project1

March 15, 2017

In [1]:
```
import pandas as pd
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
from sklearn.feature_selection import RFECV
from sklearn.linear_model import LogisticRegression
```

In [2]:
```
#Q1
wine = np.array(pd.read_csv("data.csv"))
```

In [3]:
```
#Q2
# x = np.linspace(1, wine[0].size, wine[0].size)
# for i in range(100):
#     plt.scatter(x, wine[i])
# plt.show()
```

In [4]:
```
#Q3
data, labels = np.split(wine, [wine[0].size-1], axis=1)
print(labels)
```

```
[[ 0.]
 [ 1.]
 [ 0.]
 ...,
 [ 0.]
 [ 0.]
 [ 0.]]
```

In [5]:
```
#Q4
x_train, x_valid = np.split(data, [int(data.shape[0]*0.8)])
labels_train, labels_valid = np.split(labels, [int(labels.shape[0]*0.8)])
print(x_train.shape)
print(x_valid.shape)
print(labels_train.shape)
print(labels_valid.shape)
```

```
(4800, 13)
(1200, 13)
(4800, 1)
(1200, 1)
```

```
In [6]: #Q5
        num_train_samples = [100,200,500,1000,2000,4800]
```

```
In [7]: #Q6
        for train_size in num_train_samples:
            clf = LogisticRegression()
            clf.fit(x_train[:train_size], labels_train[:train_size].ravel())
            print("SAMPLE SIZE: ", train_size, ", Performance:", clf.score(x_valid,
```

```
SAMPLE SIZE:  100 , Performance: 0.736666666667
SAMPLE SIZE:  200 , Performance: 0.264166666667
SAMPLE SIZE:  500 , Performance: 0.8375
SAMPLE SIZE:  1000 , Performance: 0.948333333333
SAMPLE SIZE:  2000 , Performance: 0.98
SAMPLE SIZE:  4800 , Performance: 0.98
```

```
In [8]: #Q7
        for pen in ['l1','l2']:
            print("--- USING:",pen,"---")
            for train_size in num_train_samples:
                clf = LogisticRegression(penalty=pen)
                clf.fit(x_train[:train_size], labels_train[:train_size].ravel())
                print("SAMPLE SIZE: ", train_size, ", Performance:", clf.score(x_va

        print("\nL1 does better, because it has learned better to ignore certain fe
```

```
--- USING: l1 ---
SAMPLE SIZE:  100 , Performance: 0.736666666667
SAMPLE SIZE:  200 , Performance: 0.266666666667
SAMPLE SIZE:  500 , Performance: 0.978333333333
SAMPLE SIZE:  1000 , Performance: 0.975833333333
SAMPLE SIZE:  2000 , Performance: 0.984166666667
SAMPLE SIZE:  4800 , Performance: 0.988333333333
--- USING: l2 ---
SAMPLE SIZE:  100 , Performance: 0.736666666667
SAMPLE SIZE:  200 , Performance: 0.264166666667
SAMPLE SIZE:  500 , Performance: 0.8375
SAMPLE SIZE:  1000 , Performance: 0.945833333333
SAMPLE SIZE:  2000 , Performance: 0.98
SAMPLE SIZE:  4800 , Performance: 0.98

L1 does better, because it has learned better to ignore certain features because it
```

```
In [9]: #Q8
        #Q7
        for c in [0.01,0.03,0.1,0.3,1,3,10,30,50,100,300]:
            print("C =",c)
            for train_size in num_train_samples:
                clf = LogisticRegression(penalty='l1',C=c)
                clf.fit(x_train[:train_size], labels_train[:train_size].ravel())
                print("SAMPLE SIZE: ", train_size, ", Performance:", clf.score(x_va

        print("\nC should be higher here, so I will go with a value of", 50)
```

```
C = 0.01
SAMPLE SIZE:  100 , Performance: 0.263333333333
SAMPLE SIZE:  200 , Performance: 0.263333333333
SAMPLE SIZE:  500 , Performance: 0.273333333333
SAMPLE SIZE:  1000 , Performance: 0.8825
SAMPLE SIZE:  2000 , Performance: 0.94
SAMPLE SIZE:  4800 , Performance: 0.943333333333
C = 0.03
SAMPLE SIZE:  100 , Performance: 0.263333333333
SAMPLE SIZE:  200 , Performance: 0.263333333333
SAMPLE SIZE:  500 , Performance: 0.818333333333
SAMPLE SIZE:  1000 , Performance: 0.925
SAMPLE SIZE:  2000 , Performance: 0.945833333333
SAMPLE SIZE:  4800 , Performance: 0.970833333333
C = 0.1
SAMPLE SIZE:  100 , Performance: 0.400833333333
SAMPLE SIZE:  200 , Performance: 0.264166666667
SAMPLE SIZE:  500 , Performance: 0.760833333333
SAMPLE SIZE:  1000 , Performance: 0.925
SAMPLE SIZE:  2000 , Performance: 0.974166666667
SAMPLE SIZE:  4800 , Performance: 0.98
C = 0.3
SAMPLE SIZE:  100 , Performance: 0.74
SAMPLE SIZE:  200 , Performance: 0.263333333333
SAMPLE SIZE:  500 , Performance: 0.860833333333
SAMPLE SIZE:  1000 , Performance: 0.965833333333
SAMPLE SIZE:  2000 , Performance: 0.978333333333
SAMPLE SIZE:  4800 , Performance: 0.980833333333
C = 1
SAMPLE SIZE:  100 , Performance: 0.736666666667
SAMPLE SIZE:  200 , Performance: 0.266666666667
SAMPLE SIZE:  500 , Performance: 0.979166666667
SAMPLE SIZE:  1000 , Performance: 0.975833333333
SAMPLE SIZE:  2000 , Performance: 0.984166666667
SAMPLE SIZE:  4800 , Performance: 0.988333333333
C = 3
SAMPLE SIZE:  100 , Performance: 0.736666666667
```

```
SAMPLE SIZE:  200 , Performance: 0.94
SAMPLE SIZE:  500 , Performance: 0.8375
SAMPLE SIZE:  1000 , Performance: 0.979166666667
SAMPLE SIZE:  2000 , Performance: 0.985
SAMPLE SIZE:  4800 , Performance: 0.988333333333
C = 10
SAMPLE SIZE:  100 , Performance: 0.736666666667
SAMPLE SIZE:  200 , Performance: 0.826666666667
SAMPLE SIZE:  500 , Performance: 0.833333333333
SAMPLE SIZE:  1000 , Performance: 0.925
SAMPLE SIZE:  2000 , Performance: 0.985
SAMPLE SIZE:  4800 , Performance: 0.989166666667
C = 30
SAMPLE SIZE:  100 , Performance: 0.736666666667
SAMPLE SIZE:  200 , Performance: 0.775833333333
SAMPLE SIZE:  500 , Performance: 0.951666666667
SAMPLE SIZE:  1000 , Performance: 0.849166666667
SAMPLE SIZE:  2000 , Performance: 0.985
SAMPLE SIZE:  4800 , Performance: 0.988333333333
C = 50
SAMPLE SIZE:  100 , Performance: 0.736666666667
SAMPLE SIZE:  200 , Performance: 0.750833333333
SAMPLE SIZE:  500 , Performance: 0.969166666667
SAMPLE SIZE:  1000 , Performance: 0.903333333333
SAMPLE SIZE:  2000 , Performance: 0.984166666667
SAMPLE SIZE:  4800 , Performance: 0.988333333333
C = 100
SAMPLE SIZE:  100 , Performance: 0.736666666667
SAMPLE SIZE:  200 , Performance: 0.736666666667
SAMPLE SIZE:  500 , Performance: 0.9725
SAMPLE SIZE:  1000 , Performance: 0.923333333333
SAMPLE SIZE:  2000 , Performance: 0.984166666667
SAMPLE SIZE:  4800 , Performance: 0.9875
C = 300
SAMPLE SIZE:  100 , Performance: 0.736666666667
SAMPLE SIZE:  200 , Performance: 0.736666666667
SAMPLE SIZE:  500 , Performance: 0.9725
SAMPLE SIZE:  1000 , Performance: 0.8325
SAMPLE SIZE:  2000 , Performance: 0.984166666667
SAMPLE SIZE:  4800 , Performance: 0.9875
```

C should be higher here, so I will go with a value of 50

```
In [11]: #Q9
         # optimal LR that I found
         test = np.array(pd.read_csv("test.csv"))
```

```
ids = np.linspace(0,test.shape[0]-1, test.shape[0])

clf = LogisticRegression(penalty='l1',C=50)
clf.fit(data, labels.ravel()) #give all the data
# print("SAMPLE SIZE: ", data.shape[0], ", Predictions:", )
pred = clf.predict(test);
ids = np.reshape(ids, (ids.shape[0],1))
pred = np.reshape(pred, (ids.shape[0],1))
out = np.concatenate((ids, pred), axis=1)
out = out.astype(int)
np.savetxt("predictions.csv", out, fmt='%i', delimiter=",")
```

In [ ]:

In [ ]: