Recidivism of Criminal Offenders

A machine-learning approach to justice

Estienne Granet
Department of Statistics
Harvard University
egranet@g.harvard.edu

Victor Lei
Department of Computer Science
Harvard University
vlei@g.harvard.edu

December 11, 2015

Abstract

Recidivism of criminal offenders is a difficult topic in criminal justice. If we view the justice system as serving an important rehabilitatory purpose, then recidivism represents indicators for potential improvement in the correctional system. Using a public data-set on two cohorts of inmates released from North Carolina prisons in 1978 and 1980, we build two predictors of whether an individual will offend again in the future. Our first predictor is one-layer feed-forward neural networks. The second predictor is random forest. Results are presented below.

Introduction

Our goal is to effectively forecast whether an offender will offender again in the future when he or she is released from prison. To achieve this goal, we rely on a survey conducted on prisoners released from North Carolina prisons in 1978 and 1980. The survey contains information on the background of the offenders, including their involvement in drugs or alcohol, level of schooling, nature of the crime resulting in the sample conviction, number of prior incarcerations and recidivism during the followup period following release from the sample incarceration. The dataset also contains information on the the marital status, the sex, the age and the race of the offenders. In hard numbers, the dataset contains 9,327 profiles for prisoners released in 1978 and 9,549 profiles for prisoners released in 1980. Each entry has 16 variables and most variables are indicators. In order to be consistent with the dataset used in previous work, we process the dataset by removing all defective entries (130 from each dataset) and all entries with missing values (4,709 from the 1978 data and 3,810 from the 1980 data.) [6]

1 Literature Review

Historically, recidivism prediction and analysis has been based primarily on explicative models borrowed from survival analysis such as [2] or [8]. More recently, there has been some attempts to use neural networks to improve accuracy.

Schmidt et al. [9] used 'split population' survival time models on the same dataset to both predict the probability of recidivism and probability distribution over time of recidivism. They note that these goals are conceptually distinct, and that therefore the explanatory variables are also likely to be different. Their predictions use a slightly different methodology in that they consider accuracy for the offenders who are most likely and least likely to be recidivists based on their model, thresholded by some proportion of the population. While they were able to predict recidivism more accurately than previous statistical models, they note that level of inaccuracy means the predictions are of questionable practical utility and that 'there is still need for better models and better data.'

Palocsay et al. [6] used the same North Carolina dataset to fit both a feedforward neural network with one hidden layer, and a logistic regression model. They found that the neural network provided superior total accuracy over the logistic regression model for both the 1978, and the 1980 cohorts. In addition, the neural network was also found to have higher accuracy predicting both recidivist and non-recidivist sub-groups. The neural network with the highest validation set accuracy had 39 nodes in the hidden layer. They use a monitoring set to stop training and prevent overfitting. They conclude that neural networks are a viable alternative for modeling recidivism but are highly dependent on network topology and parameters. Their results provide us with a reference for comparison that we use when evaluating our models.

2 Neural Network

2.1 Overview

Neural networks were discovered some time ago, but lately, they have been experiencing a resurgence in usage. There is likely significant complexity in the interactions between the different variables, so simple models like logistic regression are not going to be as capable of modeling them compared to neural networks. Prior work on this data set has considered the use of neural networks and found them to have good predictive power for recidivism compared to traditional modeling techniques like logistic regression models. We seek to harness the improvements in neural networks that have been made over the last few years in an attempt to improve predictive accuracy and reduce model complexity in order to improve generalizability. We find that we can achieve a small, but notable improvement in accuracy on unseen data, as well as a significant reduction in the number of nodes in the hidden layer.

2.2 Model and Inference

As this is a plain binary classifications problem, we use a multi-layer perceptron feedforward neural network with a single hidden layer, bias terms, and the logistic function as the activation function (f) for both the hidden and output layers. There is a single output neuron ($\hat{y} \in (0, 1)$) used for classification with a 0.5 threshold.

For some input vector x, linear weights W, and bias terms b:

$$\hat{y} = f(W_2 f(W_1 x + b_1) + b_2)$$
$$E = -y \ln(\hat{y}) - (1 - y) \ln(1 - \hat{y})$$

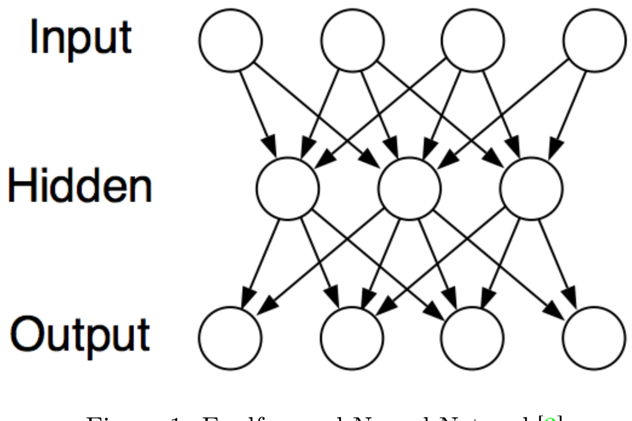

Figure 1: Feedforward Neural Network[3]

Using cross-entropy error (E), we can apply backpropagation and optimize the weights and bias terms by stochastic gradient descent using $\frac{\partial E}{\partial W}$, and $\frac{\partial E}{\partial b}$. Much has been written on the classical backpropagation equations so we omit them here, but see [4] for an overview. We can then adjust the weights by $\eta \frac{\partial E}{\partial W}$ for some η, and do the same for the bias terms. Stochastic gradient descent is generally seen as preferable for neural networks given the significant speedups and the ability to escape from local minima so we use that here.[4] As some of the input variables are binary indicators, and some are large integers (like age in months), we normalize each input variable such that the mean is 0 and variance is 0 so the weights for some the larger valued variables do not dominate the indicator variables. In order to counteract overfitting, we use an early-stopping heuristic to stop training the model when the validation set performance does not improve for a certain number of epochs (100 epochs was a good threshold for this particular problem.) We used the PyBrain library in the implementation, and specified the parameters for the neural network as described above.

2.3 Experiments

Dataset	Palocsay et al.	Our results	Naive baseline
1978	69.20% (39)	70.69% (9)	64.69%
1980	66.98% (26)	68.22% (5)	62.24%

Table 1: Validation set performance for NN predictor

After splitting the 1978 dataset and the 1980 dataset into training and validation sets (7:3 ratio) and training the neural network, validation set performance is improved compared to the results from Palocsay et al [6]. The results are also achieved with far fewer nodes in the hidden layer, making it faster to train and likely more generalizable. As the figures below reveal, the neural network starts to quickly overfit as the number of nodes in the hidden layer increases.

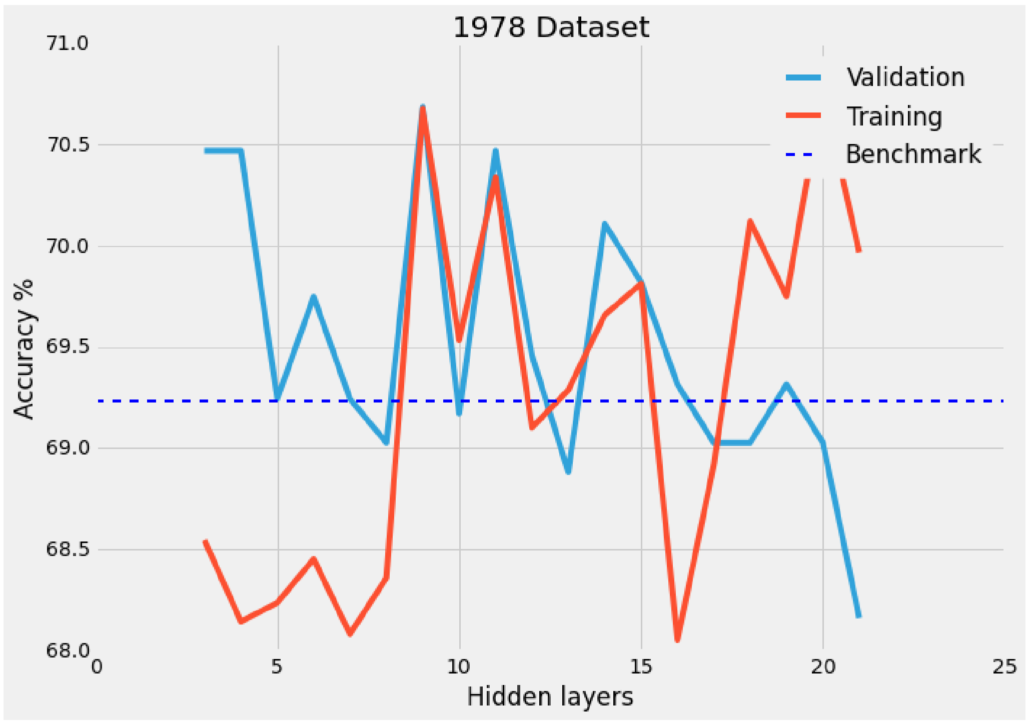

Figure 2: 1978 Dataset

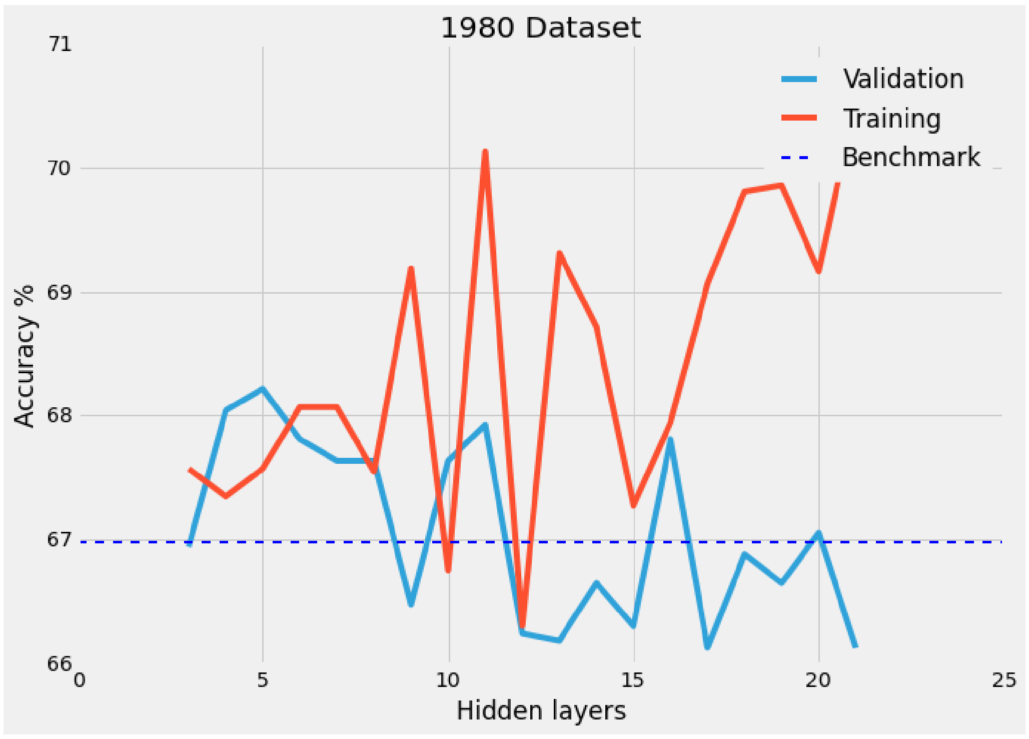

Figure 3: 1980 Dataset

2.4 Discussion

We also attempted variations such as using a tanh activation function rather than the logistic function, but the validation set results were notably lower despite recommendations to the contrary [4]. We suspect it is something about the nature of the data in this specific problem, but there have been some suggestions that binary classification using $\{-1, 1\}$ rather than $\{0, 1\}$ is better for tanh activation.[4]

We attempted to use two hidden layers, but again, validation set results were notably lower likely because even a single hidden layer with a non-linear activation function can approximate significant complexity. Having more than two hidden layers rarely improves performance, and often makes it worse, even though it can theoretically approximate more complex functions. [1]

On the whole, the final configuration worked quite effectively, achieving a substantial improvement over the baseline, and a notable improvement over Pelocsay et al.'s results on the same dataset. In addition, we get better results with far fewer nodes in the hidden layer. Since their paper was from 2000, we believe this improvement is largely due to the strides made around efficient training of neural networks. Now, we can get to very good prediction accuracy with less complexity. Indeed, our experiments to use the same number of nodes as Pelocsay et al. resulted in severe overfitting with poor validation set performance. They also used a proprietary software package with a separate monitoring set used to stop further training so it is difficult to know exactly what other differences there are.

As discussed in a later section, the limited information in the dataset is likely making it difficult to get significantly better results; we can hypothesize that recidivism is a very complex topic that is not so easily described by a person's prior background. For example, a person's social circle, the neighborhood they reside in, and situational factors like success in finding a job, all are potential explanatory variables not captured here.[10] We also note that many of profiles had similar backgrounds, but with ultimately different recidivism outcomes, suggesting there is significant relevant information which is not captured in this dataset.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc.

2.5 Subsection title

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc.

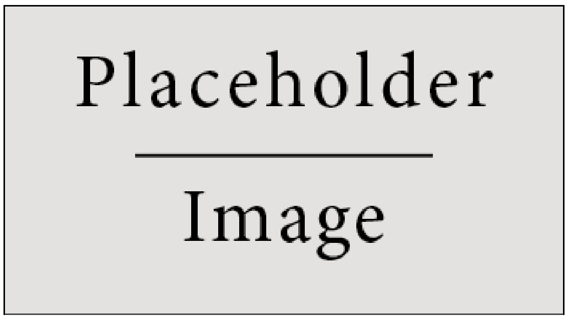

Figure 4: Caption of the figure

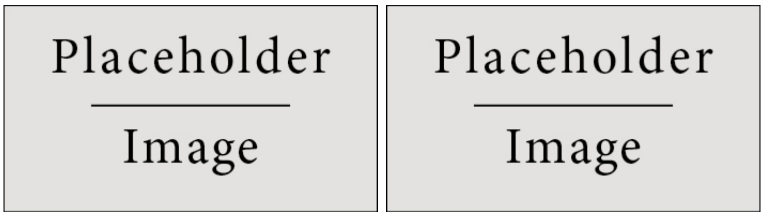

Figure 5: Caption of two figures near one another

Conclusion

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc.

References

[1] Y. Bengio, Y. LeCun, et al. Scaling learning algorithms towards ai.

[2] C.-F. Chung, P. Schmidt, and A. Witte. Survival analysis: A survey. *Journal of Quantitative Criminology*, 7(1):59–98, March 2003.

[3] M. Jordan and C. Bishop. An introduction to graphical models, December 1997.

[4] Y. LeCun, L. Bottou, G. Orr, and K. Muller. Efficient backprop. In G. Orr and M. K., editors, *Neural Networks: Tricks of the trade*. Springer, 1998.

[5] K. Murphy. *Machine Learning, A Probabilist Perspective.* MIT Press, August 2012.

[6] S. Palocsay, P. Wang, and R. Brookshire. Predicting criminal recidivism using neural networks. *Socio-Economic Planning Sciences*, 34(4):271–284, Decemeber 2000.

[7] L. Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 1998.

[8] P. Schmidt and A. Witte. *Predicting Recidivism Using Survival Models.* Springer-Verlag, New York, 1988.

[9] P. Schmidt and A. D. Witte. Predicting criminal recidivism using 'split population'survival time models. *Journal of Econometrics*, 40(1):141–159, 1989.

[10] M. S. Tillyer and B. Vose. Social ecology, individual risk, and recidivism: A multilevel examination of main and moderating influences. *Journal of Criminal Justice*, 39(5):452–459, 2011.

3 Code for Neural Network

```
# Get and process input data
import pandas as pd
import numpy as np

var = dict([ (1, ('WHITE',1)),(2, ('ALCHY',1)),(3, ('JUNKY',1)),(4, ('SUPER',1)),
                (5, ('MARRIED',1)),(6, ('FELON',1)),(7, ('WORKREL',1)),(8, ('PROPTY',1)),
                (9, ('PERSON',1)),(10, ('MALE',1)),(11, ('PRIORS',2)),(13, ('SCHOOL',2)),
                (15, ('RULE',2)),(17, ('AGE',3)),(20, ('TSERVD',3)),
                (23, ('FOLLOW',2)),(25, ('RECID',1)),(26, ('TIME',2)),(28, ('FILE',1)) ] )

def cleanData(data):
    res = []
    cols = [x[1][0] for x in var.items()] # Get the column names
    for line in data:
        line = line.strip()

        curLine = []
        for i in xrange(len(line)):
            if i+1 not in var:
                continue
            name, sz = var[i+1]
            curLine.append(int(line[i:i+sz]))

        res.append(curLine)

    ret = pd.DataFrame(data=res, columns=cols)
    ret = ret[ret.FILE != 3] # Remove incomplete data points

    # Remove some irrelevant columns
    del ret['TIME']
    del ret['FILE']
    del ret['FOLLOW']
    return ret
```

```
raw_1978 = open('data/1978.txt','rb').readlines()
raw_1980 = open('data/1980.txt','rb').readlines()


d1978 = cleanData(raw_1978)
d1980 = cleanData(raw_1980)

print d1978.head()
print len(d1978)

# Neural network for CS 281
# Reference: http://pybrain.org/docs/tutorial

import pybrain
from pybrain.tools.shortcuts import buildNetwork
from pybrain.structure import SoftmaxLayer, TanhLayer
from pybrain.datasets import ClassificationDataSet
from pybrain.supervised.trainers import BackpropTrainer
from pybrain.utilities import percentError
import pickle

def createTrainTest(df, cutProp = 0.3):
    '''
    Input:
        df: Pandas dataframe
        cutProp (default = 0.3): Proportion of samples in the test/validation set

    Output:
        trainD: ClassificationDataSet
        testD: ClassificationDataSet
    '''
    # Load the dataframe data into the pybrain data set
    np_d1978 = (df.values).astype(float)
    np.random.shuffle(np_d1978)
    cutoff = int(np_d1978.shape[0] * cutProp) # < cutoff is the test set, otherwise training set

    # Normalize
    for c in xrange( np_d1978.shape[1] - 1):
        c_sd = np.std(np_d1978[cutoff:, c])
        c_mean = np.mean(np_d1978[cutoff:, c])

        np_d1978[cutoff:, c] = (np_d1978[cutoff:, c] - c_mean) / c_sd

        # Normalize the test set as well
        np_d1978[:cutoff, c] = (np_d1978[:cutoff, c] - c_mean) / c_sd

    testD = ClassificationDataSet(np_d1978.shape[1]-1, nb_classes=2, class_labels=['No', 'Yes'])
    trainD = ClassificationDataSet(np_d1978.shape[1]-1, nb_classes=2, class_labels=['No', 'Yes'])


    for r_idx in xrange(len(np_d1978)):
        r = np_d1978[r_idx, :]
        inD, outD = r[:-1], r[-1]

        if r_idx < cutoff:
            testD.appendLinked(inD, outD)
        else:
            trainD.appendLinked(inD, outD)

    # net = buildNetwork(2, 3, 1, hiddenclass=TanhLayer, outclass=SoftmaxLayer)
    trainD._convertToOneOfMany(bounds=[0,1])
    testD._convertToOneOfMany(bounds=[0,1])

    return trainD, testD


def findBest(trainD, testD, lower = 3, upper = 26, skip = 1):
    resErr = []

    for hid in xrange(lower, upper, skip):
        print 'Trying with', hid, 'hidden layers...'
        net = buildNetwork(trainD.indim, hid, trainD.outdim, bias=True)
        t = BackpropTrainer(net, dataset=trainD)

        bestEpoch = None
        bestErr = 101.
        bestTrErr = 101.
        b0Corr = None
        b1Corr = None

        totalIt = 1000
        epPerIt = 10
        maxContinueEpochs = 100

        for it in xrange(totalIt):
            #     t.trainUntilConvergence(maxEpochs=2000)

            t.trainEpochs(epPerIt)
            testRes = t.testOnClassData(dataset=testD)
            trnresult = percentError( t.testOnClassData(),
                                      trainD['class'] )
            tstresult = percentError( testRes, testD['class'] )

            N1, N0 = 0, 0
            c_1, c_0 = 0, 0
            for i in xrange(len(testRes)):
                if testD['class'][i] == 0:
                    N0 += 1
                    if testRes[i] == 0:
                        c_0 += 1
                else:
                    N1 += 1
                    if testRes[i] == 1:
                        c_1 += 1

            if tstresult < bestErr:
                bestErr = tstresult
                bestTrErr = trnresult
                bestEpoch = t.totalepochs
                b0Corr = float(c_0)/float(N0)
                b1Corr = float(c_1)/float(N1)
            print 'Prop. of 1s:', sum(testRes)/float(len(testRes))
            print '% of class 0 correct:', float(c_0)/float(N0)
            print '% of class 1 correct:', float(c_1)/float(N1)

            print "epoch: %4d" % t.totalepochs, \
                  "  train acc: %5.2f%%" % (100-trnresult), \
                  "  test acc: %5.2f%%" % (100-tstresult)

            # How long since the last best was found?
            if  t.totalepochs - bestEpoch > maxContinueEpochs:
                print 'Finishing with this cycle; no better scores seen for a while'
                break

        resErr.append( (bestErr, bestTrErr, hid, bestEpoch, b0Corr, b1Corr, t) )
        print resErr[-1]

    return sorted(resErr)


trainD, testD = createTrainTest(d1978)

print trainD.calculateStatistics()
print testD.calculateStatistics()

sRes = findBest(trainD, testD, 3, 22, 1)

print sRes

# Reference: http://stackoverflow.com/questions/6568007/how-do-i-save-and-restore-multiple-variabl
with open('1978_best_trainer.pickle', 'w') as f:
    pickle.dump(sRes[0][-1], f)

with open('1978_res.log', 'w') as f:
    f.write(str(sRes))


trainD, testD = createTrainTest(d1980)

print trainD.calculateStatistics()
print testD.calculateStatistics()

sRes = findBest(trainD, testD, 3, 22, 1)

print sRes

# Reference: http://stackoverflow.com/questions/6568007/how-do-i-save-and-restore-multiple-variabl
with open('1980_best_trainer.pickle', 'w') as f:
    pickle.dump(sRes[0][-1], f)

with open('1980_res.log', 'w') as f:
    f.write(str(sRes))
```