In [570]:

```
import gzip
from collections import defaultdict
import scipy
import scipy.optimize
import numpy
import random
import numpy as np
import pandas as pd
import numpy
import urllib
import random
import ast

# import warnings filter
import warnings
from warnings import simplefilter
# ignore all future warnings
```

Data

- Polish Bankruptcy data : https://archive.ics.uci.edu/ml/datasets/Polish+companies+bankruptcy+ data

Tasks — Diagnostics (week 2):

In the first homework, we had two issues with the classifiers we built. Namely (1) the data were not shuffled, and (2) the labels were highly imbalanced. Both of these made it difficult to effectively build an accurate classifier. Here we'll try and correct for those issues using the Bankruptcy dataset.

1. Download and parse the bankruptcy data. We'll use the 5year.arff file. Code to read the data is available in the stub.

In [571]:

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

In [572]:

```
while not '@data' in f.readline(): # filter past odd lines
    pass
```

In [573]:

```
def download_bank(file):
    dataset = []
    for l in file:
        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)
    return dataset
```

In [574]:

```
dataset = download_bank(f)
```

Train a logistic regressor (e.g. sklearn.linear model.LogisticRegression) with regularization coefficient C = 1.0.

In [575]:

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

In [576]:

```
from sklearn import linear_model

model = linear_model.LogisticRegression()
model.fit(X, y)
```

```
c:\users\steph\appdata\local\programs\python\python37\lib\site-packages\sklearn\svm\base.py:929: C
onvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

Out[576]:

```
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)
```

- I've created a helper function to calculate the BER for predictions and labels based on the lecture slides.

In [577]:

```
def calc_BER(predictions, y_true):
    TP = sum([(p and l) for (p,l) in zip(predictions, y_true)])
    FP = sum([(p and not l) for (p,l) in zip(predictions, y_true)])
    TN = sum([(not p and not l) for (p,l) in zip(predictions, y_true)])
    FN = sum([(not p and l) for (p,l) in zip(predictions, y_true)])

    TPR = TP / (TP + FN)
    TNR = TN / (TN + FP)
    BER = 1 - 1/2 * (TPR + TNR)
    return BER
```

- This helper function calculates the accuracy of a model given the data of features for prediction, the true labels, and the type of data set it is: Training, Validation, or Testing set

In [532]:

```
def calc_accuracy_BER(mod, X, labels, test=''):
    pred = mod.predict(X)
    correctPred = [1 if pred[i]  == labels[i] else 0 for i in range(len(pred))]
    accur = sum(correctPred) / len(correctPred)
    ber = calc_BER(pred, labels)
    return [accur, ber]
```

- This is purely a funciton that makes answers that only have a few values easier to read, since they are not being formatted in a huge dataframe

In [533]:

```
def format_results(ls, test):
    accur = ls[0]
    ber = ls[1]
    print("{} Accuracy = ".format(test) + str(accur))
    print("{} Balanced Error Rate = ".format(test) + str(ber))
```

Report the accuracy and Balanced Error Rate (BER) of your classifier (1 mark).

In [534]:

```
format_results(calc_accuracy_BER(model, X, y), '')
```

```
 Accuracy = 0.9660178159023425
 Balanced Error Rate = 0.4859769445504388
```

```
Balanced Error Rate = 0.46597694455064566
```

2. (CSE158 only) Retrain the above model using the class weight='balanced' option. Report the accuracy and BER of your new classifier (1 mark).

In [578]:

```
bal_model = linear_model.LogisticRegression(class_weight='balanced')
bal_model.fit(X, y)
```

```
c:\users\steph\appdata\local\programs\python\python37\lib\site-packages\sklearn\svm\base.py:929: C
onvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

Out[578]:

```
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)
```

In [579]:

```
format_results(calc_accuracy_BER(bal_model, X, y), '')
```

```
 Accuracy = 0.7845595513032002
 Balanced Error Rate = 0.20609657314615837
```

3. Shuffle the data, and split it into training, validation, and test splits, with a 50/ 25/ 25% ratio. Using the class weight='balanced' option, and training on the training set, report the training/validation/test accuracy and BER (1 mark).

In [580]:

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

In [581]:

```
while not '@data' in f.readline(): # filter past odd lines
    pass
```

In [582]:

```
dataset = download_bank(f)
```

In [584]:

```
# shuffle the data
import random
random.shuffle(dataset)
X = [values[:-1] for values in dataset]
y = [values[-1] for values in dataset]
```

In [585]:

```
# split the data
N = len(X)
fifty = int(np.around(N*.5))
seventy_five = int(np.around(N*.75))

X_train = X[:fifty]
X_valid = X[fifty:seventy_five]
X_test = X[seventy_five:]

y_train = y[:fifty]
y_valid = y[fifty:seventy_five]
y_test = y[seventy_five:]
```

In [586]:

```
# train on training set
random_model = linear_model.LogisticRegression(class_weight='balanced')
random_model.fit(X_train, y_train)
```

```
c:\users\steph\appdata\local\programs\python\python37\lib\site-packages\sklearn\svm\base.py:929: C
onvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

Out[586]:

```
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)
```

In [587]:

```
# Training Set Results
format_results(calc_accuracy_BER(random_model, X_train, y_train), 'Training')
```

```
Training Accuracy = 0.8410290237467019
Training Balanced Error Rate = 0.2159016844575119
```

In [588]:

```
# Validation Set Results
format_results(calc_accuracy_BER(random_model, X_valid, y_valid), 'Validation')
```

```
Validation Accuracy = 0.8348745046235139
Validation Balanced Error Rate = 0.2236995341614907
```

In [589]:

```
# Testing Set Results
format_results(calc_accuracy_BER(random_model, X_test, y_test), 'Testing')
```

```
Testing Accuracy = 0.8390501319261213
Testing Balanced Error Rate = 0.23842216444588882
```

4. Implement a complete regularization pipeline with the balanced classifier. Consider values of C in the range {10^-4, 10^-3, . . . , 10^3, 10^4}. Report (or plot) the train, validation, and test BER for each value of C.

In [630]:

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

In [631]:

```
while not '@data' in f.readline(): # filter past odd lines
    pass
```

In [632]:

```
# reload and reshuffle

dataset = download_bank(f)
random.shuffle(dataset)
X = [values[:-1] for values in dataset]
y = [values[-1] for values in dataset]

# split the data
N = len(X)
fifty = int(np.around(N*.5))
```

```
fifty = int(np.around(N*.5))
seventy_five = int(np.around(N*.75))

X_train = X[:fifty]
X_valid = X[fifty:seventy_five]
X_test = X[seventy_five:]

y_train = y[:fifty]
y_valid = y[fifty:seventy_five]
y_test = y[seventy_five:]
```

In [633]:

```
# C values: {10^-4, 10^-3, . . . , 10^3, 10^4}

c_model = [10**i for i in range(-4, 5)]
c_model
```

Out[633]:

```
[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
```

In [634]:

```
train_BER = []
valid_BER = []
test_BER = []

# to find BER, I need to compare the actual labels with the predicted labels
from sklearn.linear_model import RidgeClassifier

for mod in c_model:
    reg_model = RidgeClassifier(mod, class_weight='balanced', fit_intercept=False)
    reg_model.fit(X_train, y_train)

    tr = calc_accuracy_BER(reg_model, X_train, y_train, 'Training')
    val = calc_accuracy_BER(reg_model, X_valid, y_valid, 'Validation')
    te = calc_accuracy_BER(reg_model, X_test, y_test, 'Testing')

    train_BER.append(tr[1])
    valid_BER.append(val[1])
    test_BER.append(te[1])
```

```
c:\users\steph\appdata\local\programs\python\python37\lib\site-
packages\sklearn\linear_model\ridge.py:147: LinAlgWarning: Ill-conditioned matrix (rcond=2.96012e-
17): result may not be accurate.
  overwrite_a=True).T
```

In [635]:

```
df = pd.DataFrame({'Training BER': train_BER, 'Validation BER': valid_BER,'Testing BER': test_BER}
)
df.index = c_model
df = df.rename_axis('C Values')
df
```

Out[635]:

C Values	Training BER	Validation BER	Testing BER
0.0001	0.112660	0.216098	0.199506
0.0010	0.112999	0.215409	0.200189
0.0100	0.112999	0.199280	0.198823
0.1000	0.113679	0.179708	0.198823
1.0000	0.115039	0.195837	0.175494
10.0000	0.143719	0.185906	0.165143
100.0000	0.174545	0.225451	0.229561

C Values	Training BER	Validation BER	Testing BER
1000.0000	0.225009	0.228024	0.149019
10000.0000	0.258313	0.264307	0.199874

Based on these values, which classifier would you select (in terms of generalization performance) and why (1 mark)?

To see which value of C we should use with our model on unseen data, we need to look at the performance of each C Value on the data in the validation set (performance here measured by BER (Balanced Error Rate)).

In [636]:

```
min_validation_BER = df['Validation BER'].idxmin()
min_validation_BER
```

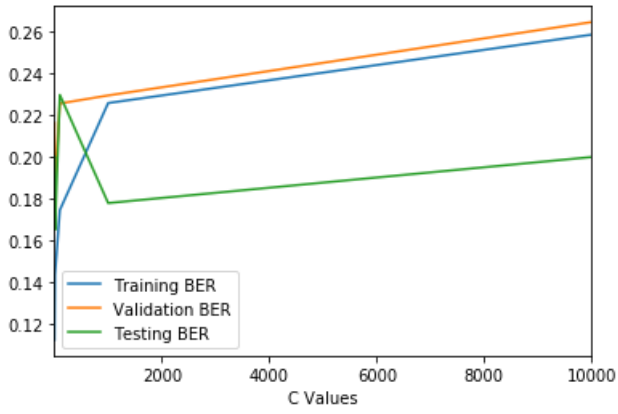
Out[636]:

```
0.1
```

We see that a C Value of 0.1 yields the lowest BER on the validation set. The plot also below helps show the starting point (C = 0.1) is the lowest BER.

In [638]:

```
df.plot();
```



Note: When I reshuffled the data a few times, the "best" C value alternated between the lower values of C (between C = .001 and C = 100). It is important to note that the way the data is shuffled can affect the answer for what the "best" value of C is. However, the larger values for C (i.e. 1000, 10000) didnt produce a low BER for any of the time I shuffled the data. This artifact of the data is reflected in the plot above (and was confirmed in the plots of other shuffled data results). The C values before the 2000 mark show some fluctuation in the BER score, but after that it is mostle constantly increasing.

5. (CSE158 only) Compute the Fβ scores for β = 1, β = 0.1, and β = 10 for the above classifier, using C = 1 (on the test set) (1 mark).

In [639]:

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

In [640]:

```
while not '@data' in f.readline(): # filter past odd lines
    pass
```

In [641]:

```
dataset = download_bank(f)

import random
```

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

# split the data
N = len(X)
fifty = int(np.around(N*.5))
seventy_five = int(np.around(N*.75))

X_train = X[:fifty]
X_valid = X[fifty:seventy_five]
X_test = X[seventy_five:]

y_train = y[:fifty]
y_valid = y[fifty:seventy_five]
y_test = y[seventy_five:]
```

In [642]:

```
model = linear_model.LogisticRegression(C=1.0, class_weight='balanced')
model.fit(X_train, y_train)
predictions = model.predict(X_test)
```

```
c:\users\steph\appdata\local\programs\python\python37\lib\site-packages\sklearn\svm\base.py:929: C
onvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

In [643]:

```
# Args: (y_true, y_pred)
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
```

In [644]:

```
def F_bscore(prec, rec, b):
    return (1 + b**2) * ((prec * rec) / ((b**2 * prec) + rec))
```

- Using the Testing Set, the Fβ score is calculated:

In [646]:

```
scores = []

b_list = [1, 0.1, 10]
for b in b_list:
    precision = precision_score(y_test, predictions)
    recall = recall_score(y_test, predictions)
    scores.append(F_bscore(precision, recall, b))

df = pd.DataFrame({'β = 1': [scores[0]], 'β = 0.01': [scores[1]], 'β = 10': [scores[2]]})
df.index = ['Fβ Score']
df
```

Out[646]:

	β = 1	β = 0.01	β = 10
Fβ Score	0.231156	0.134846	0.808844

Tasks — Dimensionality Reduction (week 3):

Next we'll consider using PCA to build a lower-dimensional feature vector to do prediction.

7. Following the stub code, compute the PCA basis on the training set. Report the first PCA component (i.e., pca.components_[0]) (1 mark).

In [561]:

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

In [562]:

```
while not '@data' in f.readline(): # filter past odd lines
    pass
```

In [563]:

```
dataset = download_bank(f)

import random
random.shuffle(dataset)
X = [values[:-1] for values in dataset]
y = [values[-1] for values in dataset]

# split the data
N = len(X)
fifty = int(np.around(N*.5))
seventy_five = int(np.around(N*.75))

X_train = X[:fifty]
X_valid = X[fifty:seventy_five]
X_test = X[seventy_five:]

y_train = y[:fifty]
y_valid = y[fifty:seventy_five]
y_test = y[seventy_five:]
```

In [564]:

```
from sklearn.decomposition import PCA # PCA library
```

In [565]:

```
n = len(X_train[0]) # extract all features
pca = PCA(n_components = n)
pca.fit(X_train) # computes covarience matrix
```

Out[565]:

```
PCA(copy=True, iterated_power='auto', n_components=65, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

The first PCA component is shown below. This is the eivenvectors of the basis that describes all features of the training data (since we extracted all the features) in order of eigenvalue, increasing by importance.

In [647]:

```
pca.components_[0]
```

Out[647]:

```
array([ 1.72366413e-18, -7.96886398e-08,  3.91014929e-07,  1.11279754e-06,
        5.16667884e-06,  2.78916853e-03, -7.49568382e-07,  1.90082849e-06,
        6.27019137e-06, -1.01100433e-06, -1.64752592e-07,  2.35569473e-07,
        1.35245326e-06, -6.82370062e-06,  1.90036111e-06, -4.06995993e-04,
        8.60036434e-07,  6.80892851e-06,  1.72858080e-06,  5.46735386e-07,
        2.99039874e-05, -1.20289660e-07,  2.25196723e-07,  4.95199103e-07,
        8.84159465e-07,  7.76555544e-07,  7.78796412e-07, -1.15899837e-04,
        1.94249727e-06,  4.82950151e-06, -2.13729773e-06,  5.22896972e-07,
       -5.09301352e-04,  5.19872173e-06, -2.84545331e-06,  1.97850712e-07,
       -1.15035841e-06,  1.13187107e-03, -6.82727802e-07,  1.74804872e-07,
        2.71420122e-06, -3.43963110e-06,  3.27610269e-07,  2.95458611e-05,
       -3.62967208e-07, -1.43794759e-06,  4.12005757e-06, -9.63444238e-05,
        2.90798241e-07,  5.12485806e-07,  4.68138138e-06, -8.69021852e-07,
       -1.35724399e-06, -4.33884092e-06,  2.04207565e-06,  9.99995191e-01,
        2.07380213e-07, -6.18703290e-07, -3.03031270e-07,  4.94394301e-07,
```

```
       -1.77436616e-04, -1.44371281e-05, -2.75137808e-04,  6.64450308e-06,
       -1.81247104e-05])
```

8. Next we'll train a model using a low-dimensional feature vector. By representing the data in the above basis, i.e.:

Xpca*train* = *numpy.matmul(Xtrain, pca.components*.T)

Xpca*valid* = *numpy.matmul(Xvalid, pca.components*.T)

Xpca*test* = *numpy.matmul(Xtest, pca.components*.T)

compute the validation and test BER of a model that uses just the first N components (i.e., dimensions) for N = 5, 10, . . . , 25, 30. Again use class weight='balanced' and C = 1.0 (2 marks).

In [648]:

```
# calculate transformed (rotated) dataset

Xpca_train = numpy.matmul(X_train, pca.components_.T)
Xpca_valid = numpy.matmul(X_valid, pca.components_.T)
Xpca_test = numpy.matmul(X_test, pca.components_.T)
```

In [649]:

```
dim = [i for i in range(5, 35, 5)]
results = {'Validation BER': [], 'Testing BER': []}

for d in dim:
    # train a model on reduced data
    mod = linear_model.LogisticRegression(C=1.0, class_weight='balanced')
    reduced_train = [x[:d] for x in Xpca_train]
    mod.fit(reduced_train, y_train)

    reduced_val = [x[:d] for x in Xpca_valid]
    reduced_test = [x[:d] for x in Xpca_test]

    val = mod.predict(reduced_val)
    test = mod.predict(reduced_test)

    val_ = calc_BER(val, y_valid) # pred, true
    test_ = calc_BER(test, y_test)

    results['Validation BER'] = results['Validation BER'] + [val_]
    results['Testing BER'] = results['Testing BER'] + [test_]
```

```
c:\users\steph\appdata\local\programs\python\python37\lib\site-packages\sklearn\svm\base.py:929: C
onvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
c:\users\steph\appdata\local\programs\python\python37\lib\site-packages\sklearn\svm\base.py:929: C
onvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

In [650]:

```
df = pd.DataFrame(results)
df.index = dim
df = df.rename_axis('N Components')
df
```

Out[650]:

N Components	Validation BER	Testing BER
5	0.377104	0.254699
10	0.336311	0.267214
15	0.376120	0.233546
20	0.344508	0.252115

N Components	Validation BER	Testing BER
25	0.296803	0.163627
30	0.280902	0.160891

The table above shows us that as we increase the number of components of the basis we use, the BER decreases (for validation and testing sets). This makes sense because PCA trys to compress the data into a fewer number of the "useful" dimensions. So the more "useful" dimensions you use for prediction, the more accurate your predictions will be. Obviously, using all dimension will be the most "useful" as far as BER scores go, but the point is to try and make the problem less complex. Choosing the model you for the problem you are solving depends on the right balance you want/ can afford (space and time complexity) between too many dimensions and too low of a accuracy score (whichever form you use).

In [ ]: