hw2

October 28, 2019

[1]:
```
from sklearn import linear_model
import numpy as np
import random
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.metrics import confusion_matrix
from warnings import filterwarnings
filterwarnings('ignore')
```

[2]:
```
f = open("data/5year.arff", 'r')

# Reading in data
while not '@data' in f.readline():
    pass

dataset = []
for l in f:
    if '?' in l:
        continue
    l = l.split(',')
    values = [1] + [float(x) for x in l]
    values[-1] = values[-1] > 0
    dataset.append(values)

X = [d[:-1] for d in dataset]
y = [d[-1] for d in dataset]
```

[3]:
```
# Fit model
mod = linear_model.LogisticRegression(C=1.0)
mod.fit(X,y)
```

[3]:
```
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)
```

[4]:
```
pred = mod.predict(X)
accuracy = sum(pred == y) / len(pred == y)
print(accuracy)
```

0.96634774002

[5]:
```
# True positives, false positives, etc.
def BER(pred, y):
    TP_ = np.logical_and(pred, y)
    FP_ = np.logical_and(pred, np.logical_not(y))
    TN_ = np.logical_and(np.logical_not(pred), np.logical_not(y))
    FN_ = np.logical_and(np.logical_not(pred), y)

    TP = sum(TP_)
    FP = sum(FP_)
    TN = sum(TN_)
    FN = sum(FN_)

    # BER
    BER = 1 - 0.5 * (TP / (TP + FN) + TN / (TN + FP))
    return BER

print(BER(pred, y))
```

0.485806237825

1. The accuracy of my classifier is 0.96634774002. The BER of my classifier is 0.485806237825.

[6]:
```
random.shuffle(dataset)
X = [d[:-1] for d in dataset]
y = [d[-1] for d in dataset]
```

[7]:
```
n = len(dataset)
train = dataset[0 : int(n * 0.5)]
validate = dataset[int(n * 0.5) : int(n * 0.75)]
test = dataset[int(n * 0.75) : ]

Xtrain = [d[:-1] for d in train]
Ytrain = [d[-1] for d in train]
Xvalidate = [d[:-1] for d in validate]
Yvalidate = [d[-1] for d in validate]
Xtest = [d[:-1] for d in test]
Ytest = [d[-1] for d in test]
```

[8]:
```
mod = linear_model.LogisticRegression(C=1.0,class_weight='balanced')
mod.fit(Xtrain,Ytrain)
```

```
[8]: LogisticRegression(C=1.0, class_weight='balanced', 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)
```

```
[9]: pred_train = mod.predict(Xtrain)
accuracy_train = sum(pred_train == Ytrain) / len(pred_train)
print("The accuracy of training is", accuracy_train)
print("The BER of training is", BER(pred_train, Ytrain))

pred_validate = mod.predict(Xvalidate)
accuracy_validate = sum(pred_validate == Yvalidate) / len(pred_validate)
print("The accuracy of validation is", accuracy_validate)
print("The BER of validation is", BER(pred_validate, Yvalidate))

pred_test = mod.predict(Xtest)
accuracy_test = sum(pred_test == Ytest) / len(pred_test)
print("The accuracy of test is", accuracy_test)
print("The BER of test is", BER(pred_test, Ytest))
```

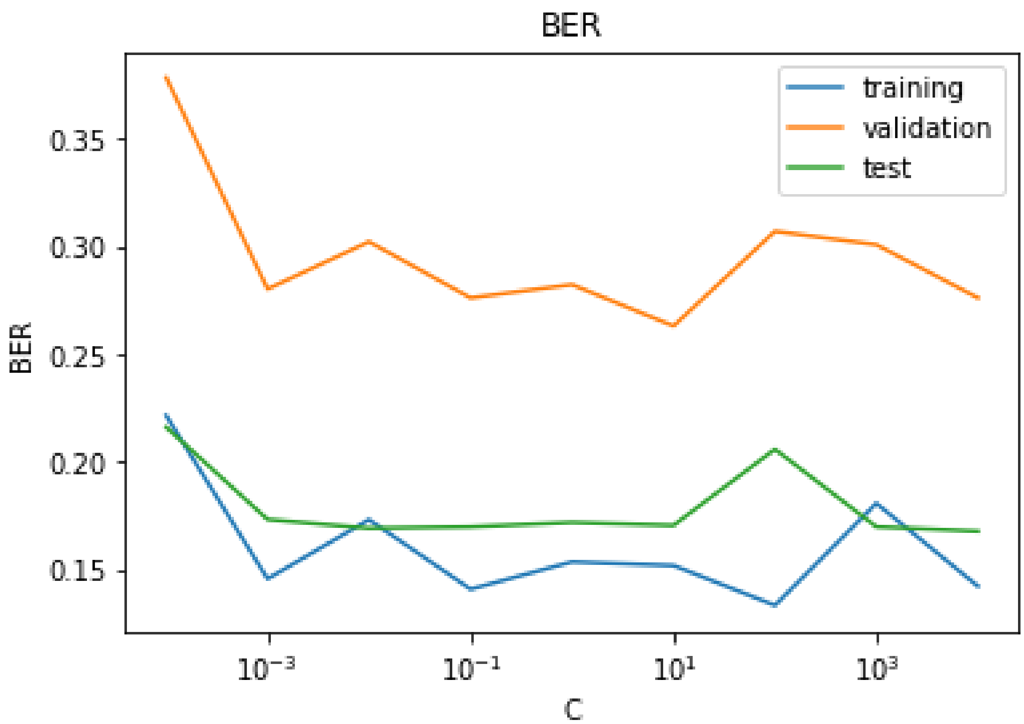
```
The accuracy of training is 0.810561056106
The BER of training is 0.153727851096
The accuracy of validation is 0.765171503958
The BER of validation is 0.282261741906
The accuracy of test is 0.788918205805
The BER of test is 0.172020112393
```

3. The accuracy and BER of training validation and test is shown above.

```
[10]: Crange = np.logspace(-4, 4, 9, base = 10)
BERtrain = []
BERvalidate = []
BERtest = []
```

```
[11]: for curC in Crange:
    mod = linear_model.LogisticRegression(C = curC, class_weight = 'balanced')
    mod.fit(Xtrain,Ytrain)

    pred_train = mod.predict(Xtrain)
    BERtrain.append(BER(pred_train, Ytrain))

    pred_validate = mod.predict(Xvalidate)
    BERvalidate.append(BER(pred_validate, Yvalidate))

    pred_test = mod.predict(Xtest)

    BERtest.append(BER(pred_test, Ytest))
```

[12]:
```
plt.title('BER')
plt.plot(Crange, BERtrain, label='training')
plt.plot(Crange, BERvalidate, label='validation')
plt.plot(Crange, BERtest, label='test')
plt.legend() #
plt.xscale('log')

plt.xlabel('C')
plt.ylabel('BER')
plt.show()
```



4. I will choose C = 10 as the BER of validation of the classifier of C = 10 are the lowest.

[23]:
```
def precAndRec(pred, ytest):
    retrieved = sum(pred)
    relevant = sum(ytest)
    intersection = sum([y and p for y,p in zip(ytest,pred)])

    precision = intersection / retrieved
    recall = intersection / relevant
    return precision, recall

def fBeta(precision, recall, beta):
    return (1 + beta**2)*(precision * recall) / (beta ** 2* precision  + recall)
```

[24]:
```
# mod = linear_model.LogisticRegression(C=1, solver='lbfgs', class_weight =␣
↪'balance')
# mod.fit(Xtrain, Ytrain)
weights = [1.0] * len(Ytrain)
mod = linear_model.LogisticRegression(C=1, solver='lbfgs')
mod.fit(Xtrain, Ytrain, sample_weight=weights)
predTest = mod.predict(Xtest)
```

[25]:
```
precTest, recallTest = precAndRec(predTest, Ytest)
print("F_1 score of unweighted classiffier is", fBeta(precTest, recallTest, 1))
print("F_10 score of unweighted classiffier is", fBeta(precTest, recallTest,␣
↪10))
```

```
F_1 score of unweighted classiffier is 0.142857142857
F_10 score of unweighted classiffier is 0.0876355748373
```

[26]:
```
def weightedY(y):
    weights = []
    for yy in y:
        if yy == True:
            weights.append(10.0)
        else:
            weights.append(1.0)
    return weights
```

[27]:
```
weights = weightedY(Ytrain)
mod = linear_model.LogisticRegression(C=1, solver='lbfgs')
mod.fit(Xtrain, Ytrain, sample_weight=weights)
predTest = mod.predict(Xtest)
```

[28]:
```
precTest, recallTest = precAndRec(predTest, Ytest)
print("F_1 score of weighted classiffier is", fBeta(precTest, recallTest, 1))
print("F_10 score of weighted classiffier is", fBeta(precTest, recallTest, 10))
```

```
F_1 score of weighted classiffier is 0.236842105263
F_10 score of weighted classiffier is 0.386315342116
```

6. The result of F_1 and F_10 score of unweighted and weighted classifier is shown above. Using weighted classifier, we can shown that F_1 and F_10 score are both improved.

[29]:
```
pca = PCA(n_components=len(Xtrain[0]))
pca.fit(Xtrain)
print(pca.components_[0])
```

```
[ -5.14024615e-19  -4.70716233e-09   2.32351081e-07   8.49547651e-07
   4.32903250e-06   4.44892063e-04  -7.88401253e-07   1.36293673e-06
   4.56849651e-06  -7.29058162e-07  -9.09471336e-08   2.22669622e-07
   1.19383771e-06   2.56346099e-07   1.36293673e-06  -1.36691803e-04
```

```
  9.54981556e-07  5.03693471e-06  1.36293673e-06  2.91862171e-07
  3.02680654e-05  6.73286467e-08  2.00503820e-07  2.46031581e-07
  4.47455472e-07  5.87107416e-07  8.44464215e-07 -2.04054145e-05
  2.38608778e-05  3.59409921e-06 -1.01848383e-06  2.83485774e-07
 -2.00464436e-04  2.92040506e-06 -1.66957042e-06  1.68278203e-07
 -7.65899215e-07  4.96064749e-03 -4.52637712e-07  1.88400596e-07
  2.33338326e-06 -6.58014658e-07  2.21932739e-07  5.12981729e-05
  2.10307971e-05 -6.13872159e-08  3.41828975e-06  5.31348925e-05
  2.40595021e-07  2.56164580e-07  3.58977191e-06 -6.29873868e-07
 -5.37871245e-07  1.44341683e-06  2.37213523e-05  9.99987541e-01
  1.94089131e-07  6.23468163e-07 -2.50642039e-07  6.12123773e-07
 -6.57481466e-05 -5.63801511e-06 -2.00226318e-04  3.91282878e-06
 -1.65448929e-06]
```

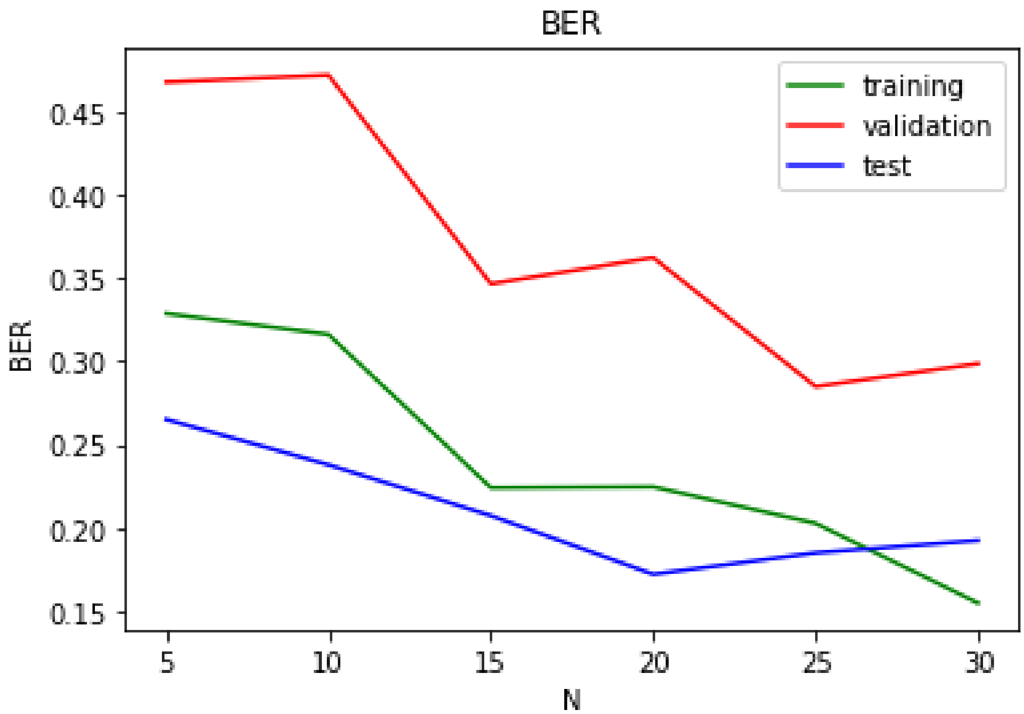
7. The first PCA component is shown above.

[30]:
```
Xpca_train = np.matmul(Xtrain, pca.components_.T)
Xpca_valid = np.matmul(Xvalidate, pca.components_.T)
Xpca_test = np.matmul(Xtest, pca.components_.T)
```

[31]:
```
Nrange = np.arange(5, 35, 5)
BERtrain = []
BERvalidate = []
BERtest = []
for N in Nrange:
    Xcur_train = [d[:N] for d in Xpca_train]
    Xcur_valid = [d[:N] for d in Xpca_valid]
    Xcur_test = [d[:N] for d in Xpca_test]

    mod = linear_model.LogisticRegression(C=1.0,class_weight='balanced')
    mod.fit(Xcur_train,Ytrain)

    predTrain = mod.predict(Xcur_train)
    predValid = mod.predict(Xcur_valid)
    predTest = mod.predict(Xcur_test)

    BERtrain.append(BER(predTrain, Ytrain))
    BERvalidate.append(BER(predValid, Yvalidate))
    BERtest.append(BER(predTest, Ytest))
```

[32]:
```
plt.title('BER')
plt.plot(Nrange, BERtrain, color='green', label='training')
plt.plot(Nrange, BERvalidate, color='red', label='validation')
plt.plot(Nrange, BERtest, color='blue', label='test')
plt.legend() #

plt.xlabel('N')
plt.ylabel('BER')
```

```
plt.show()
```



8. The BER of training, validation and test of different N is shown above.