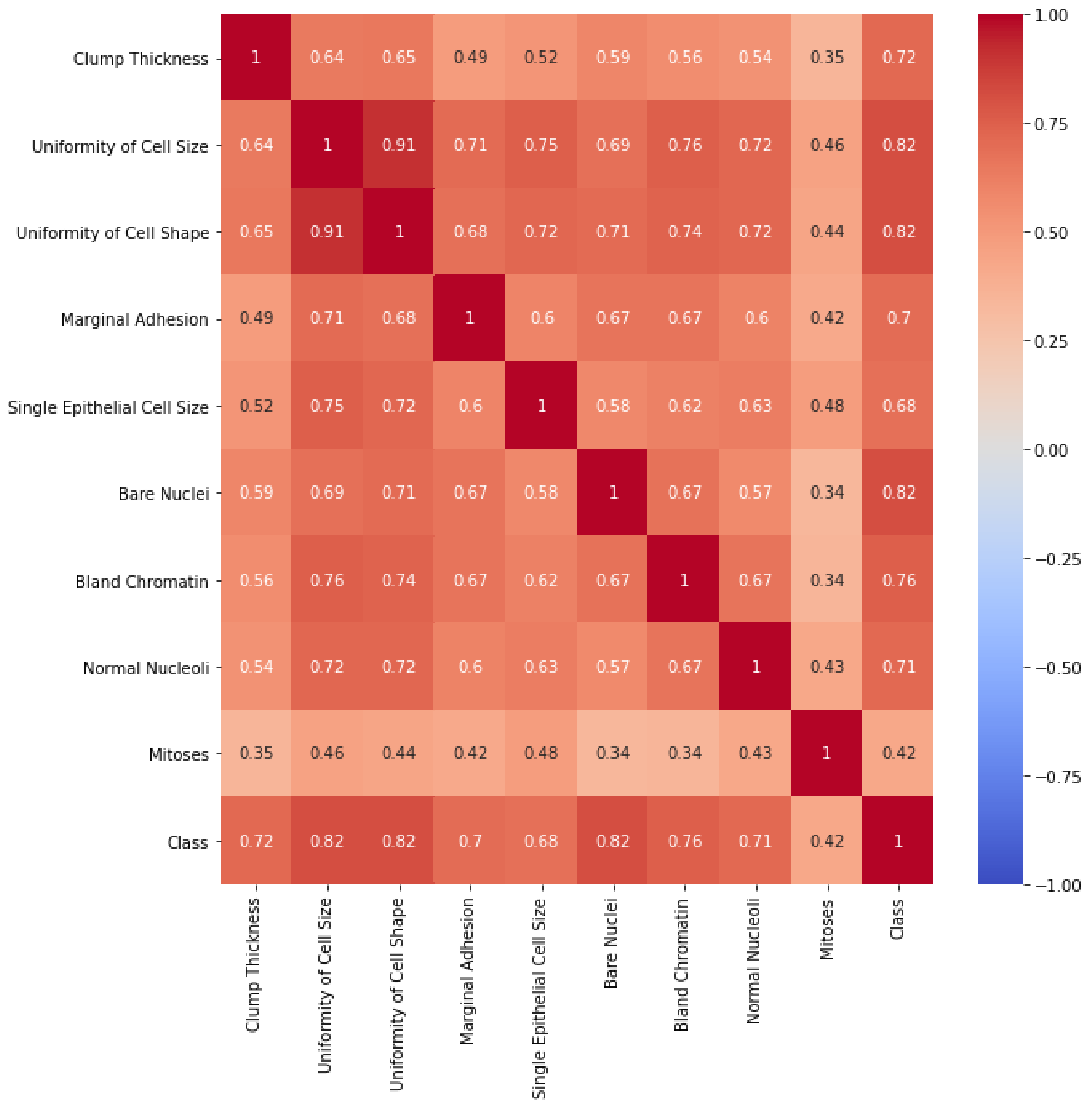




From the last 2 figures we can say that if the value of Bland Chromatin is 8, 9 and 10. Then the Uniformity of cell shape and Uniformity of cell size is surely going to be 0 hence, if the Bland Chromatin values is known then we can determine where to check for Uniformity value or not.

Hence, Bland Chromatin may be more relevant.

[28]:
```
plt.figure(1, figsize=(10,10))
sns.heatmap(df.corr(), vmin=-1, cmap='coolwarm', annot=True)
plt.show()
```



	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
Clump Thickness	1	0.64	0.65	0.49	0.52	0.59	0.56	0.54	0.35	0.72
Uniformity of Cell Size	0.64	1	0.91	0.71	0.75	0.69	0.76	0.72	0.46	0.82
Uniformity of Cell Shape	0.65	0.91	1	0.68	0.72	0.71	0.74	0.72	0.44	0.82
Marginal Adhesion	0.49	0.71	0.68	1	0.6	0.67	0.67	0.6	0.42	0.7
Single Epithelial Cell Size	0.52	0.75	0.72	0.6	1	0.58	0.62	0.63	0.48	0.68
Bare Nuclei	0.59	0.69	0.71	0.67	0.58	1	0.67	0.57	0.34	0.82
Bland Chromatin	0.56	0.76	0.74	0.67	0.62	0.67	1	0.67	0.34	0.76
Normal Nucleoli	0.54	0.72	0.72	0.6	0.63	0.57	0.67	1	0.43	0.71
Mitoses	0.35	0.46	0.44	0.42	0.48	0.34	0.34	0.43	1	0.42
Class	0.72	0.82	0.82	0.7	0.68	0.82	0.76	0.71	0.42	1

From the above heat map we can say that the most strongly corelated features are Uniformity of cell shape and uniformity of cell size after that the most relevant features can be checked by looking at the corelation between the feature and the target class. Here, Those features are Uniformity of cell shape, Uniformity of cell size, bare nuclei and Bland Chromatin

MODELS AND ALGORITHMS

Fitting the data:

As discussed earlier "From the above bar graphs we can say that if the value of Mitosis, Clump Thickness and Marginal Adhesion is less than 5 then the Class is Benign and if the values are greater than that of 5 then the class is Malignant. Using this Information we can also turn this categorical data to Binary data, By setting all the values <= 5 to 0 and values > 5 to 1. And for the Class Label we can set the Benign class to 0 Malignant class to 1"

The data can fitted in a Decision tree model using the above mentioned method.

DECISION TREE CODE USED FROM ASSIGNMENT 2 (DECISION TREE WITH ENTROPY)

[29]:
```
dataset = df
# DATA PRE_PROCESSING
for x in dataset.keys():
    if x!= 'Class':
        dataset.loc[dataset[x] <= 5, x] = 0
        dataset.loc[(dataset[x] > 5) ,x] = 1
    else:
        dataset.loc[dataset[x] <= 3, x] = 0
        dataset.loc[(dataset[x] > 3) ,x] = 1
```

[40]:
```
m = 699
import pandas as pd
import numpy as np
from pprint import pprint
from numpy import random
#Entropy Calculation
def entropy(target):
    ele,count = np.unique(target,return_counts=True)
    entropy = np.sum([(-count[i]/np.sum(count))*np.log2(count[i]/np.sum(count))␣
 ↪for i in range(len(ele))])
    return entropy

#Information Gain Calculation

def InfoGain(data,split_attribute,target_name="Class"):
    total_entropy = entropy(data[target_name])
    vals,counts = np.unique(data[split_attribute],return_counts=True)
    Weighted_Entropy = np.sum([(counts[i]/np.sum(counts))*entropy(data.
 ↪where(data[split_attribute]==vals[i]).
                                dropna()[target_name])for i in range(len(vals))])

    #Information gain = Total Entropy - Weighted Entropy
    Information_Gain = total_entropy - Weighted_Entropy
    return Information_Gain

def ID3(data,originaldata,features,target_attribute_name="Class",
        parent_node_class=None):
```

```
    #If all target_values have the same value, Then return this value
    if len(np.unique(data[target_attribute_name])) <= 1:
        return np.unique(data[target_attribute_name])[0]

    #If the dataset is empty
    elif len(data) == 0:
        return np.unique(originaldata[target_attribute_name])[np.argmax(np.
↪unique(originaldata[target_attribute_name],
                                                                               ␣
↪return_counts=True)[1])]

    #If the feature space is empty i.e. Attribute list == 0
    elif len(features) == 0:
        return parent_node_class

    #If none of the above condition holds true grow the tree and look for its␣
↪subtree
    else:
        parent_node_class = np.unique(data[target_attribute_name])[np.argmax(np.
↪unique(data[target_attribute_name],
                                                                               ␣
↪return_counts=True)[1])]

    #Select the feature which best splits the dataset having the heightest info␣
↪gain
    item_values = [InfoGain(data,feature,target_attribute_name)for feature in␣
↪features] #Return the infgain values
    best_feature_index = np.argmax(item_values)
    best_feature = features[best_feature_index]

    #Create the tree structure; Tree in the form of dictionary
    tree = {best_feature:{}}

    #Remove the feature with the best info gain once added to the tree
    features = [i for i in features if i != best_feature]

    #Grow the tree branch under the root node
    for value in np.unique(data[best_feature]):
        value = value
        sub_data = data.where(data[best_feature]==value).dropna()
        #call the ID3 algotirthm
        subtree =␣
↪ID3(sub_data,dataset,features,target_attribute_name,parent_node_class)
        #Add the subtree
        tree[best_feature][value] = subtree
    return(tree)
```

```
#Predict
def predict(query,tree,default=1):
    for key in list(query.keys()):
        # if all the target values are same. then the tree will have no key.␣
↪Hence, try and except block
        try:
            if key in list(tree.keys()):
                try:
                    result = tree[key][query[key]]
                except:
                    return default

                result = tree[key][query[key]]
                if isinstance(result,dict):
                    return predict(query,result)
                else:
                    return result
        except:
            return tree

# split the data into train and test data with 80:20 proportion
def split(dataset):
    train_data = dataset.iloc[:round(4*m/5)].reset_index(drop=True)
    test_data = dataset.iloc[round(4*m/5):].reset_index(drop=True)
    return train_data,test_data

train_data = split(dataset)[0]
print("TRAIN DATA")
print(train_data)
test_data = split(dataset)[1]
print("TEST DATA")
print(test_data)

def test(data,tree):
    queries = data.iloc[:,:-1].to_dict(orient="records")
    predicted = pd.DataFrame(columns=["predicted"])
    #calculation of accuracy and error
    for i in range(len(data)):
        predicted.loc[i,"predicted"] = predict(queries[i],tree,1.0)

    print("The Prediction accuracy is:",(np.
↪sum(predicted["predicted"]==data["Class"])/len(data))*100,'%')
    print(f'error {100 - (np.sum(predicted["predicted"]==data["Class"])/
↪len(data))*100}%')

    return 1 - (np.sum(predicted["predicted"]==data["Class"])/len(data))
```

```
tree = ID3(train_data,train_data,train_data.columns[:-1])
pprint(tree)

error = test(test_data,tree)
error_train = test(train_data,tree)

print(f"Error_train(f) is: {error_train} \ni.e. {error_train*100}% \nError(f) is:
 ↪ {error} \ni.e. {error*100}%")
```

```
TRAIN DATA
     Clump Thickness  Uniformity of Cell Size  Uniformity of Cell Shape  \
0                  0                        0                         0
1                  0                        0                         0
2                  0                        0                         0
3                  1                        1                         1
4                  0                        0                         0
..               ...                      ...                       ...
554                0                        0                         0
555                0                        0                         0
556                0                        0                         0
557                0                        0                         0
558                0                        0                         0

     Marginal Adhesion  Single Epithelial Cell Size  Bare Nuclei  \
0                    0                            0            0
1                    0                            1            1
2                    0                            0            0
3                    0                            0            0
4                    0                            0            0
..                 ...                          ...          ...
554                  0                            0            0
555                  0                            0            0
556                  0                            0            0
557                  0                            0            0
558                  0                            0            0

     Bland Chromatin  Normal Nucleoli  Mitoses  Class
0                  0                0        0      0
1                  0                0        0      0
2                  0                0        0      0
3                  0                1        0      0
4                  0                0        0      0
..               ...              ...      ...    ...
554                0                0        0      0
555                0                1        0      0
```

```
556                0                0        0      0
557                0                0        0      0
558                0                0        0      0

[559 rows x 10 columns]
TEST DATA
     Clump Thickness  Uniformity of Cell Size  Uniformity of Cell Shape  \
0                  0                        0                         0
1                  0                        0                         0
2                  0                        0                         0
3                  0                        0                         0
4                  0                        0                         0
..               ...                      ...                       ...
135                0                        0                         0
136                0                        0                         0
137                0                        1                         1
138                0                        1                         1
139                0                        1                         1

     Marginal Adhesion  Single Epithelial Cell Size  Bare Nuclei  \
0                    0                            0            0
1                    0                            0            0
2                    0                            0            0
3                    0                            0            0
4                    0                            0            0
..                 ...                          ...          ...
135                  0                            0            0
136                  0                            0            0
137                  0                            1            0
138                  0                            0            0
139                  0                            0            0

     Bland Chromatin  Normal Nucleoli  Mitoses  Class
0                  0                0        0      0
1                  0                0        0      0
2                  0                0        0      0
3                  0                0        0      0
4                  0                0        0      0
..               ...              ...      ...    ...
135                0                0        0      0
136                0                0        0      0
137                1                1        0      1
138                1                1        0      1
139                1                0        0      1

[140 rows x 10 columns]
{'Bare Nuclei': {0: {'Clump Thickness': {0.0: {'Uniformity of Cell Size': {0.0:
{'Marginal Adhesion': {0.0: {'Normal Nucleoli': {0.0: {'Bland Chromatin': {0.0:
```

```
{'Uniformity of Cell Shape': {0.0: {'Single Epithelial Cell Size': {0.0:
{'Mitoses': {0.0: 0.0,
      1.0: 0.0}},
                                                                    1.0: 0.0}},
                              1.0: 0.0}},
                                                                           1.0:
0.0}},
                                                 1.0: {'Single Epithelial Cell
Size': {0.0: {'Uniformity of Cell Shape': {0.0: {'Bland Chromatin': {0.0:
{'Mitoses': {0.0: 0.0}}}}}},
       1.0: 0.0}}}},
                       1.0: {'Uniformity of Cell Shape': {0.0: {'Single
Epithelial Cell Size': {0.0: {'Bland Chromatin': {0.0: {'Normal Nucleoli': {0.0:
{'Mitoses': {0.0: 0.0}}}}}}}}}}},
                                                                           1.0:
{'Uniformity of Cell Shape': {0.0: {'Normal Nucleoli': {0.0: 1.0,
                                                        1.0: 0.0}},
                              1.0: 1.0}}}},
                                         1.0: {'Uniformity of Cell Size': {0.0:
{'Normal Nucleoli': {0.0: {'Single Epithelial Cell Size': {0.0: {'Mitoses':
{0.0: {'Bland Chromatin': {0.0: {'Uniformity of Cell Shape': {0.0: {'Marginal
Adhesion': {0.0: 0.0}},
                                                          1.0: {'Marginal
Adhesion': {0.0: 0.0}}}},
                       1.0: {'Uniformity of Cell Shape': {0.0: {'Marginal
Adhesion': {0.0: 1.0}}}}}},
1.0: 1.0}},
                                                           1.0: 1.0}},
                     1.0: 1.0}},
                                                                           1.0:
{'Uniformity of Cell Shape': {0.0: 1.0,
                              1.0: {'Mitoses': {0.0: {'Normal Nucleoli': {0.0:
1.0,
                                                                          1.0:
{'Bland Chromatin': {0.0: {'Single Epithelial Cell Size': {0.0: {'Marginal
Adhesion': {0.0: 1.0,
      1.0: 1.0}},
                                                          1.0: 1.0}},
                    1.0: {'Single Epithelial Cell Size': {0.0: 1.0,
                                                          1.0: {'Marginal
Adhesion': {0.0: 1.0,
      1.0: 1.0}}}}}}}},
                                                1.0: 1.0}}}}}}},
                 1: {'Marginal Adhesion': {0.0: {'Bland Chromatin': {0.0:
{'Normal Nucleoli': {0.0: {'Clump Thickness': {0.0: {'Uniformity of Cell Size':
{0.0: {'Uniformity of Cell Shape': {0.0: {'Single Epithelial Cell Size': {0.0:
{'Mitoses': {0.0: 0.0}},
                                                                    1.0:
```

```
{'Mitoses': {0.0: 1.0}}}},
                              1.0: 1.0}},
                                                                                    1.0:
0.0}},
                                        1.0: {'Single Epithelial Cell Size':
{0.0: {'Mitoses': {0.0: {'Uniformity of Cell Size': {0.0: {'Uniformity of Cell
Shape': {0.0: 1.0,
      1.0: 1.0}},
                                                   1.0: {'Uniformity of Cell
Shape': {0.0: 1.0,
      1.0: 1.0}}}},
                 1.0: 1.0}},
1.0: 1.0}}}},
               1.0: 1.0}},
                                                                              1.0:
{'Uniformity of Cell Shape': {0.0: {'Normal Nucleoli': {0.0: 1.0,
                                                  1.0: {'Single Epithelial Cell
Size': {0.0: {'Clump Thickness': {0.0: 1.0,
                                 1.0: {'Uniformity of Cell Size': {0.0:
{'Mitoses': {0.0: 1.0}}}}}},
        1.0: 1.0}}}},
                        1.0: 1.0}}}},
                                          1.0: 1.0}}}}
The Prediction accuracy is: 98.57142857142858 %
error 1.4285714285714164%
The Prediction accuracy is: 96.2432915921288 %
error 3.7567084078712014%
Error_train(f) is: 0.03756708407871201
i.e. 3.756708407871201%
Error(f) is: 0.014285714285714235
i.e. 1.4285714285714235%
```

RESULTS:

From the above decision tree we can can observe that the most relevant feature/ the root node is Bare Nuclei which had a high co-relation with the class label (Target value) The other relevant features that can be seen in the decision tree are Uniformity of cell shape, uniformity of cell size, Bland Chromatin and Clump Thickness.

Predicted testing accuracy is 98.57% with testing error of 1.428%

Predicted training accuracy is 96.243% with training error of 3.756%

Were you able to train over all features?

Trained over all features (Excluding id number)

What kinds of computational tradeoffs did you face, and how did you settle them?

Id number trade off (Each entry had a different id number hence was a irrelevant feature in the data) Therefore, Dropped the column id number from the dataset

DECISION TREE WITH GINI INDEX

[39]:
```
def Gini (target):
    elements, counts = np.unique(target, return_counts = True)
    gini = 1 - np.sum([(counts[i]/np.sum(counts))**2 for i in␣
 ↪range(len(elements))])
    return gini

def GiniIndex (data, split_attribute, target_name = 'Class'):
    total_gini = Gini(data[target_name])
    vals,counts = np.unique(data[split_attribute],return_counts=True)

    gini_index = np.sum([(counts[i]/np.sum(counts))*Gini(data.
 ↪where(data[split_attribute]==vals[i]).
                                                                ␣
 ↪dropna()[target_name])for i in range(len(vals))])
    return gini_index

def ID3(data,originaldata,features,target_attribute_name="Class",
        parent_node_class=None):

    #If all target_values have the same value, Then return this value
    if len(np.unique(data[target_attribute_name])) <= 1:
        return np.unique(data[target_attribute_name])[0]

    #If the dataset is empty
    elif len(data) == 0:
        return np.unique(originaldata[target_attribute_name])[np.argmax(np.
 ↪unique(originaldata[target_attribute_name],
                                                                              ␣
 ↪return_counts=True)[1])]

    #If the feature space is empty i.e. Attribute list == 0
    elif len(features) == 0:
        return parent_node_class

    #If none of the above condition holds true grow the tree and look for its␣
 ↪subtree
    else:
        parent_node_class = np.unique(data[target_attribute_name])[np.argmax(np.
 ↪unique(data[target_attribute_name],
                                                                              ␣
 ↪return_counts=True)[1])]

    #Select the feature which best splits the dataset having the heightest info␣
 ↪gain
```

```
    item_values = [GiniIndex(data,feature,target_attribute_name)for feature in␣
↪features] #Return the infgain values
    best_feature_index = np.argmax(item_values)
    best_feature = features[best_feature_index]

    #Create the tree structure; Tree in the form of dictionary
    tree = {best_feature:{}}

    #Remove the feature with the best info gain once added to the tree
    features = [i for i in features if i != best_feature]

    #Grow the tree branch under the root node
    for value in np.unique(data[best_feature]):
        value = value
        sub_data = data.where(data[best_feature]==value).dropna()
        #call the ID3 algotirthm
        subtree =␣
↪ID3(sub_data,dataset,features,target_attribute_name,parent_node_class)
        #Add the subtree
        tree[best_feature][value] = subtree
    return(tree)

#Predict
def predict(query,tree,default=1):
    for key in list(query.keys()):
        # if all the target values are same. then the tree will have no key.␣
↪Hence, try and except block
        try:
            if key in list(tree.keys()):
                try:
                    result = tree[key][query[key]]
                except:
                    return default

                result = tree[key][query[key]]
                if isinstance(result,dict):
                    return predict(query,result)
                else:
                    return result
        except:
            return tree

# split the data into train and test data with 80:20 proportion
def split(dataset):
    train_data = dataset.iloc[:round(4*m/5)].reset_index(drop=True)
    test_data = dataset.iloc[round(4*m/5):].reset_index(drop=True)
    return train_data,test_data
```

```
train_data = split(dataset)[0]
print("TRAIN DATA")
print(train_data)
test_data = split(dataset)[1]
print("TEST DATA")
print(test_data)

def test(data,tree):
    queries = data.iloc[:,:-1].to_dict(orient="records")
    predicted = pd.DataFrame(columns=["predicted"])
    #calculation of error and accuracy
    for i in range(len(data)):
        predicted.loc[i,"predicted"] = predict(queries[i],tree,1.0)
    print("The Prediction accuracy is:",(np.
 ↪sum(predicted["predicted"]==data["Class"])/len(data))*100,'%')
    print(f'error {100 - (np.sum(predicted["predicted"]==data["Class"])/
 ↪len(data))*100}%')
    return 1 - (np.sum(predicted["predicted"]==data["Class"])/len(data))

tree = ID3(train_data,train_data,train_data.columns[:-1])
pprint(tree)

error = test(test_data,tree)
error_train = test(train_data,tree)

print(f"Error_train(f) is: {error_train} \ni.e. {error_train*100}% \nError(f) is:
 ↪ {error} \ni.e. {error*100}%")
```

```
TRAIN DATA
     Clump Thickness  Uniformity of Cell Size  Uniformity of Cell Shape  \
0                  0                        0                         0
1                  0                        0                         0
2                  0                        0                         0
3                  1                        1                         1
4                  0                        0                         0
..               ...                      ...                       ...
554                0                        0                         0
555                0                        0                         0
556                0                        0                         0
557                0                        0                         0
558                0                        0                         0

     Marginal Adhesion  Single Epithelial Cell Size  Bare Nuclei  \
0                    0                            0            0
1                    0                            1            1
2                    0                            0            0
```

```
3                    0                            0            0
4                    0                            0            0
..                 ...                          ...          ...
554                  0                            0            0
555                  0                            0            0
556                  0                            0            0
557                  0                            0            0
558                  0                            0            0

     Bland Chromatin  Normal Nucleoli  Mitoses  Class
0                  0                0        0      0
1                  0                0        0      0
2                  0                0        0      0
3                  0                1        0      0
4                  0                0        0      0
..               ...              ...      ...    ...
554                0                0        0      0
555                0                1        0      0
556                0                0        0      0
557                0                0        0      0
558                0                0        0      0

[559 rows x 10 columns]
TEST DATA
     Clump Thickness  Uniformity of Cell Size  Uniformity of Cell Shape  \
0                  0                        0                         0
1                  0                        0                         0
2                  0                        0                         0
3                  0                        0                         0
4                  0                        0                         0
..               ...                      ...                       ...
135                0                        0                         0
136                0                        0                         0
137                0                        1                         1
138                0                        1                         1
139                0                        1                         1

     Marginal Adhesion  Single Epithelial Cell Size  Bare Nuclei  \
0                    0                            0            0
1                    0                            0            0
2                    0                            0            0
3                    0                            0            0
4                    0                            0            0
..                 ...                          ...          ...
135                  0                            0            0
136                  0                            0            0
137                  0                            1            0
138                  0                            0            0
```

```
139                0                          0            0

     Bland Chromatin  Normal Nucleoli  Mitoses  Class
0                  0                0        0      0
1                  0                0        0      0
2                  0                0        0      0
3                  0                0        0      0
4                  0                0        0      0
..               ...              ...      ...    ...
135                0                0        0      0
136                0                0        0      0
137                1                1        0      1
138                1                1        0      1
139                1                0        0      1

[140 rows x 10 columns]
{'Mitoses': {0: {'Single Epithelial Cell Size': {0.0: {'Marginal Adhesion':
{0.0: {'Uniformity of Cell Size': {0.0: {'Uniformity of Cell Shape': {0.0:
{'Bland Chromatin': {0.0: {'Normal Nucleoli': {0.0: {'Clump Thickness': {0.0:
{'Bare Nuclei': {0: 0.0,
      1: 0.0}},
                                                                1.0: {'Bare
Nuclei': {0: 0.0,
      1: 0.0}}}},
                                      1.0: {'Bare Nuclei': {0: {'Clump
Thickness': {0.0: 0.0,
    1.0: 1.0}},
                                                            1: 1.0}}}},
            1.0: {'Normal Nucleoli': {0.0: {'Clump Thickness': {0.0: {'Bare
Nuclei': {0: 0.0,
      1: 1.0}},
                                                                1.0: {'Bare
Nuclei': {0: 1.0}}}},
                                      1.0: {'Bare Nuclei': {1: {'Clump
Thickness': {0.0: 1.0,
    1.0: 1.0}}}}}}}},
                                                                  1.0: {'Normal
Nucleoli': {0.0: {'Clump Thickness': {0.0: {'Bland Chromatin': {0.0: {'Bare
Nuclei': {0: 0.0,
      1: 1.0}}}},
                                      1.0: {'Bland Chromatin': {0.0: {'Bare
Nuclei': {0: 1.0,
      1: 1.0}},
                                                                1.0: 1.0}}}},
            1.0: 1.0}}}},
                               1.0: {'Bare Nuclei': {0: {'Clump Thickness':
{1.0: {'Uniformity of Cell Shape': {0.0: 1.0,
                                1.0: {'Normal Nucleoli': {1.0: {'Bland
Chromatin': {0.0: 1.0,
    1.0: 1.0}}}}}}}}},
                                                     1: {'Uniformity of Cell
Shape': {0.0: 1.0,
      1.0: {'Clump Thickness': {0.0: {'Bland Chromatin': {0.0: {'Normal
Nucleoli': {0.0: 0.0,
    1.0: 1.0}},
                                                          1.0: 1.0}},
                                1.0: {'Bland Chromatin': {0.0: {'Normal
Nucleoli': {0.0: 1.0,
    1.0: 1.0}},
                                                          1.0: 1.0}}}}}}}}}},
1.0: {'Uniformity of Cell Size': {0.0: {'Uniformity of Cell Shape': {0.0:
{'Bland Chromatin': {0.0: {'Normal Nucleoli': {0.0: {'Clump Thickness': {0.0:
{'Bare Nuclei': {0: 0.0,
      1: 1.0}},
                                                                1.0: 1.0}},
                                      1.0: 1.0}},
            1.0: 1.0}},
                                                                  1.0: 1.0}},
                               1.0: 1.0}}}},
                                                 1.0: {'Normal Nucleoli': {0.0:
{'Bland Chromatin': {0.0: {'Bare Nuclei': {0: {'Uniformity of Cell Shape': {0.0:
{'Clump Thickness': {0.0: {'Uniformity of Cell Size': {0.0: 0.0,
                                                        1.0: 1.0}},
                      1.0: 1.0}},
                                                                            1.0:
1.0}},
                                           1: {'Uniformity of Cell Shape': {0.0:
{'Uniformity of Cell Size': {0.0: {'Marginal Adhesion': {0.0: {'Clump
Thickness': {0.0: 1.0,
    1.0: 1.0}},
                                                          1.0: 1.0}},
                              1.0: 1.0}},
                                                                            1.0:
1.0}}}},
                     1.0: {'Marginal Adhesion': {0.0: {'Uniformity of Cell
Size': {0.0: {'Uniformity of Cell Shape': {0.0: {'Clump Thickness': {0.0: {'Bare
Nuclei': {0: 0.0,
      1: 1.0}},
                                                                1.0: 1.0}}}},
   1.0: 1.0}},
                                                 1.0: 1.0}}}},
                                                                           1.0:
{'Marginal Adhesion': {0.0: {'Bare Nuclei': {0: {'Bland Chromatin': {0.0:
{'Uniformity of Cell Shape': {0.0: {'Uniformity of Cell Size': {0.0: {'Clump
Thickness': {0.0: 0.0,
    1.0: 1.0}},
                                                                       1.0: 1.0}},
                                  1.0: 1.0}},
                                                                                  1.0: 1.0}},
                                                      1: 1.0}},
                                 1.0: {'Clump Thickness': {0.0: {'Uniformity of Cell
Size': {0.0: 1.0,
   1.0: {'Bland Chromatin': {0.0: {'Uniformity of Cell Shape': {0.0: 0.0,
                                                                            1.0: 1.0}},
                                       1.0: 1.0}}}},
                                                           1.0: {'Uniformity of Cell
Shape': {0.0: 1.0,
    1.0: {'Uniformity of Cell Size': {0.0: 1.0,
                                          1.0: {'Bland Chromatin': {1.0: {'Bare
Nuclei': {0: 1.0,
      1: 1.0}}}}}}}}}}}}}}}},
             1: {'Marginal Adhesion': {0.0: {'Single Epithelial Cell Size':
{0.0: {'Uniformity of Cell Size': {0.0: {'Bland Chromatin': {0.0: {'Normal
Nucleoli': {0.0: {'Uniformity of Cell Shape': {0.0: {'Bare Nuclei': {0: {'Clump
Thickness': {0.0: 0.0,
    1.0: 1.0}}}},
                                          1.0: 1.0}}}},
                                                                      1.0: 1.0}},
                                        1.0: 1.0}},
1.0: 1.0}},
                                           1.0: 1.0}}}}
The Prediction accuracy is: 98.57142857142858 %
error 1.4285714285714164%
The Prediction accuracy is: 95.16994633273703 %
error 4.830053667262973%
Error_train(f) is: 0.04830053667262968
i.e. 4.830053667262968%
Error(f) is: 0.014285714285714235
i.e. 1.4285714285714235%
```

RESULTS:

From the above decision tree we can can observe that the relevant features are Bare Nuclei, Uniformity of cell shape, Normal Nuclei, Bland Chromatin and Mitoses which had a high co-relation with the class label (Target value)

Predicted testing accuracy is 98.57% with testing error of 1.42%

Predicted training accuracy is 95.16% with training error of 4.830%

Were you able to train over all features? Trained over all features (Excluding id number)

What kinds of computational tradeoffs did you face, and how did you settle them? Id number trade off (Each entry had a different id number hence was a irrelevant feature in the data) Therefore, Dropped the column id number from the dataset

MODEL VALIDATION

```
[43]: #ID3 For Tree Pruning
      def ID3(d,s,data,originaldata,features,target_attribute_name="Class",
              parent_node_class=None):

          #If all target_values have the same value, Then return this value
          if len(np.unique(data[target_attribute_name])) <= 1:
              return np.unique(data[target_attribute_name])[0]

          #If the dataset is empty
          elif len(data) == 0:
              return np.unique(originaldata[target_attribute_name])[np.argmax(np.
       ↪unique(originaldata[target_attribute_name],
                                                                                    ␣
       ↪return_counts=True)[1])]

          #If the feature space is empty i.e. Attribute list == 0
          elif len(features) == 0:
              return parent_node_class

          #If none of the above condition holds true grow the tree and look for its␣
       ↪subtree

          else:
              parent_node_class = np.unique(data[target_attribute_name])[np.argmax(np.
       ↪unique(data[target_attribute_name],
                                                                                    ␣
       ↪return_counts=True)[1])]

          #Select the feature which best splits the dataset having the heightest info␣
       ↪gain
          item_values = [InfoGain(data,feature,target_attribute_name)for feature in␣
       ↪features] #Return the infgain values
          best_feature_index = np.argmax(item_values)
          best_feature = features[best_feature_index]

          #Create the tree structure; Tree in the form of dictionary
          tree = {best_feature:{}}

          #Remove the feature with the best info gain once added to the tree
          features = [i for i in features if i != best_feature]

          #Grow the tree branch under the root node
          for value in np.unique(data[best_feature]):
              value = value
              if (d>depth):
                  if len(np.unique(data[target_attribute_name])) <=1:
                          tree[best_feature][value]  = np.
       ↪unique(data[target_attribute_name])[0]
                  else:
                          tree[best_feature][value] = np.argmax(np.
       ↪unique(data[target_attribute_name],return_counts=True)[1])
              else:
                  d = d+1
                  sub_data = data.where(data[best_feature]==value).dropna()
                  if len(sub_data) <=s:
                      if len(np.unique(data[target_attribute_name])) <=1:
                          tree[best_feature][value]  = np.
       ↪unique(data[target_attribute_name])[0]
                      else:
                          tree[best_feature][value] = np.argmax(np.
       ↪unique(data[target_attribute_name],return_counts=True)[1])
                  else:
                      #call the ID3 algotirthm
                      subtree =␣
       ↪ID3(d,s,sub_data,dataset,features,target_attribute_name,parent_node_class)
                      #Add the subtree
                      tree[best_feature][value] = subtree
          return(tree)

      def count_irrelevant_feature(features):
          feature_num = []
          count = 0
          for x in features:
              feature_num.append(int(x[1:]))
          for x in np.unique(feature_num):
              if x>10:
                  count+=1
          return count


      d = 0
      s = 10
      depth =5
      m =699

      dataset = df
      train_data = split(dataset)[0]
      test_data = split(dataset)[1]
      tree = ID3(d,s,train_data,train_data,train_data.columns[:-1])
      pprint(tree)
      error = test(test_data,tree)
      error_train = test(train_data,tree)
```

```
print(f"Error_train(f) is: {error_train} \ni.e. {error_train*100}% \nError(f) is:
 ↪ {error} \ni.e. {error*100}%")
```

```
{'Bare Nuclei': {0: {'Clump Thickness': {0.0: {'Uniformity of Cell Size': {0.0:
{'Marginal Adhesion': {0.0: {'Normal Nucleoli': {0.0: {'Bland Chromatin': {0.0:
{'Uniformity of Cell Shape': {0.0: 0,
                              1.0: 0}},
                                                                          1.0:
0}},
                                                   1.0: 0}},
                       1.0: 0}},
                                                                          1.0:
0}},
                                         1.0: {'Uniformity of Cell Size': {0.0:
{'Normal Nucleoli': {0.0: {'Single Epithelial Cell Size': {0.0: {'Mitoses':
{0.0: 0,
1.0: 0}},
                                                           1.0: 0}},
                     1.0: 1}},
                                                                          1.0:
{'Uniformity of Cell Shape': {0.0: 1,
                              1.0: 1}}}}}},
                 1: {'Marginal Adhesion': {0.0: {'Bland Chromatin': {0.0:
{'Normal Nucleoli': {0.0: {'Clump Thickness': {0.0: 1,
                                         1.0: 1}},
               1.0: 1.0}},
                                                                    1.0:
{'Uniformity of Cell Shape': {0.0: {'Normal Nucleoli': {0.0: 1,
                                                  1.0: 1}},
                        1.0: 1}}}},
                                           1.0: 1.0}}}}
The Prediction accuracy is: 92.85714285714286 %
error 7.142857142857139%
The Prediction accuracy is: 94.27549194991055 %
error 5.7245080500894545%
Error_train(f) is: 0.057245080500894496
i.e. 5.724508050089449%
Error(f) is: 0.0714285714285714
i.e. 7.14285714285714%
```

RESULTS: From the above decision tree we can can observe that the most relevant features are Bare Nuclei, Uniformity of cell shape, Uniformity of Size, Clump Thickness and Bland Chromatin which had a high co-relation with the class label (Target value)

Predicted testing accuracy is 92.85% with testing error of 7.14% Predicted training accuracy is 94.27% with training error of 5.72%

How did you try to avoid overfitting the data?

By adding a Depth limit of 5 to the tree and Sample size limit of 10, We can simply say that by pruning the tree I have tried to avoid overfitting the data.

How did you handle the modest (in ML terms) size of the data set?

The size of the data is 699 samples which is not that huge as compared to the datasets that we worked on in the assignments. But the steps then to handle the data are pre-processing the data before fitting the model to it. and later pruning the tree to avoid over fitting.

EVALUATE YOUR MODEL:

Where is your model particularly successful, where does it lack?

My model give a good testing prediction rate before prunning. and the test errors is also very small. While, It does not give a good result after prunning

Does it need a certain amount of features in order to interpolate well?

No it is not necessary to have a certain amount of features in order to interpolate well, because the class label (Target column takes up only 2 values). Hence, to build a decision tree on that does not require much column and can work on any amount of features equal to or greater than 2 (all the features in the dataset of cancer are having a good weightage: almost every feature can handle the data if required)

Are there some features it is really good at predicting and some it is really poor at predicting? Yes there are some relevant and irrelevant features as seen in the decision tree results.

Why do you think that is?

Because 1 feature may have a greater impact on the output than the other feature.

ANALYSE THE DATA:

How does your model stack up against the basic completion agent?

The model performance is above average but the error of the basic completion agent is less than that of the model. Hence, the working nature of the basic completion agent is better.

What features were particularly valuable in predicting/interpolating?

Uniformity of cell size and Uniformity of cell shape including bare nuclei

What features weren't particularly useful at all? id number was not useful at all.. (Marginal adhesion and Mitoses were less useful)

Were there any features about the researcher/experimental environment that were particularly relevant to predicting answers - and if so, what conclusions can you draw about the replicability of those effects?

Uniformity of cell size and Uniformity of cell shape are strongly corelated a high value in a feature leads to a high value in the other feature.

Also, If the Uniformity of cell size and shape is small (around 1) there is a high chance that the output class is 2 (i.e. benign)(From the bar graph drawn earlier)

[46]:
```
shuffle_df = df.sample(frac=1)
train_size = int(0.8 * len(df))
```

```
train_set = shuffle_df[:train_size]
test_set = shuffle_df[train_size:]
```

[47]:
```
print(train_set.shape)
print(test_set.shape)
```

```
(559, 10)
(140, 10)
```

[48]:
```
X_train = np.array(train_set.drop(['Class'], axis=1),'float32')
y_train = np.array(train_set['Class'],'float32')
X_test = np.array(test_set.drop(['Class'], axis=1),'float32')
y_test = np.array(test_set['Class'],'float32')
```

[49]:
```
def linear_regression(X, y, lamb):
    w = np.dot( np.linalg.inv(np.dot(X.T, X) + lamb*np.eye(X.shape[1])) , np.
 ↪dot(X.T,y))
    return w
w = linear_regression(X_train, y_train, 0.01)
print(w)
error = np.average(np.square((np.subtract(y_test,np.dot(w, X_test.T) ))))
error
```

```
[ 0.28279077  0.10420654  0.1478538   0.14076259  0.13200904  0.38035536
  0.09504456  0.13284301 -0.04232733]
```

[49]:
```
0.04430478691567913
```

Approximately equal to that of ID3 algorithm

GENERATE DATA

[69]:
```
df = pd.read_csv ('bcw.data')
df_null_BareNuclei = df[df['Bare Nuclei'].isnull()]
df_BareNuclei = df.copy()
df_BareNuclei.head()
```

[69]:
```
   id number  Clump Thickness  Uniformity of Cell Size  \
0    1000025                5                        1
1    1002945                5                        4
2    1015425                3                        1
3    1016277                6                        8
4    1017023                4                        1

   Uniformity of Cell Shape  Marginal Adhesion  Single Epithelial Cell Size  \
0                         1                  1                            2
1                         4                  5                            7
2                         1                  1                            2
```

```
3                        8                  1                            3
4                        1                  3                            2

   Bare Nuclei  Bland Chromatin  Normal Nucleoli  Mitoses  Class
0          1.0                3                1        1      2
1         10.0                3                2        1      2
2          2.0                3                1        1      2
3          4.0                3                7        1      2
4          1.0                3                1        1      2
```

[70]:
```
## Removing id number as already shown above
df_BareNuclei.drop(['id number'], axis=1, inplace=True)
df_BareNuclei.head(5)
```

[70]:
```
   Clump Thickness  Uniformity of Cell Size  Uniformity of Cell Shape  \
0                5                        1                         1
1                5                        4                         4
2                3                        1                         1
3                6                        8                         8
4                4                        1                         1

   Marginal Adhesion  Single Epithelial Cell Size  Bare Nuclei  \
0                  1                            2          1.0
1                  5                            7         10.0
2                  1                            2          2.0
3                  1                            3          4.0
4                  3                            2          1.0

   Bland Chromatin  Normal Nucleoli  Mitoses  Class
0                3                1        1      2
1                3                2        1      2
2                3                1        1      2
3                3                7        1      2
4                3                1        1      2
```

[74]:
```
col = df_BareNuclei["Bare Nuclei"].isnull()
BareNuclei_train = df_BareNuclei[~col].copy()
BareNuclei_test = df_BareNuclei[col].copy()

print(BareNuclei_train.shape)
print(BareNuclei_test.shape)
```

```
(683, 10)
(16, 10)
```

[75]:
```
X_BareNuclei = BareNuclei_train.drop(['Bare Nuclei'], axis=1)
y_BareNuclei = BareNuclei_train['Bare Nuclei']
```

```
BareNuclei_test.drop(['Bare Nuclei'], axis=1, inplace=True)
print(X_BareNuclei.shape)
print(y_BareNuclei.shape)
print(BareNuclei_test.shape)
```

```
(683, 9)
(683,)
(16, 9)
```

[76]:
```
w_BareNuclei = linear_regression(X_BareNuclei, y_BareNuclei, 0)
```

[77]:
```
pred_BareNuclei = np.rint(np.dot(w_BareNuclei,BareNuclei_test.T))
pred_BareNuclei
```

[77]:
```
array([5., 7., 2., 2., 2., 2., 3., 2., 3., 7., 2., 3., 5., 2., 2., 2.])
```

[84]:
```
BareNuclei_test['Bare Nuclei'] = pred_BareNuclei
BareNuclei_test.head(10)
```

[84]:
```
     Clump Thickness  Uniformity of Cell Size  Uniformity of Cell Shape  \
23                 8                        4                         5
40                 6                        6                         6
139                1                        1                         1
145                1                        1                         3
158                1                        1                         2
164                5                        1                         1
235                3                        1                         4
249                3                        1                         1
275                3                        1                         3
292                8                        8                         8

     Marginal Adhesion  Single Epithelial Cell Size  Bland Chromatin  \
23                   1                            2                7
40                   9                            6                7
139                  1                            1                2
145                  1                            2                2
158                  1                            3                1
164                  1                            2                3
235                  1                            2                3
249                  1                            2                3
275                  1                            2                2
292                  1                            2                6

     Normal Nucleoli  Mitoses  Class  Bare Nuclei
23                 3        1      4          5.0
40                 8        1      2          7.0
139                1        1      2          2.0
```

```
145                   1         1      2           2.0
158                   1         1      2           2.0
164                   1         1      2           2.0
235                   1         1      2           3.0
249                   1         1      2           2.0
275                   1         1      2           3.0
292                  10         1      4           7.0
```

[85]:
```
X_val = BareNuclei_test.drop(['Class'], axis=1)
y_val = BareNuclei_test['Class']
```

[87]:
```
error = np.average(np.square((np.subtract(y_val,np.dot(w, X_val.T) ))))/20
error
```

[87]:
```
0.2919098370415426
```

[88]:
```
np.unique(pred_BareNuclei, return_counts=True)
```

[88]:
```
(array([2., 3., 5., 7.]), array([9, 3, 2, 2]))
```

RESULTS:

Use your system to try to generate realistic data, and compare your generated data to the real data. How good does it look?

The data generated by my system has an high error rate than that of the real data. The data looks average (Not too good not too bad)

What does it mean for it to 'look good'?

To have a relalistic data which will have the values surrounded around the mean and variance of that feature, and give a low testing and training error (high prediction when fitted of the model)

[ ]:
```
ADDITIONAL ALGORITHMS
```

[ ]:
```
K NEAREST NEIGHBOR ALGORITHM
```

[93]:
```
df = pd.read_csv ('bw.data')
df
```

[93]:
```
     1000025  5   1  1.1  1.2  2   1.3   3  1.4  1.5  2.1
0    1002945  5   4    4    5  7  10.0   3    2    1    2
1    1015425  3   1    1    1  2   2.0   3    1    1    2
2    1016277  6   8    8    1  3   4.0   3    7    1    2
3    1017023  4   1    1    3  2   1.0   3    1    1    2
4    1017122  8  10   10    8  7  10.0   9    7    1    4
..       ... ..  ..  ...  ... ..   ...  ..  ...  ...  ...
693   776715  3   1    1    1  3   2.0   1    1    1    2
694   841769  2   1    1    1  2   1.0   1    1    1    2
```

```
695   888820  5  10    10    3  7   3.0   8   10    2    4
696   897471  4   8     6    4  3   4.0  10    6    1    4
697   897471  4   8     8    5  4   5.0  10    4    1    4

[698 rows x 11 columns]
```

[94]:
```
df.replace({2.1:{2:1, 4:0}}, inplace=True)
```

[97]:
```
from random import seed
from random import randrange
from csv import reader
from math import sqrt

# Load a CSV file
def load_csv(filename):
    dataset = list()
    with open('data.csv', 'r') as file:
        csv_reader = reader(file)
        for row in csv_reader:
            if not row:
                continue
            dataset.append(row)
    return dataset

# Convert string column to float
def str_column_to_float(dataset, column):
    for row in dataset:
        row[column] = float(row[column].strip())

# Convert string column to integer
def str_column_to_int(dataset, column):
    class_values = [row[column] for row in dataset]
    unique = set(class_values)
    lookup = dict()
    for i, value in enumerate(unique):
        lookup[value] = i
    for row in dataset:
        row[column] = lookup[row[column]]
    return lookup

# Find the min and max values for each column
def dataset_minmax(dataset):
    minmax = list()
    for i in range(len(dataset[0])):
        col_values = [row[i] for row in dataset]
        value_min = min(col_values)
        value_max = max(col_values)
        minmax.append([value_min, value_max])
    return minmax

# Rescale dataset columns to the range 0-1
def normalize_dataset(dataset, minmax):
    for row in dataset:
        for i in range(len(row)):
            row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] - minmax[i][0])

# Split a dataset into k folds
def cross_validation_split(dataset, n_folds):
    dataset_split = list()
    dataset_copy = list(dataset)
    fold_size = int(len(dataset) / n_folds)
    for _ in range(n_folds):
        fold = list()
        while len(fold) < fold_size:
            index = randrange(len(dataset_copy))
            fold.append(dataset_copy.pop(index))
        dataset_split.append(fold)
    return dataset_split

# Calculate accuracy percentage
def accuracy_metric(actual, predicted):
    correct = 0
    for i in range(len(actual)):
        if actual[i] == predicted[i]:
            correct += 1
    return correct / float(len(actual)) * 100.0

# Evaluate an algorithm using a cross validation split
def evaluate_algorithm(dataset, algorithm, n_folds, *args):
    folds = cross_validation_split(dataset, n_folds)
    scores = list()
    for fold in folds:
        train_set = list(folds)
        train_set.remove(fold)
        train_set = sum(train_set, [])
        test_set = list()
        for row in fold:
            row_copy = list(row)
            test_set.append(row_copy)
            row_copy[-1] = None
        predicted = algorithm(train_set, test_set, *args)
        actual = [row[-1] for row in fold]
        accuracy = accuracy_metric(actual, predicted)
        scores.append(accuracy)
```

```python
    return scores

# Calculate the Euclidean distance between two vectors
def euclidean_distance(row1, row2):
    distance = 0.0
    for i in range(len(row1)-1):
        distance += (row1[i] - row2[i])**2
    return sqrt(distance)

# Locate the most similar neighbors
def get_neighbors(train, test_row, num_neighbors):
    distances = list()
    for train_row in train:
        dist = euclidean_distance(test_row, train_row)
        distances.append((train_row, dist))
    distances.sort(key=lambda tup: tup[1])
    neighbors = list()
    for i in range(num_neighbors):
        neighbors.append(distances[i][0])
    return neighbors

# Make a prediction with neighbors
def predict_classification(train, test_row, num_neighbors):
    neighbors = get_neighbors(train, test_row, num_neighbors)
    output_values = [row[-1] for row in neighbors]
    prediction = max(set(output_values), key=output_values.count)
    return prediction

# kNN Algorithm
def k_nearest_neighbors(train, test, num_neighbors):
    predictions = list()
    for row in test:
        output = predict_classification(train, row, num_neighbors)
        predictions.append(output)
    #print(predictions)
    return(predictions)

# Test the kNN on the Iris Flowers dataset
seed(1)
filename = 'iris.csv'
dataset = load_csv(filename)
for i in range(len(dataset[0])-1):
    str_column_to_float(dataset, i)
# convert class column to integers
str_column_to_int(dataset, len(dataset[0])-1)
# evaluate algorithm
n_folds = 5
```

```
num_neighbors = 5
scores = evaluate_algorithm(dataset, k_nearest_neighbors, n_folds, num_neighbors)
print('Scores: %s' % scores)
print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))
```

```
Scores: [60.431654676258994, 61.15107913669065, 60.431654676258994,
64.74820143884892, 61.87050359712231]
Mean Accuracy: 61.727%
```

Accuracy of 61.727% which is far less than that of ID3 accuracy

NEURAL NET WITH ONLY 1 HIDDEN LAYER

[99]:
```
# Shallow Neural net with only one hidden layer

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

np.random.seed(1)


def main():
    train_x = pd.read_csv("cancer_data.csv")
    train_x = np.array(train_x)
    train_y = pd.read_csv("cancer_data_y.csv")
    train_y = np.array(train_y)

    d = model(train_x.T, train_y.T, n_h=20, num_iters=50000, alpha=0.0002,␣
 ↪print_cost=True)


def sigmoid(z):
    s = 1 / (1 + np.exp(-z))
    return s


def layers(X, Y):
    """
    :param X:
    :param Y:
    :return:
    """
    n_x = X.shape[0]
    n_y = Y.shape[0]
    return n_x, n_y
```

```python
def initialize(n_x, n_h, n_y):
    """
    :param n_x:
    :param n_h:
    :param n_y:
    :return:
    """
    np.random.seed(2)
    W1 = np.random.randn(n_h, n_x) * 0.01
    b1 = np.random.rand(n_h, 1)
    W2 = np.random.rand(n_y, n_h)
    b2 = np.random.rand(n_y, 1)
    parameters = {"W1": W1,
                  "b1": b1,
                  "W2": W2,
                  "b2": b2}

    return parameters


def forward_prop(X, parameters):
    W1 = parameters['W1']
    b1 = parameters['b1']
    W2 = parameters['W2']
    b2 = parameters['b2']

    Z1 = np.dot(W1, X) + b1
    A1 = np.tanh(Z1)
    Z2 = np.dot(W2, A1) + b2
    A2 = sigmoid(Z2)

    cache = {"Z1": Z1,
             "A1": A1,
             "Z2": Z2,
             "A2": A2}

    return A2, cache


def compute_cost(A2, Y, parameters):
    m = Y.shape[1]
    W1 = parameters['W1']
    W2 = parameters['W2']
    logprobs = np.multiply(np.log(A2), Y) + np.multiply(np.log(1 - A2), (1 - Y))
    cost = - np.sum(logprobs) / m
    cost = np.squeeze(cost)
    return cost


def back_prop(parameters, cache, X, Y):
    m = Y.shape[1]
    W1 = parameters['W1']
    W2 = parameters['W2']
    A1 = cache['A1']
    A2 = cache['A2']

    dZ2 = A2 - Y
    dW2 = (1 / m) * np.dot(dZ2, A1.T)
    db2 = (1 / m) * np.sum(dZ2, axis=1, keepdims=True)

    dZ1 = np.multiply(np.dot(W2.T, dZ2), 1 - np.square(A1))
    dW1 = (1 / m) * np.dot(dZ1, X.T)
    db1 = (1 / m) * np.sum(dZ1, axis=1, keepdims=True)

    grads = {"dW1": dW1,
             "db1": db1,
             "dW2": dW2,
             "db2": db2}

    return grads


def update_params(parameters, grads, alpha):
    W1 = parameters['W1']
    b1 = parameters['b1']
    W2 = parameters['W2']
    b2 = parameters['b2']

    dW1 = grads['dW1']
    db1 = grads['db1']
    dW2 = grads['dW2']
    db2 = grads['db2']

    W1 = W1 - alpha * dW1
    b1 = b1 - alpha * db1
    W2 = W2 - alpha * dW2
    b2 = b2 - alpha * db2

    parameters = {"W1": W1,
                  "b1": b1,
                  "W2": W2,
                  "b2": b2}
    return parameters
```

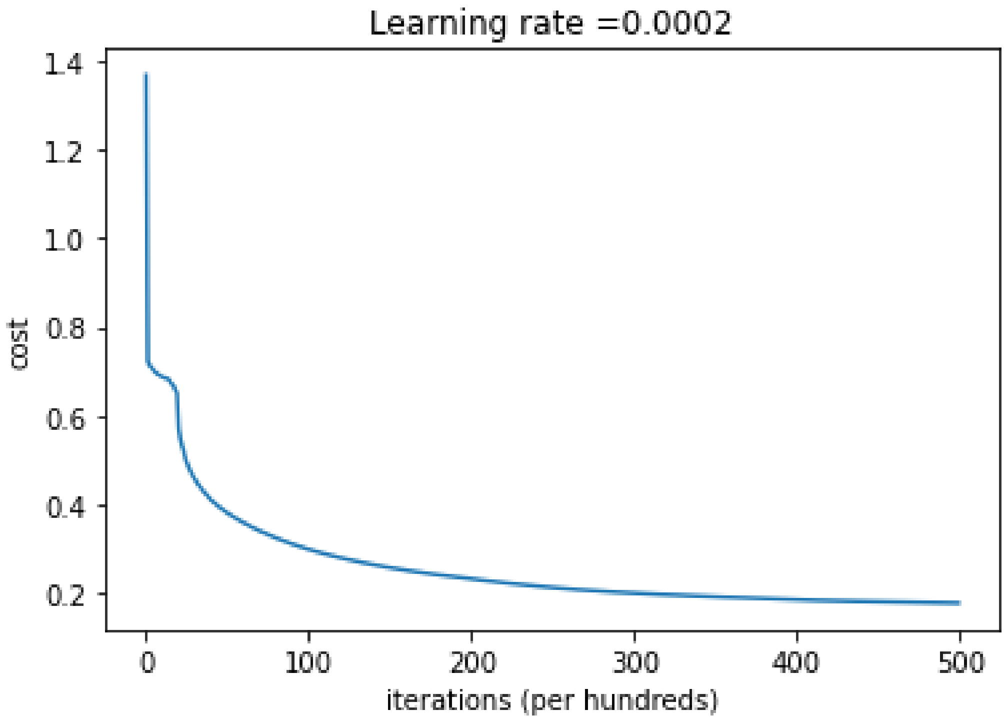
```
def model(X, Y, n_h, num_iters, alpha, print_cost):
    np.random.seed(3)
    n_x = layers(X, Y)[0]
    n_y = layers(X, Y)[1]

    parameters = initialize(n_x, n_h, n_y)
    W1 = parameters['W1']
    b1 = parameters['b1']
    W2 = parameters['W2']
    b2 = parameters['b2']

    costs = []
    for i in range(0, num_iters):

        A2, cache = forward_prop(X, parameters)

        cost = compute_cost(A2, Y, parameters)
        grads = back_prop(parameters, cache, X, Y)
        if (i > 20000):
            alpha1 = (20000 / i) * alpha
            parameters = update_params(parameters, grads, alpha1)
        else:
            parameters = update_params(parameters, grads, alpha)

        if i % 100 == 0:
            costs.append(cost)
        if print_cost and i % 1000 == 0:
            print("Cost after iteration %i: %f" % (i, cost))
            if i <= 20000:
                print("Learning rate after iteration %i: %f" % (i, alpha))
            else:
                print("Learning rate after iteration %i: %f" % (i, alpha1))

    X_test = pd.read_csv("test_cancer_data.csv")
    X_test = np.array(X_test)
    X_test = X_test.T
    Y_test = pd.read_csv("test_cancer_data_y.csv")
    Y_test = np.array(Y_test)
    Y_test = Y_test.T

    predictions = predict(parameters, X)
    print('Accuracy on training set: %.2f' % float(
        (np.dot(Y, predictions.T) + np.dot(1 - Y, 1 - predictions.T)) / float(Y.
 ↪size) * 100) + '%')
    truePositive = 0
    trueNegative = 0
    falseNegative = 0
    falsePositive = 0
    predList = predictions.tolist()
    tlist = Y.tolist()

    array_length = len(predList[0])
    for i in range(array_length):
        if predList[0][i] == 1 and tlist[0][i] == 1:
            truePositive += 1
        elif predList[0][i] == 0 and tlist[0][i] == 0:
            trueNegative += 1
        elif predList[0][i] == 0 and tlist[0][i] == 1:
            falseNegative += 1
        elif predList[0][i] == 1 and tlist[0][i] == 0 :
            falsePositive += 1
        else:
            print(predList[0][i])
            print(tlist[0][i])

    tpr = truePositive / (truePositive + falseNegative) * 100
    fpr = falsePositive / (falsePositive + trueNegative) * 100
    precision = truePositive / (truePositive + falsePositive) * 100
    print("On training set:\nTrue Positive:  ", truePositive)
    print("True Negative:  ", trueNegative)
    print("False Negative:  ", falseNegative)
    print("False Positive:  ", falsePositive)
    print("True Positive Rate / Recall: %.2f" % tpr+str('%'))
    print("Precision: %.2f" %precision+str('%'))
    print("False Positive Rate / Fallout: %.2f" %fpr+str('%'))

    predictions = predict(parameters, X_test)
    print('Accuracy on test set: %.2f' % float(
        (np.dot(Y_test, predictions.T) + np.dot(1 - Y_test, 1 - predictions.T)) /
↪ float(Y_test.size) * 100) + '%')
    truePositive = 0
    trueNegative = 0
    falseNegative = 0
    falsePositive = 0
    predList = predictions.tolist()
    tlist = Y_test.tolist()

    assert (len(predictions[0])== len(tlist[0]))
    array_length = len(predList[0])
    for i in range(array_length):
        if predList[0][i] == 1 and tlist[0][i] == 1:
            truePositive += 1
        elif predList[0][i] == 0 and tlist[0][i] == 0:
            trueNegative += 1
        elif predList[0][i] == 0 and tlist[0][i] == 1:
            falseNegative += 1
        elif predList[0][i] == 1 and tlist[0][i] == 0 :
            falsePositive += 1
        else:
            print(predList[0][i])
            print(tlist[0][i])

    tpr = truePositive / (truePositive + falseNegative) * 100
    fpr = falsePositive / (falsePositive + trueNegative) * 100
    precision = truePositive / (truePositive + falsePositive) * 100
    print("On Test set:\nTrue Positive:  ", truePositive)
    print("True Negative:  ", trueNegative)
    print("False Negative:  ", falseNegative)
    print("False Positive:  ", falsePositive)
    print("True Positive Rate / Recall: %.2f" % tpr+str('%'))
    print("Precision: %.2f" %precision+str('%'))
    print("False Positive Rate / Fallout: %.2f" %fpr+str('%'))

    plt.plot(costs)
    plt.ylabel('cost')
    plt.xlabel('iterations (per hundreds)')
    plt.title("Learning rate =" + str(alpha))
    plt.show()

    return parameters


def predict(parameters, X):
    A2, cache = forward_prop(X, parameters)
    predictions = np.round(A2)

    return predictions


main()
```

```
Cost after iteration 0: 1.368816
Learning rate after iteration 0: 0.000200
Cost after iteration 1000: 0.688126
Learning rate after iteration 1000: 0.000200
Cost after iteration 2000: 0.572371
Learning rate after iteration 2000: 0.000200
Cost after iteration 3000: 0.456880
Learning rate after iteration 3000: 0.000200
```

```
Cost after iteration 4000: 0.410189
Learning rate after iteration 4000: 0.000200
Cost after iteration 5000: 0.380970
Learning rate after iteration 5000: 0.000200
Cost after iteration 6000: 0.358918
Learning rate after iteration 6000: 0.000200
Cost after iteration 7000: 0.340165
Learning rate after iteration 7000: 0.000200
Cost after iteration 8000: 0.324504
Learning rate after iteration 8000: 0.000200
Cost after iteration 9000: 0.310779
Learning rate after iteration 9000: 0.000200
Cost after iteration 10000: 0.298861
Learning rate after iteration 10000: 0.000200
Cost after iteration 11000: 0.288570
Learning rate after iteration 11000: 0.000200
Cost after iteration 12000: 0.279487
Learning rate after iteration 12000: 0.000200
Cost after iteration 13000: 0.271351
Learning rate after iteration 13000: 0.000200
Cost after iteration 14000: 0.264014
Learning rate after iteration 14000: 0.000200
Cost after iteration 15000: 0.257393
Learning rate after iteration 15000: 0.000200
Cost after iteration 16000: 0.251425
Learning rate after iteration 16000: 0.000200
Cost after iteration 17000: 0.246047
Learning rate after iteration 17000: 0.000200
Cost after iteration 18000: 0.241176
Learning rate after iteration 18000: 0.000200
Cost after iteration 19000: 0.236722
Learning rate after iteration 19000: 0.000200
Cost after iteration 20000: 0.232629
Learning rate after iteration 20000: 0.000200
Cost after iteration 21000: 0.227717
Learning rate after iteration 21000: 0.000190
Cost after iteration 22000: 0.223480
Learning rate after iteration 22000: 0.000182
Cost after iteration 23000: 0.219584
Learning rate after iteration 23000: 0.000174
Cost after iteration 24000: 0.215937
Learning rate after iteration 24000: 0.000167
Cost after iteration 25000: 0.212544
Learning rate after iteration 25000: 0.000160
Cost after iteration 26000: 0.209475
Learning rate after iteration 26000: 0.000154
Cost after iteration 27000: 0.206740
Learning rate after iteration 27000: 0.000148
```

```
Cost after iteration 28000: 0.204292
Learning rate after iteration 28000: 0.000143
Cost after iteration 29000: 0.202079
Learning rate after iteration 29000: 0.000138
Cost after iteration 30000: 0.200059
Learning rate after iteration 30000: 0.000133
Cost after iteration 31000: 0.198197
Learning rate after iteration 31000: 0.000129
Cost after iteration 32000: 0.196467
Learning rate after iteration 32000: 0.000125
Cost after iteration 33000: 0.194846
Learning rate after iteration 33000: 0.000121
Cost after iteration 34000: 0.193315
Learning rate after iteration 34000: 0.000118
Cost after iteration 35000: 0.191860
Learning rate after iteration 35000: 0.000114
Cost after iteration 36000: 0.190464
Learning rate after iteration 36000: 0.000111
Cost after iteration 37000: 0.189118
Learning rate after iteration 37000: 0.000108
Cost after iteration 38000: 0.187811
Learning rate after iteration 38000: 0.000105
Cost after iteration 39000: 0.186536
Learning rate after iteration 39000: 0.000103
Cost after iteration 40000: 0.185286
Learning rate after iteration 40000: 0.000100
Cost after iteration 41000: 0.184063
Learning rate after iteration 41000: 0.000098
Cost after iteration 42000: 0.183089
Learning rate after iteration 42000: 0.000095
Cost after iteration 43000: 0.182283
Learning rate after iteration 43000: 0.000093
Cost after iteration 44000: 0.181516
Learning rate after iteration 44000: 0.000091
Cost after iteration 45000: 0.180786
Learning rate after iteration 45000: 0.000089
Cost after iteration 46000: 0.180088
Learning rate after iteration 46000: 0.000087
Cost after iteration 47000: 0.179421
Learning rate after iteration 47000: 0.000085
Cost after iteration 48000: 0.178782
Learning rate after iteration 48000: 0.000083
Cost after iteration 49000: 0.178169
Learning rate after iteration 49000: 0.000082
Accuracy on training set: 93.78%
On training set:
True Positive:   162
True Negative:   230
```

```
False Negative:   14
False Positive:   12
True Positive Rate / Recall: 92.05%
Precision: 93.10%
False Positive Rate / Fallout: 4.96%
Accuracy on test set: 89.93%
On Test set:
True Positive:   32
True Negative:   102
False Negative:   3
False Positive:   12
True Positive Rate / Recall: 91.43%
Precision: 72.73%
False Positive Rate / Fallout: 10.53%
```



RESULT:

Accuracy on training set 93.78% Accuracy on Test set 89.93%

the decrease in cost is notable and the results are close to the ID3 algorithm

The dataset works good on Neural network, ID3 model and Linear Regression model.

THANK YOU FOR THIS SEMESTER!! HAPPY HOLIDAYS

```
[ ]:
```