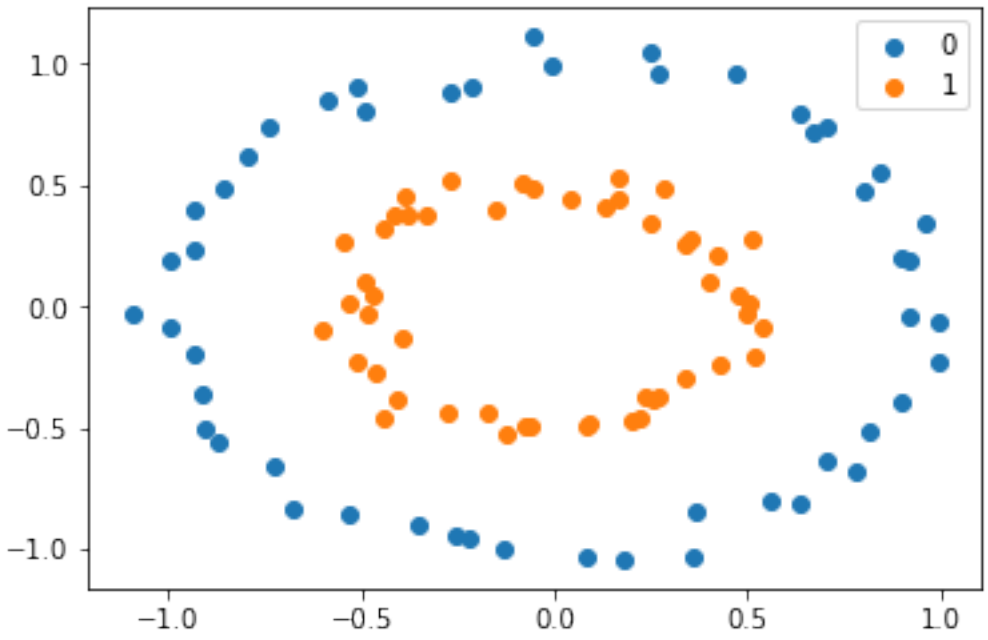
Random Forest Challenge

September 10, 2019

In [1]:
```
import numpy as np # to build the algorithm
import matplotlib.pyplot as plt # to visualize
from sklearn.datasets import make_circles # to generate a dataset
```

In [2]:
```
# Generate a dataset
X, y = make_circles(n_samples=100, noise=0.05, factor = 0.5)
plt.scatter(X[:,0][y==0], X[:,1][y==0], label=0)
plt.scatter(X[:,0][y==1], X[:,1][y==1], label=1)
plt.legend()
```

Out[2]:
```
<matplotlib.legend.Legend at 0x1186c8f28>
```



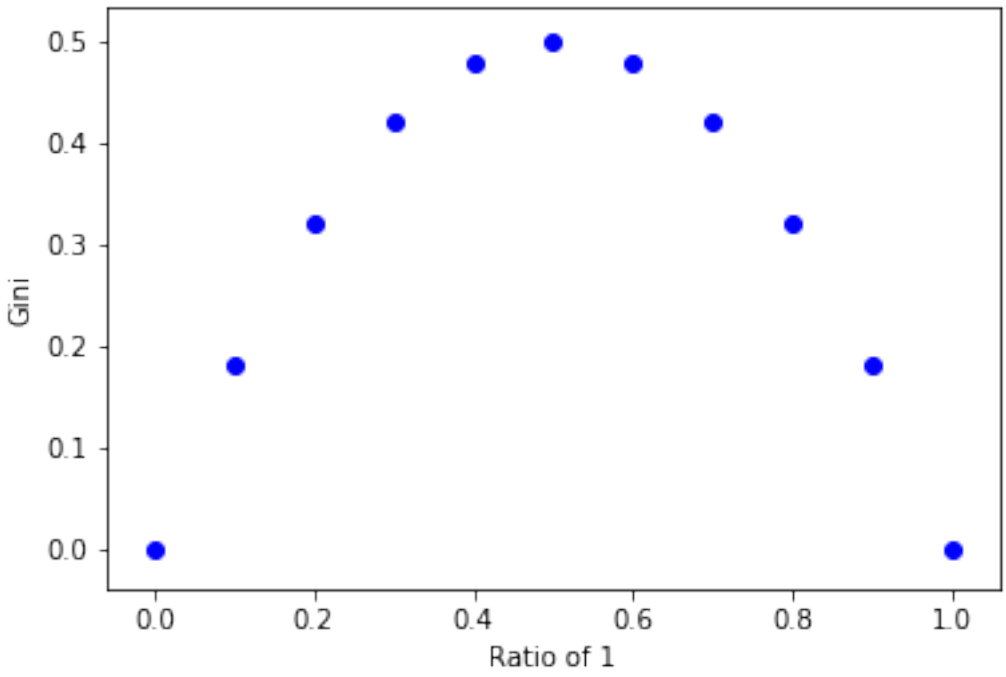
In [3]:
```
def gini_calculator(y):
    '''
    Calculates the gini impurity of a group
```

```
        y contains all the labels ie 1001110...
        gini = 1 - p1^2 - p0^2
        '''

        return gini
```

In [4]:
```
# Sanity check 1
# Plot gini vs ratio of 1
for i in range(11):
    num_ones = i
    num_zeros = 10-i
    prop = num_ones/10
    gini = gini_calculator(np.concatenate((np.ones(num_ones), np.zeros(num_zeros))))
    plt.scatter(prop, gini, color='b')
plt.xlabel('Ratio of 1')
plt.ylabel('Gini')
```

Out[4]: Text(0, 0.5, 'Gini')



In [5]:
```
def gini_of_a_split(y1, y2):
    '''
    Weighted average gini
    for y1 and y2
    '''
    g1 = gini_calculator(y1)
```

```
        w1 = len(y1)/(len(y1)+len(y2))
        g2 = gini_calculator(y2)
        w2 = len(y2)/(len(y1)+len(y2))
        avg_gini = g1*w1 + g2*w2
        return avg_gini
```

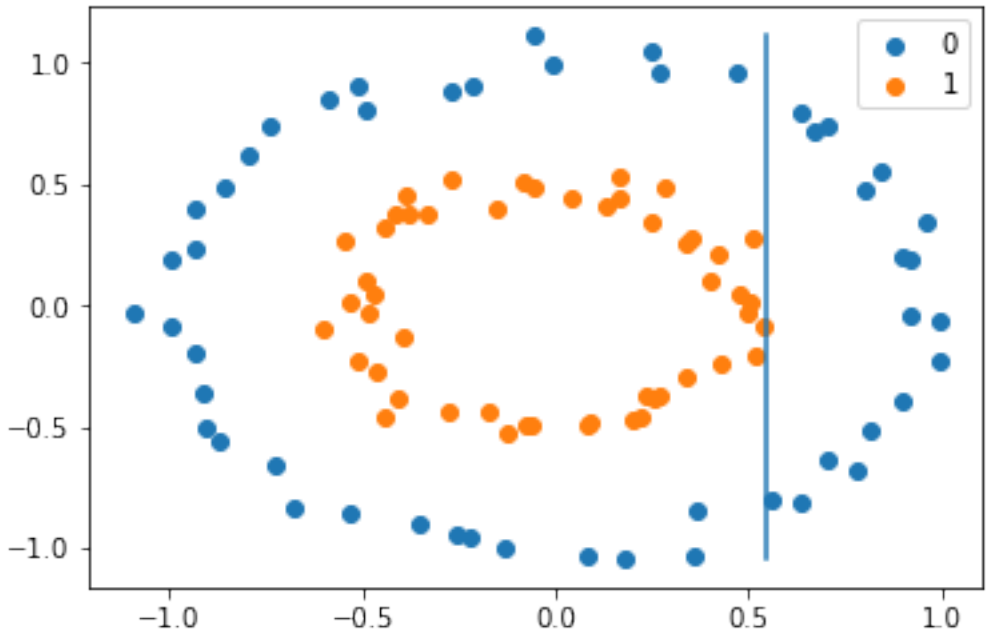
In [6]:
```
def split_finder(X, y):
    '''
    Finds the best split
    in terms of feature and its value
    that minimizes the average gini

    return best feature index and value
    '''

    return best_split
```

In [7]:
```
# Sanity check 2
# Visualize the first split
best_split = split_finder(X, y)
plt.scatter(X[:,0][y==0], X[:,1][y==0], label=0)
plt.scatter(X[:,0][y==1], X[:,1][y==1], label=1)
plt.legend()

if best_split[0]==0:
    plt.plot([best_split[1],best_split[1]],[min(X[:,1]),max(X[:,1])])
elif best_split[0]==1:
    plt.plot([min(X[:,0]),max(X[:,0])],[best_split[1],best_split[1]])
```



```
In [8]: def splitter(X, y):
            '''
            Given X and y
            calculates the best split
            returns splitted dataset and the best split
            In other words, this is one node.
            Building block of a tree.
            '''

            return [X1, y1], [X2, y2], split
```

```
In [9]: def fit_tree(X, y):
            '''
            Given X and y
            Repeat splitter to create a tree.
            return the tree i.e. the trained model
            '''

            return tree
```

```
In [10]: tree = fit_tree(X,y)
```

```
In [11]: def predict_tree(X, tree):
             '''
             Given the data and the model
             predict labels
             '''

             return y_pred
```

```
In [12]: y_pred = predict_tree(X, tree)
```

```
In [13]: def accuracy(y_pred, y):
             return sum(y_pred==y)/len(y)
```

```
In [14]: accuracy(y_pred, y)
```

```
Out[14]: 1.0
```

```
In [15]: # Putting all together
         # with Train/Test
         X_train, y_train = make_circles(n_samples=100, noise=0.05, factor = 0.5)
         X_test, y_test = make_circles(n_samples=100, noise=0.05, factor = 0.5)
         tree = fit_tree(X_train, y_train)
         y_pred_train = predict_tree(X_train, tree)
         y_pred_test = predict_tree(X_test, tree)

         print('Training acc:', accuracy(y_pred_train, y_train))
         print('Testing acc:', accuracy(y_pred_test, y_test))
```

```
Training acc: 1.0
Testing acc: 0.9
```

```
In [16]: def fit_forest(X,y):
             '''
             Fit 30 trees
             by randomly sampling from
             X and y
             return 30 trees
             '''
             num_trees = 30
             forest = []
             for i in range(num_trees):
                 # Randomly sample from X and y
                 # Fit tree
                 forest.append(tree) # Save to forest
             return forest
```

```
In [17]: def predict_forest(X, forest):
             '''
             Predict the labels for X
             for all 30 trees
             calculate the average of 30 trees
             return avg. predictions
             '''

             return y_pred
```

```
In [18]: X_train, y_train = make_circles(n_samples=100, noise=0.05, factor = 0.5)
         X_test, y_test = make_circles(n_samples=100, noise=0.05, factor = 0.5)
         forest = fit_forest(X_train,y_train)
         y_pred = predict_forest(X_test, forest)
         accuracy(y_pred, y_test)
```

```
Out[18]: 1.0
```