In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
from scipy import stats
```

In [2]:
```
data = pd.read_csv('bank_features.csv')
data.head()
```

Out[2]:

	duration	poutcome_success	month_oct	contact_unknown	month_mar	month_jan	day	month_nov	month_jul	loan_yes	marital_married	pout
0	79	0	1	0	0	0	19	0	0	0	1	
1	220	0	0	0	0	0	11	0	0	1	1	
2	185	0	0	0	0	0	16	0	0	0	0	
3	199	0	0	1	0	0	3	0	0	1	1	
4	226	0	0	1	0	0	5	0	0	0	1	

In [3]:
```
data['y_yes'].value_counts()
```

Out[3]:
```
0    4000
1     521
Name: y_yes, dtype: int64
```

Up-Sampling

In [4]:
```
# Split the data into two parts
# sample = 0 and sample = 1
data_no = data[data['y_yes'] == 0]
data_yes = data[data['y_yes'] == 1]
```

In [5]:
```
from sklearn.utils import resample
```

In [6]:
```
data_yes_resample = resample(data_yes,replace = True,
                             n_samples = 2000)
data_yes_resample
```

Out[6]:

	duration	poutcome_success	month_oct	contact_unknown	month_mar	month_jan	day	month_nov	month_jul	loan_yes	marital_married	p
1276	252	0	0	0	0	0	23	0	0	0	0	
4009	255	1	1	0	0	0	27	0	0	0	0	
1469	415	1	0	0	0	0	15	0	0	0	1	
3681	412	0	0	1	0	0	21	0	0	0	0	
1056	250	1	0	0	0	0	4	0	0	0	1	
...	...	...	...	...	...	...	...	...	...	...	...	...
334	298	0	0	0	0	0	16	0	0	0	0	
2653	1472	0	0	0	0	0	26	0	0	0	0	
4505	1234	0	0	1	0	0	26	0	0	0	0	
944	314	0	0	0	0	0	1	0	0	0	1	
3558	481	0	0	0	0	0	24	0	0	0	0	

2000 rows × 15 columns

In [7]:
```
data_resample = pd.concat([data_no,data_yes_resample],
                          axis=0)
```

In [8]:
```
data_resample['y_yes'].value_counts()/len(data_resample)
```

Out[8]:
```
0    0.666667
1    0.333333
Name: y_yes, dtype: float64
```

In [9]:
```
# Split data into independent and dependent columns
X = data_resample.iloc[:,:-1] #independent columns
y = data_resample.iloc[:,-1] # dependent columns
```

In [10]:
```
# Splitting data into train and test
np.random.seed(1001)
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.3)
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

Out[10]:
```
((4200, 14), (1800, 14), (4200,), (1800,))
```

Building Machine Learning Model

In [11]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [12]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
```

In [13]:
```
model.fit(x_train,y_train)
```

Out[13]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

Maximum Likelihood Estimation

In [14]:
```
y_train.head()
```

Out[14]:
```
3050    0
2685    1
2860    0
1677    0
4367    0
Name: y_yes, dtype: int64
```

In [15]:
```
# training data
y_pred_train = model.predict_proba(x_train)
y_pred_train
```

Out[15]:
```
array([[0.91782374, 0.08217626],
       [0.25976135, 0.74023865],
       [0.46405359, 0.53594641],
       ...,
       [0.95739474, 0.04260526],
       [0.57320052, 0.42679948],
       [0.97254655, 0.02745345]])
```

In [16]:
```
y_pred_train_1 = y_pred_train[:,1]
y_pred_train_1
```

Out[16]:
```
array([0.08217626, 0.74023865, 0.53594641, ..., 0.04260526, 0.42679948,
       0.02745345])
```

In [17]:
```
from sklearn.metrics import log_loss
def predict_threshold(y,a):
    if y >= a:
        return 1
    else:
        return 0
```

In [18]:
```
y_pred_train_prob_1 = y_pred_train_1
probabilities = np.linspace(0,1,num=100)
mle = []
for p in probabilities:
    loss = []
    for y in y_pred_train_prob_1:
        y_pred_0_5 = predict_threshold(y,p)
        loss.append(y_pred_0_5)
    loss = pd.Series(loss)
    loss_act = log_loss(y_train,loss)
    mle.append(loss_act)
```

In [19]:
```
mle
```

Out[19]:
```
[23.108621080589966,
 23.001712869185102,
 22.508290355008803,
 20.954009054592788,
 19.506636346342308,
 18.207290011584043,
 17.072418228978556,
 15.929322357042786,
 15.049385540095052,
 14.177670908674202,
 13.453983602647263,
 12.55759917817998,
 11.743449935605343,
 10.962195717689465,
 10.337192343356763,
 9.753307892632767,
 9.29277659550842,
 8.791129230480472,
 8.347046112016695,
 7.993424549339936,
 7.598683872674131,
 7.409534196825384,
 7.212160050885671,
 6.998341724272537,
 6.85031230469488,
 6.6611652941709005,
 6.521360915825232,
 6.381557108620586,
 6.4062244267225905,
 6.21707665467725,
 6.126613766227849,
 6.085491796533694,
 6.1923956291907265,
 6.134827003878725,
 6.069035241138139,
 6.118374065709639,
 6.1512654731419305,
 6.1265922532493695,
 6.052577353080201,
 5.995011393092966,
 5.937443910063007,
 5.879877378934751,
 5.871650624279698,
 5.879872238665557,
 5.896318323142382,
 5.830527321923159,
 5.805855625073323,
 5.8963120405911456,
 5.896310327168081,
 5.904532512694962,
 5.904531180032578,
 5.986764648502158,
 6.011434251168249,
 6.16767995448223,
 6.2663606497106805,
 6.406160078167493,
 6.422604449221255,
 6.397932181230398,
 6.504837156169473,
 6.56240045083194,
 6.587069863117691,
 6.603516328355197,
 6.6117390850231,
 6.570620351794735,
 6.677524565212449,
 6.743311949205203,
 6.784429159390843,
 6.817321518724836,
 6.866661485578381,
 6.948895715569321,
 6.998235872803207,
 7.154481956877869,
 7.277834348956154,
 7.327174315809698,
 7.417631492848882,
 7.582101475873491,
 7.639664580155618,
 7.639664009014596,
 7.730122328335823,
 7.804133801658861,
 7.902814877647993,
 8.083731516290447,
 8.330436871588041,
 8.412671482339665,
 8.577141655744615,
```

```
 8.692269958492613,
 8.897857342083203,
 9.045880098348942,
 9.111667672722035,
 9.210349129471846,
 9.391265768114302,
 9.637971123411896,
 9.868229251950618,
 10.098487570869683,
 10.254733845324685,
 10.493214730531312,
 10.739919133927202,
 10.953728703044673,
 11.151092378065655,
 11.430690283077583]
```

In [20]:
```
ind = np.array(mle).argmin()
mx = max(mle)
val = probabilities[ind]
print(ind)
print(mx)
print("Threshold Probability Value =",val)
```

```
46
23.108621080589966
Threshold Probability Value = 0.4646464646464647
```

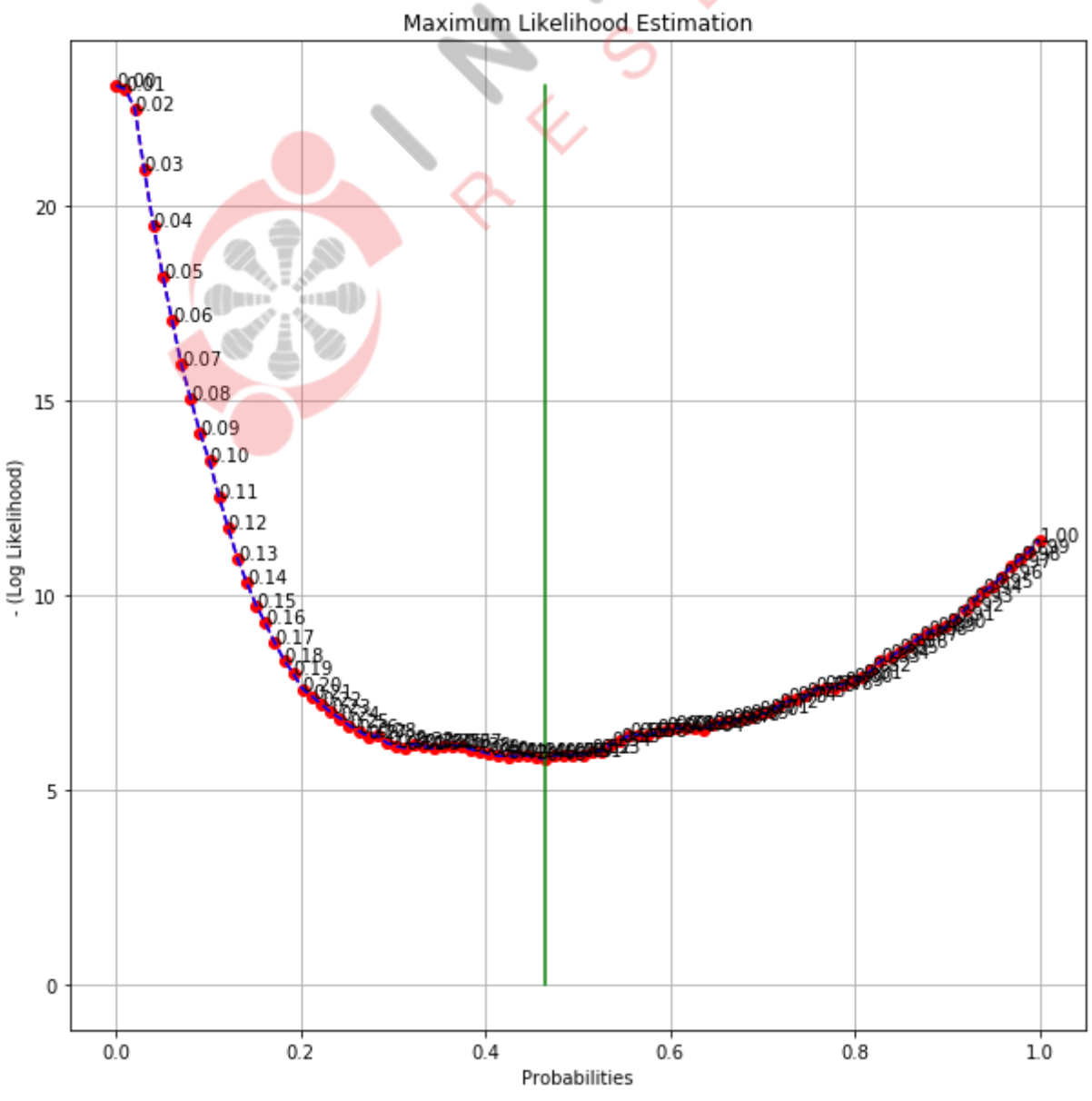
In [21]:
```
plt.figure(figsize=(10,10))
plt.plot(probabilities,mle,'ro--')
plt.plot(probabilities,mle,'b--')
ind = np.array(mle).argmin()
mx = max(mle)
val = probabilities[ind]
print("Threshold Probability Value =",val)

plt.plot([val,val],[0,mx],'g')

for i,p in enumerate(probabilities):
    plt.text(probabilities[i],mle[i],'%0.2f'%(p))

plt.xlabel('Probabilities')
plt.ylabel('- (Log Likelihood)')
plt.title('Maximum Likelihood Estimation')
plt.grid()
plt.show()
```

```
Threshold Probability Value = 0.4646464646464647
```



Apply to test data

In [22]:
```
y_pred_prob_test = model.predict_proba(x_test)
y_pred_prob_test_1 = y_pred_prob_test[:,1] # array (numpy)
```

In [23]:
```
yy = pd.Series(y_pred_prob_test_1)
```

In [24]:
```
y_pred_class_test = []
for i in yy:
    y_pred_class_test.append(predict_threshold(i,0.4040))
```

In [25]:
```
y_pred_class_test
```

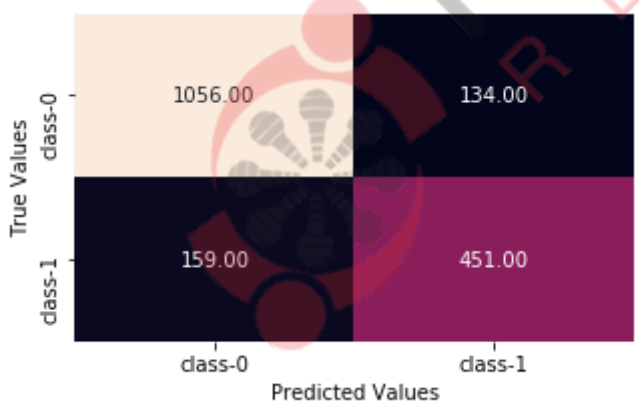
Out[25]:
```
[0,
 0,
 0,
 0,
 0,
 0,
 1,
 0,
 1,
 1,
 0,
 0,
 0,
 0,
 0,
 1,
 1,
 1,
```

Statistical Evaluation

In [26]:
```
from sklearn.metrics import classification_report, confusion_matrix
```

In [27]:
```
cm = confusion_matrix(y_test,y_pred_class_test)
plt.figure(figsize=(5,3))

sns.heatmap(cm,annot=True, fmt = '0.2f',
            xticklabels=['class-0','class-1'],
           yticklabels = ['class-0','class-1'],
           cbar = False)
plt.xlabel('Predicted Values')
plt.ylabel('True Values')
plt.show()
```



In [28]:
```
from sklearn.metrics import precision_score,recall_score,accuracy_score
```

In [29]:
```
precision_score(y_test,y_pred_class_test)
```

Out[29]:
```
0.770940170940171
```

In [30]:
```
recall_score(y_test,y_pred_class_test)
```

Out[30]:
```
0.739344262295082
```

In [31]:
```
accuracy_score(y_test,y_pred_class_test)
```

Out[31]:
```
0.8372222222222222
```

In [32]:
```
cr = classification_report(y_test,y_pred_class_test)
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.87      0.89      0.88      1190
           1       0.77      0.74      0.75       610

    accuracy                           0.84      1800
   macro avg       0.82      0.81      0.82      1800
weighted avg       0.84      0.84      0.84      1800
```

In [33]:
```
from sklearn.metrics import cohen_kappa_score
```

In [34]:
```
cohen_kappa_score(y_test,y_pred_class_test)
```

Out[34]: 0.6330619912335629

In [35]:
```
ra = (197/1800) * (1092/1800) + (101/1800)*(419/1800)
acc = (1092+419)/1800
ra,acc
```

Out[35]: (0.07945771604938272, 0.8394444444444444)

In [36]:
```
kappa = (acc - ra)/(1-ra)
kappa
```

Out[36]: 0.8255858982745342

ROC and AUC (Training Data)

- ROC: Receiver Operating Characterstic
- AUC: Area under the curve

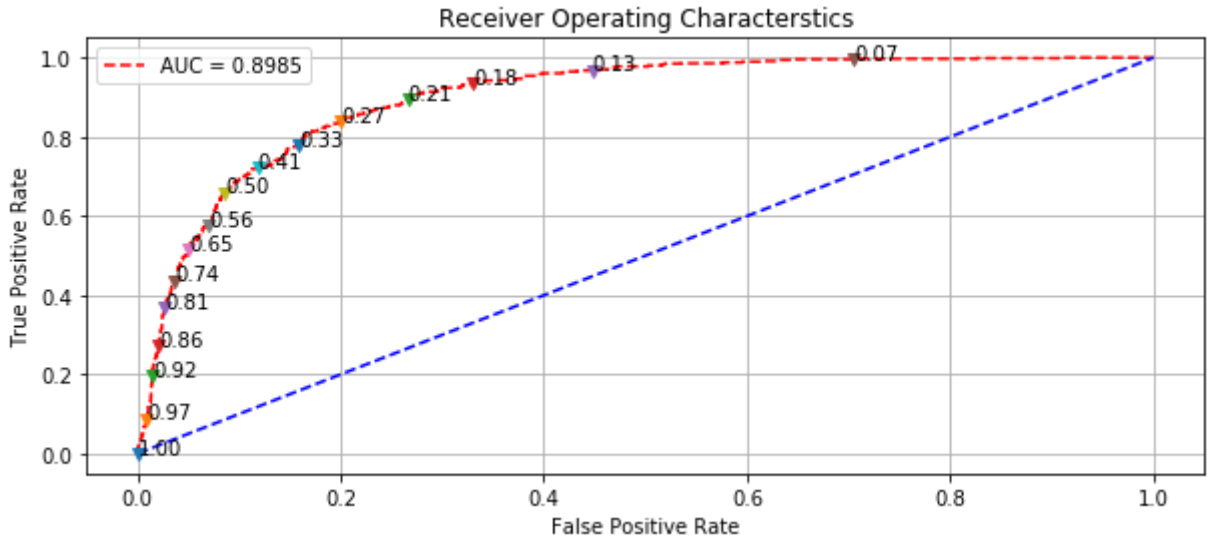
In [37]:
```
from sklearn.metrics import roc_curve,auc
```

In [38]:
```
y_pred_train_prob = model.predict_proba(x_train)
y_pred_train_prob1 = y_pred_train_prob[:,1]
```

In [39]:
```
fpr,tpr,proba = roc_curve(y_train,y_pred_train_prob1)
```

In [40]:
```
plt.figure(figsize=(10,4))
plt.plot(fpr,tpr,'r--')
plt.plot([0,1],[0,1],'b--')
for i in range(len(proba)):
    if i%50 == 1:
        plt.plot(fpr[i],tpr[i],'v')
        plt.text(fpr[i],tpr[i],'%0.2f'%proba[i])

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characterstics')
area = auc(fpr,tpr)
plt.legend(['AUC = %0.4F'%area])
plt.grid()
```



In [41]:
```
auc(fpr,tpr)
```

Out[41]: 0.8985258199134641

In [ ]:
```

```