Final Project

Kushal Ismael and Matt Kosko

6/23/2021

Contents

1 Introduction

In this report, we will predict violent crime rates using a variety of features plausibly associated with crime. The data comes from the "Communities and Crime" dataset, hosted on the University of California Irvine machine learning repository (Redmond 2009).

1.1 Description of the Data

The dataset, created in July 2009, combines data from three datasets, the 1990 Census, 1995 FBI Uniform Crime Report (UCR), and the 1990 US Law Enforcement Management and Administrative Statistics Survey (LEMAS) (Redmond 2009). The observations in the dataset are at the community level and contain information about socio-economic indicators (e.g., median family income) and crime/law enforcement (e.g., per capita number of police officers). In total, there are 1994 observations and 128 features in the data; 5 of the features are considered non-predictive (`state`, `county`, `community`, `communityname`, `fold`) and one feature is the outcome (`ViolentCrimePerPop`), leaving us with 122 predictive features.

Normally, we would scale the data as part of the preprocessing step in the machine learning process. However, this data has already been standardized to a 0-1 interval using an "equal-interval binning method" (Redmond 2009). The method used preserves the distribution of each feature as well as the ratios of values within features.

There are too many potential features to include all possible pairwise plots. However, we show a scatter plot of the violent crime rate per 100,000 population in Figure 1. We see that most communities have a relatively low violent crime rate, with a small number of very violent communities.

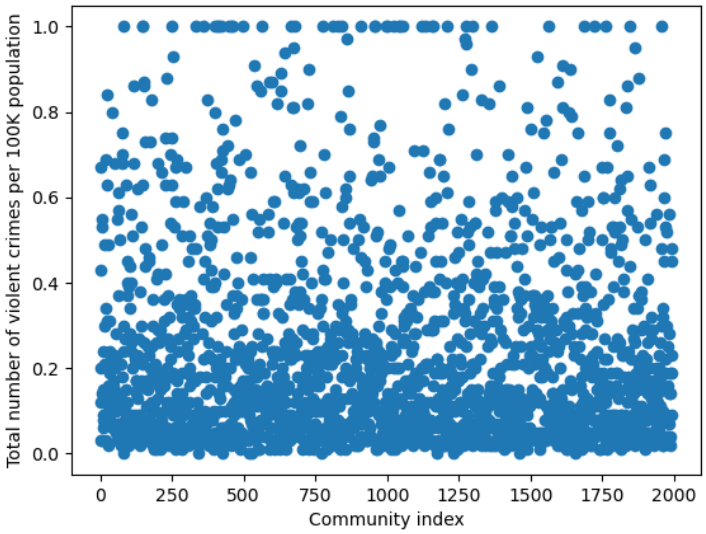

Figure 1: Violent Crime Per 100k

1.2 Machine Learning Approach

The main model that we will use is a *multi-layer perceptron* (MLP). The multilayer perceptron is a type of artificial neural network, with multiple hidden layers and neurons in each layer. MLPs are extremely useful and versatile in that they can approximate almost any function to an arbitrary degree of accuracy (Hagan, Demuth, and Beale 2014, 29). An example of a 3-layer network is taken from (Hagan, Demuth, and Beale 2014, 364) and shown in Figure 2.

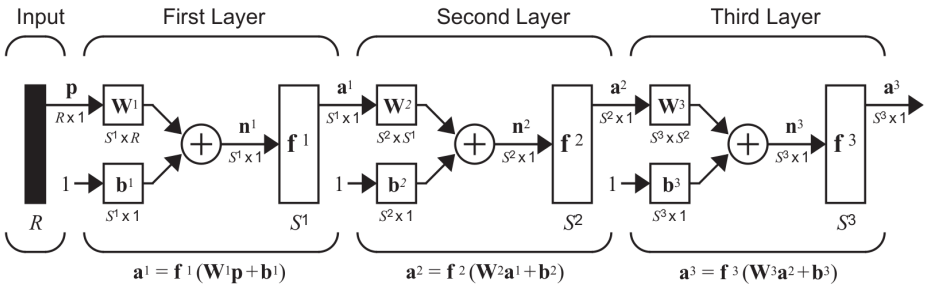

Figure 2: 3-Layer Perceptron

We use the *backpropagation* algorithm to find the appropriate weights and biases for our neural network. The backpropagation is based on optimizing some performance index; in our case, the performance index is the mean squared error (MSE). Let X be a set of weights and biases in a multilayer network. The MSE is defined as:

$$MSE(\mathbf{X}) = E(\mathbf{t} - \mathbf{a})^T(\mathbf{t} - \mathbf{a})$$

where E is the expectation operator over the set of input vectors, $\mathbf{t}$ is the target output, and $\mathbf{a}$ is the output of the netwowrk with the weights and biases $\mathbf{x}$ (Hagan, Demuth, and Beale 2014, 364). Because we do not know the probability distribution of the input vectors, we will approximate the MSE by replacing the expecation with the actual squared error.

$$\hat{MSE}(\mathbf{X})_k = (\mathbf{t}(k) - \mathbf{a}(k))^T(\mathbf{t}(k) - \mathbf{a}(k))$$

where k indicates the kth iteration in an optimization algorithm. Once we construct this, we can calculate derivatives with respect to each weight and bias terms and apply gradient descent methods to update weight and biases. The goal is to minimize the performance index.

The backpropagation algorithm is named because it consists of both forward and backward steps. In the forward portion, we feedforward inputs with initialized weights and biases and record the output error, $\mathbf{e} = \mathbf{t} - \mathbf{a}$. Initial weights are chosen to be small, from a uniform distribution centered around 0; experimental results have shown that these produce the fasting learning in backpropagation (Lari-Najafi, Nasiruddin, and Samad 1989, 218). Moreover, both extremely large weights and weights set zero should be avoided. The zero point is a saddle point on the performance surface, while the surface tends to be flat at extreme values (Hagan, Demuth, and Beale 2014, 418). In the backwards part, we backpropagate the gradient of the MSE with respect to the various weights and biases; this is accomplished through the chain rule (Aggarwal and others 2018).

Because the backpropagation algorithm involves the minimization of the performance index, it is simply a numerical optimization problem (Hagan, Demuth, and Beale 2014, 414). In class, we mostly focused on stochastic gradient descent, however for this project, we use an L-BFGS (Limited memory Broyden–Fletcher–Goldfarb–Shanno) solver. This algorithm is an approximation of the Newton method (Aggarwal and others 2018, 148). In the typical Newton method, we update the weight and bias vector $\mathbf{W}$ as:

$$\mathbf{W}(t+1) = \mathbf{W}(t) - \mathbf{H}^{-1}\nabla F(t)$$

where $\mathbf{H}^{-1}$ is the inverse of the Hessian matrix and F is the performance index (Aggarwal and others 2018, 148). In the L-BFGS algorithm, we replace $\mathbf{H}^{-1}$ with an approximation $\mathbf{G}(t)$ that is updated at each step. Added in a learning rate, we get:

$$\mathbf{W}(t+1) = \mathbf{W}(t) - \alpha(t)\mathbf{G}(t)\nabla F(t)$$

2 Experimental Setup

First, we need to preprocess the data to prepare the model for training. First, there are 25 features that have missing values. Most of the missing values are related to police variables. This is unsurprising as the police data came from surveys of "police departments with at least 100 officers" (Redmond 2009). Figure 3 shows the features with missing data and the percent of observations with missing information.

The most common approach for dealing with missing data is to simply drop the missing values from the analysis (Van Buuren 2018). This approach will not work with our data; because law enforcement information is not recorded based on the number of officers in a community, the data is, in the words of (Rubin 1976), "missing not at random." This means that the missingness of the data depends on the unobserved values themselves. Instead of dropping data, simple approaches, such as imputing missing values with the mean of the observed values are used. This approach however, is only appropriate with a handful of missing values and will otherwise lead to biased estimates of parameters (Van Buuren 2018, 12).

It is likely that these policing variables are important predictors of violent crime. As a result, we will perform two analyses, one with the high missing features dropped and another with the features imputed. As a robustness check, we will see if the accuracy of the models is different with the missing data imputed.

While this data is not high-dimensional in that the number of features is not greater than the number of observations; however, there are a lot of features for a relatively small number of observations. As a result, we will first undertake feature selection by excluding variables that are highly correlated with one another; in this case, we drop variables that are greater than 0.9 in correlation. The algorithm for removing highly correlated features is taken from (Kuhn, Johnson, and others 2013, 26:47) and the code implementing this in R in the `caret` package (Kuhn et al. 2020) was translated into Python.

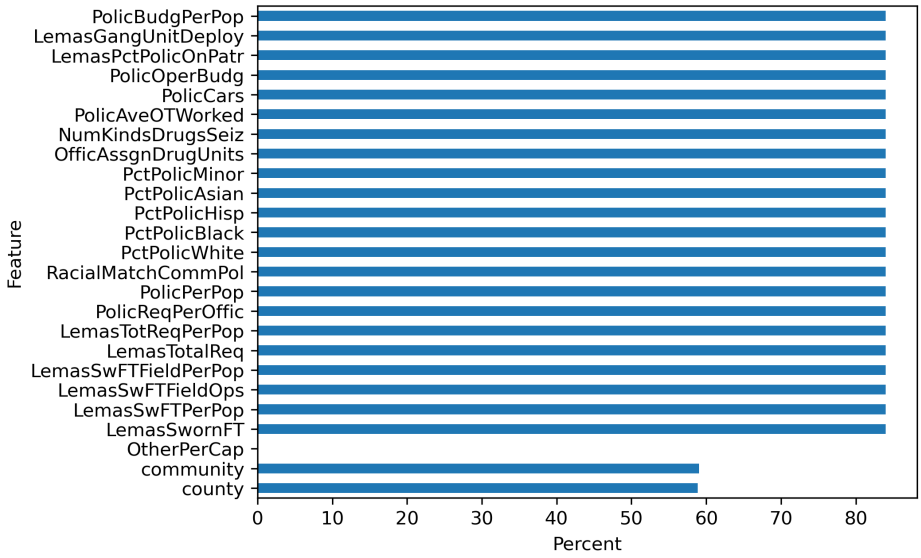

Figure 3: Missing Data Features

By standardizing the data in this way, we can improve the performance of the backpropagation algorithm (Kim 1999).

In order to accurately estimate the true error of the model, we will split the data into a training set and test set. The training set is use to actually fit the model, estimating the weights and biases. The test set is a hold out set, data that the model has not seen. The performance of the model on the test set will provide a good indication of how the model would perform on future data.

The MLP has several *hyperparamaters* that we need to choose as well, including the number of hidden layers and the number of neurons in each layer. In order to choose these parameters, we use 5-fold cross validation with the `GridSearchCV` function.

In addition to the MLP, we also train a support vector machine (SVM), linear regression model, and decision tree to compare the performance of classical machine learning methods with the MLP. The accuracy of the different approaches will be compared using the MSE on the test data; the model with the smallest MSE will be considered the "best" model.

3 Results

The result of our k-fold cross validation chose the MLP with .. layers. Table BLANK shows the MSE error for our MLP as well as the classical machine learning methods.

4 Summary and Conclusions

One potential future improvement is to inclue past information in a time series context. A model that predicts crime rates that have already occurred it of limited utility; ideally, we would want to know how the conditions today will affect violent crime rates next year or next month. One obvious application of this kind of model would be in crime prevention. If you could accurately predict violent crime rates for a particular community, policymaker would be able to

A Appendix A

```
def findCorrelations(correlations, cutoff=0.9):
    corr_mat = abs(correlations)
    varnum = corr_mat.shape[1]
    tmp = corr_mat.copy(deep=True)
    np.fill_diagonal(tmp.values, np.nan)
    maxAbsCorOrder = tmp.apply(np.nanmean, axis=1)
    maxAbsCorOrder = (-maxAbsCorOrder).argsort().values
    corr_mat = corr_mat.iloc[list(maxAbsCorOrder), list(maxAbsCorOrder)]
    del (tmp)
    deletecol = np.repeat(False, varnum)
    x2 = corr_mat.copy(deep=True)
    np.fill_diagonal(x2.values, np.nan)
    for i in range(varnum):
        if not (x2[x2.notnull()] > 0.9).any().any():
            print('No correlations above threshold')
            break
        if deletecol[i]:
            continue
        for j in np.arange(i + 1, varnum, 1):
            if (not deletecol[i] and not deletecol[j]):
                if (corr_mat.iloc[i, j] > cutoff):
                    mn1 = np.nanmean(x2.iloc[i,])
                    mn2 = np.nanmean(x2.drop(labels=x2.index[j], axis=0).values)
                    if (mn1 > mn2):
                        deletecol[i] = True
                        x2.iloc[i, :] = np.nan
                        x2.iloc[:, i] = np.nan
                    else:
                        deletecol[j] = True
                        x2.iloc[j, :] = np.nan
                        x2.iloc[:, j] = np.nan
    newOrder = [i for i, x in enumerate(deletecol) if x]
    return(newOrder)
```

References

Aggarwal, Charu C, and others. 2018. "Neural Networks and Deep Learning." *Springer* 10. Springer: 978–3.

Hagan, Martin T, Howard B Demuth, and Mark Beale. 2014. *Neural Network Design*. https://hagan.okstate.edu/nnd.html.

Kim, Daehyon. 1999. "Normalization Methods for Input and Output Vectors in Backpropagation Neural Networks." *International Journal of Computer Mathematics* 71 (2). Taylor & Francis: 161–71.

Kuhn, Max, Kjell Johnson, and others. 2013. *Applied Predictive Modeling*. Vol. 26. Springer.

Kuhn, Max, Jed Wing, Steve Weston, Andre Williams, Chris Keefer, Allan Engelhardt, Tony Cooper, et al. 2020. "Package 'Caret'." *The R Journal*, 223.

Lari-Najafi, Hossein, Mohammed Nasiruddin, and Tariq Samad. 1989. "Effect of Initial Weights on Back-Propagation and Its Variations." In *Conference Proceedings., Ieee International Conference on Systems, Man and Cybernetics*, 218–19. IEEE.

Redmond, Michael. 2009. "Communities and Crime." https://archive.ics.uci.edu/ml/datasets/Communities+and+Crime.

Rubin, Donald B. 1976. "Inference and Missing Data." *Biometrika* 63 (3). Oxford University Press: 581–92.

Van Buuren, Stef. 2018. *Flexible Imputation of Missing Data*. CRC press.