Homework2

October 28, 2019

[54]:
```
import numpy
import urllib
import random
import math
from matplotlib import pyplot as plt
import scipy.optimize
from scipy.io import arff
import random
from sklearn import svm
from sklearn import linear_model
from sklearn.decomposition import PCA
import warnings
warnings.filterwarnings('ignore')

f = open("/Users/apple/Desktop/CSE258/HW2/5year.arff", 'r')

while not '@data' in f.readline():
    pass
dataset = []
for l in f:
    if '?' in l: # Missing entry
        continue
    l = l.split(',')
    values = [1] + [float(x) for x in l]
    values[-1] = values[-1] > 0 # Convert to bool
    dataset.append(values)
X = [values[:-1] for values in dataset]
y = [values[-1] for values in dataset]
```

1 1

[55]:
```
def accuracy(X, y):
    pred = mod.predict(X)
    correct = pred == y
    accuracy = sum(pred == y)/len(y)
    return accuracy
```

```
def BER(pred, y):
    TP = sum([(p and l) for (p,l) in zip(pred, y)])
    FN = sum([(not p and l) for (p,l) in zip(pred, y)])
    TN = sum([(not p and not l) for (p,l) in zip(pred, y)])
    FP = sum([(p and not l) for (p,l) in zip(pred, y)])
    TPR = TP / (TP + FN)
    TNR = TN / (TN + FP)
    BER = 1 - 1/2 * (TPR + TNR)
    return BER


mod = linear_model.LogisticRegression(C = 1.0, solver='liblinear')
mod.fit(X, y)
pred = mod.predict(X)
print("Accuracy = " + str(accuracy(X, y)))
print("Balanced error rate = " + str(BER(pred, y)))


# TP_ = numpy.logical_and(pred, y)
# FP_ = numpy.logical_and(pred, numpy.logical_not(y))
# TN_ = numpy.logical_and(numpy.logical_not(pred), numpy.logical_not(y))
# FN_ = numpy.logical_and(numpy.logical_not(pred), y)
# TPR_ = sum(TP_) / (sum(TP_) + sum(FN_))
# TNR_ = sum(TN_) / (sum(TN_) + sum(FP_))
# BER_ = 1 - 1/2 * (TPR_ + TNR_)
# print("Balanced error rate = " + str(BER_))
```

```
Accuracy = 0.9663477400197954
Balanced error rate = 0.4810749837661251
```

2 3.

[116]:
```
random.shuffle(dataset)
```

[117]:
```
X = [values[:-1] for values in dataset]
y = [values[-1] for values in dataset]
N = len(X)
X_train = X[:N//2]
X_valid = X[N//2:3*N//4]
X_test = X[3*N//4:]
y_train = y[:N//2]
y_valid = y[N//2:3*N//4]
y_test = y[3*N//4:]

mod = linear_model.LogisticRegression(solver='liblinear',␣
 ↪class_weight='balanced')
```

```python
mod.fit(X_train, y_train)
pred_train = mod.predict(X_train)
pred_test  = mod.predict(X_test)
pred_valid = mod.predict(X_valid)

print("Train Accuracy = " + str(accuracy(X_train, y_train)))
print("Valid Accuracy = " + str(accuracy(X_valid, y_valid)))
print("Test Accuracy = " + str(accuracy(X_test, y_test)))

print("Train Balanced Error Rate = " + str(BER(pred_train, y_train)))
print("Valid Balanced Error Rate = " + str(BER(pred_valid, y_valid)))
print("Test Balanced Error Rate = " + str(BER(pred_test, y_test)))
```

```
Train Accuracy = 0.7795379537953795
Valid Accuracy = 0.7704485488126649
Test Accuracy = 0.7546174142480211
Train Balanced Error Rate = 0.17906435669593557
Valid Balanced Error Rate = 0.20260278024253187
Test Balanced Error Rate = 0.26989917413994013
```

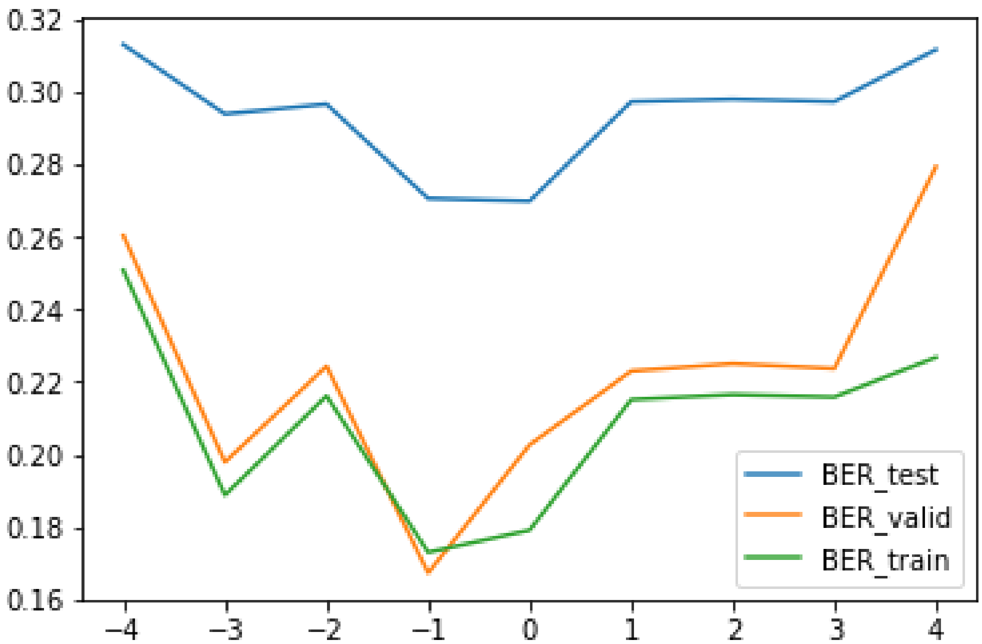
3 4.

[118]:
```python
c = 10**(-4)
BER_train = []
BER_valid = []
BER_test = []
xplot = []


# mod = linear_model.LogisticRegression(C = 100000, class_weight='balanced')
# mod.fit(X_train, y_train)
# pred_test = mod.predict(X_train)
# pred_test  = mod.predict(X_test)
# pred_valid = mod.predict(X_valid)
# BER_train.append([BER(pred_train, y_train)])
# BER_valid.append([BER(pred_valid, y_valid)])
# BER_test.append([BER(pred_test, y_test)])
# print(BER(pred_train, y_train))

while(c <= 10**4):
    xplot.append(math.log10(c))
    mod = linear_model.LogisticRegression(C = c,  class_weight='balanced')
    mod.fit(X_train, y_train)
    pred_train = mod.predict(X_train)
    pred_test  = mod.predict(X_test)
    pred_valid = mod.predict(X_valid)
```

```
    BER_train.append([BER(pred_train, y_train)])
    BER_valid.append([BER(pred_valid, y_valid)])
    BER_test.append([BER(pred_test, y_test)])
    c *= 10
plt.plot(xplot, BER_test, label = 'BER_test')
plt.plot(xplot, BER_valid, label = 'BER_valid')
plt.plot(xplot, BER_train, label = 'BER_train')
plt.legend()
plt.show()
```



I would choose the C with the least validation BER, which is c = 0.1

4 6.

[126]:
```
weights = [1.0] * len(y_train)
mod = linear_model.LogisticRegression(C=1, solver='lbfgs')
mod.fit(X_train, y_train, sample_weight=weights)

pred_test  = mod.predict(X_test)


retrieved = sum(pred_test)
relevant = sum(y_test)
intersection = sum([y and p for y,p in zip(y_test,pred_test)])
```

```
precision = intersection / retrieved
recall = intersection / relevant
F1 = 2 * (precision*recall) / (precision + recall)
F10 = 101*(precision*recall)/(100*precision + recall)
print("Unweighted:  F1 = ", F1 , "F10 = " , F10)


weights = [10.0 if y == True else 1.0 for y in y_train]
mod = linear_model.LogisticRegression(C=1, solver='lbfgs')
mod.fit(X_train, y_train, sample_weight=weights)

pred_test  = mod.predict(X_test)


retrieved = sum(pred_test)
relevant = sum(y_test)
intersection = sum([y and p for y,p in zip(y_test,pred_test)])

precision = intersection / retrieved
recall = intersection / relevant
F1 = 2 * (precision*recall) / (precision + recall)
F10 = 101*(precision*recall)/(100*precision + recall)
print("Weighted:  F1 = ", F1 , "F10 = " , F10)
```

```
Unweighted:  F1 =  0.06896551724137931 F10 =  0.03737971872686898
Weighted:  F1 =  0.22784810126582278 F10 =  0.33030523255813954
```

I would set True elements' weight to 10.0 and False elemtns' weight to 1.0. F1 and F10 both have a better performance.

4.1 7.

[127]:
```
pca = PCA(n_components=5)
pca.fit(X_train)
print(pca.components_[0])
```

```
[-4.08225207e-30 -4.94283948e-09 -1.21834927e-07 -7.86819356e-07
 -4.05306618e-06 -3.83321194e-04  3.29930701e-07 -1.04112602e-06
 -4.17683662e-06  4.76311062e-07  4.22257664e-09 -1.65632413e-07
 -1.01368522e-06  3.61901218e-06 -1.04088220e-06 -8.15097322e-06
 -8.39230607e-07 -4.59587719e-06 -9.51268382e-07 -2.26925419e-07
 -1.91069285e-05 -1.55106956e-08 -1.42470818e-07 -1.98267534e-07
 -6.35269500e-07 -4.82926012e-07 -7.53416661e-07  3.25109126e-05
 -1.84965276e-06 -2.47475652e-06  9.13353971e-07 -2.16220142e-07
  4.54327382e-04 -3.36342523e-06  1.18472731e-06 -1.24215699e-07
  6.18134441e-07  2.71603829e-03  2.60063039e-07 -1.48609549e-07
```

```
-2.17385779e-06  3.58640160e-06 -9.51395723e-08 -4.56527324e-05
-2.65449035e-05 -3.20037258e-07 -3.28339657e-06  3.02097949e-04
-1.84359458e-07 -1.92524093e-07 -3.25124831e-06  5.52632293e-07
 1.23491883e-06  1.66384524e-06 -1.88106626e-06 -9.99996075e-01
-1.34908723e-07  1.85746447e-07  1.78043768e-07  4.32882650e-07
 3.44698665e-05  9.77972129e-06  1.49342259e-04 -4.21071209e-06
 8.17443631e-06]
```

5 8.

[128]:
```
N = 5
print("Validation BER")
while (N <= 30):
    pca = PCA(n_components=N)
    pca.fit(X_valid)
    Xpca_valid = numpy.matmul(X_valid, pca.components_.T)
    mod = linear_model.LogisticRegression(C=1.0, class_weight='balanced')
    mod.fit(Xpca_valid, y_valid)
    pred_valid = mod.predict(Xpca_valid)
    TP = sum([(p and l) for (p,l) in zip(pred_valid, y_valid)])
    FN = sum([(not p and l) for (p,l) in zip(pred_valid, y_valid)])
    TN = sum([(not p and not l) for (p,l) in zip(pred_valid, y_valid)])
    FP = sum([(p and not l) for (p,l) in zip(pred_valid, y_valid)])
    if TP + FN == 0:
        TPR = 1
    else:
        TPR = TP / (TP + FN)
    TNR = TN / (TN + FP)
    BER = 1 - 1/2 * (TPR + TNR)
    N = 5 + N
    print(BER)


print("\n")
N = 5
print("Test BER")
while (N <= 30):
    pca = PCA(n_components=N)
    pca.fit(X_test)
    Xpca_test = numpy.matmul(X_test, pca.components_.T)
    mod = linear_model.LogisticRegression(C=1.0, class_weight='balanced')
    mod.fit(Xpca_test, y_test)
    pred_test = mod.predict(Xpca_test)
    TP = sum([(p and l) for (p,l) in zip(pred_test, y_test)])
    FN = sum([(not p and l) for (p,l) in zip(pred_test, y_test)])
    TN = sum([(not p and not l) for (p,l) in zip(pred_test, y_test)])
    FP = sum([(p and not l) for (p,l) in zip(pred_test, y_test)])
```

```python
    if TP + FN == 0:
        TPR = 1
    else:
        TPR = TP / (TP + FN)
    TNR = TN / (TN + FP)
    BER = 1 - 1/2 * (TPR + TNR)
    N = 5 + N
    print(BER)
```

```
Validation BER
0.3336882579118604
0.26107660455486537
0.25628512274475
0.23992901508429454
0.19100857734398113
0.19032830523513744


Test BER
0.2876830318690784
0.25758727263515224
0.1671986624107007
0.16377868977048182
0.10338450625728335
0.09859654456097688
```