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 dataset we use 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

This dataset, created in July 2009, combines data from three different sources, 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 both 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 and a target variable.

For the exploratory data analysis, we would like to plot the outcome against each feature; however, there are too many potential features to include all possible pairwise plots. Instead, we show a scatter plot of the violent crime rate per 100,000 population in Figure 1 as well as a histogram of this same data in Figure 2. We see that most communities have a relatively low violent crime rate, with a small number of very violent communities.

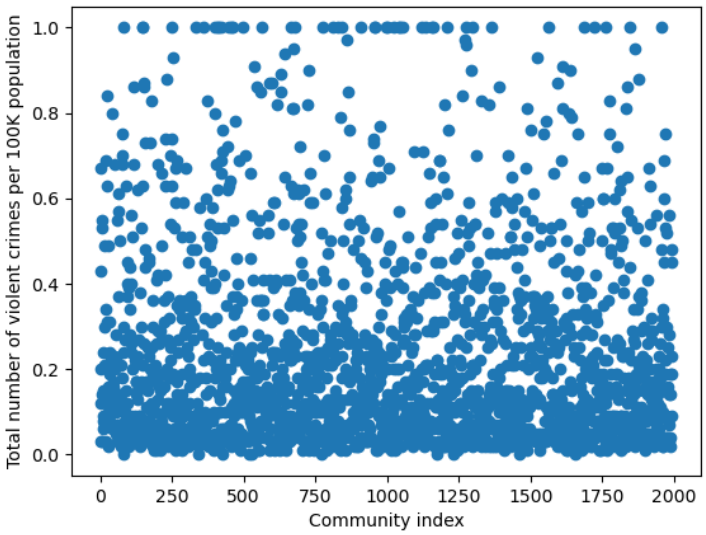

Figure 1: Violent Crime Per 100k

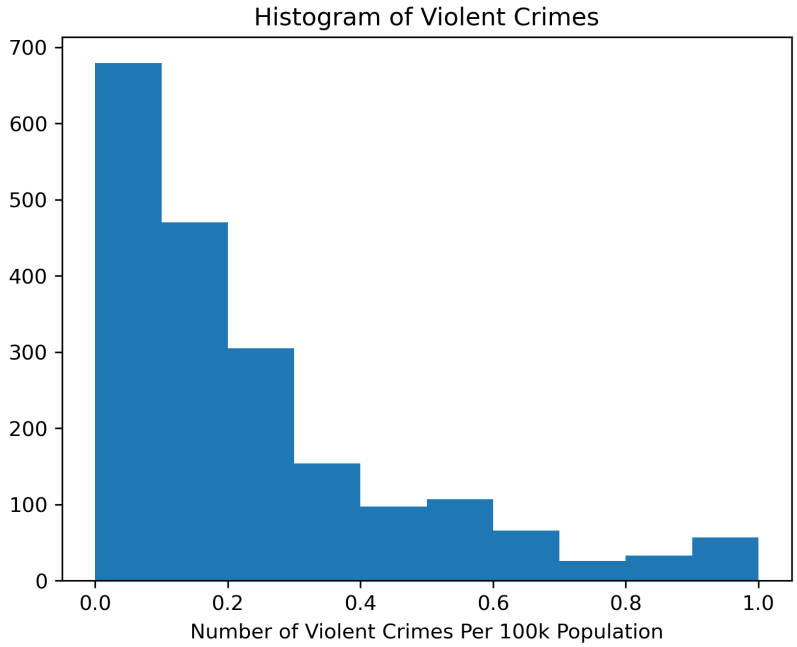

Figure 2: Histogram of Violent Crimes

1.2 Machine Learning Approach

The main model that we will use is a *multilayer 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). Because of this property we expect *a priori* that the MLP will outperform other classical machine learning (ML) methods. An example of a an MLP with 3 layers is taken from (Hagan, Demuth, and Beale 2014, 364) and shown in Figure 3.

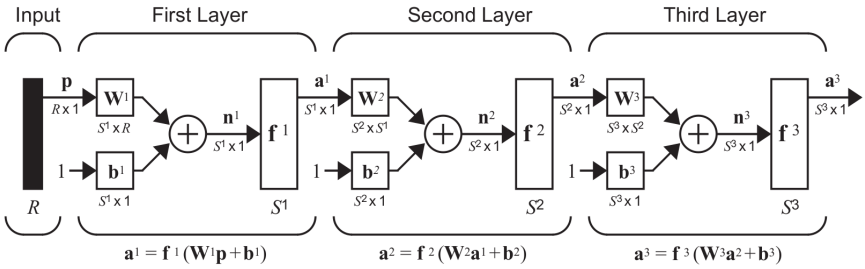

Figure 3: 3-Layer Perceptron

Once we have constructed the architecture of the neural network (choosing the number of hidden layers, neurons, and activation function; see the "Experimental Setup" section), we use the *backpropagation* algorithm to find the appropriate weights and biases of the network. The backpropagation is based on optimizing some performance index; in our case, our chosen 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, $\mathbf{t}$ is the target output, and $\mathbf{a}$ is the output of the network 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 expectation 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 index, 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 MSE.

The backpropagation algorithm is so 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 fastest 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. Normally, we would scale the data as part of the preprocessing step in the ML process. By standardizing the data in this way, we can improve the performance of the backpropagation algorithm (Kim 1999). 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. This type of scaling is desirable because it allows us to interpret changes in a feature such as doubling or halving in the same way as the original scale of the data.

Next we need to handle the missing values. There are 25 features in total that have any missing values. Most of the missing values are related to police variables. This is unsurprising as law enforcement data from LEMAS only collects information on "police departments with at least 100 officers" (Redmond 2009). Figure 4 shows the features with missing data and the percent of observations with missing information.

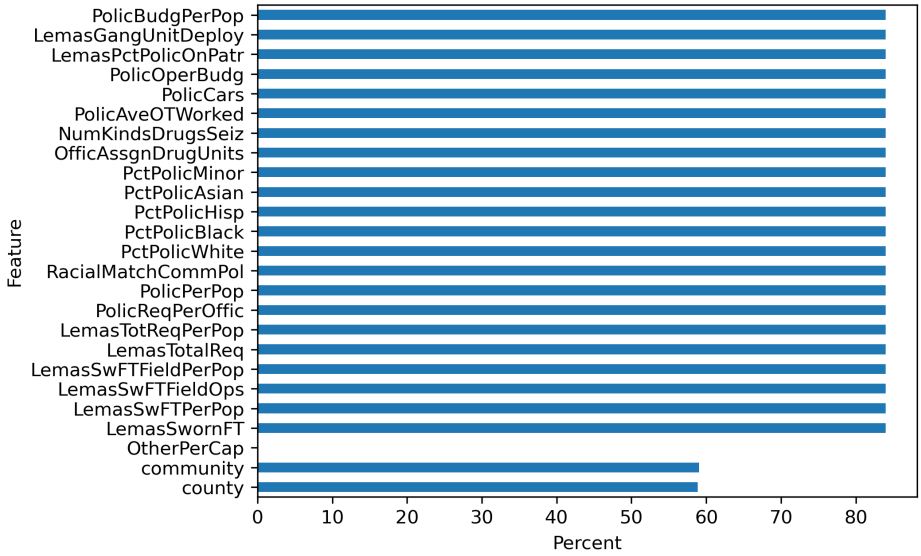

Figure 4: Missing Data Features

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" (MNAR). This means that the missingness of the data depends on the unobserved values themselves. Moreover, it is likely that policing variables with many missing values are important predictors of violent crime rates. When dropping data in not possible in MNAR cases, simple approaches, such as imputing missing values with the mean or median of the observed values are often 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). Instead, we will use a multiple imputation algorithm, the multiple imputation by chained equations (MICE) created by (Van Buuren 2018) and implemented in Python as `IterativeImputer`. Because of uncertainty in the missing data imputation model, we will perform two

analyses, one where features with missing data are dropped altogether and another where 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, as part of preprocessing we will 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 (the code is in Appendix B). The algorithm proceeds as follows:

1. Calculate correlation matrix (`.corr()`)
2. Find two predictors with largest absolute pairwise correlation (A and B)
3. Find average correlation for all features with A and all features with B
4. If A greater than B, remove A. Else, remove B
5. Repeat until no correlations above threshold

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 *hyperparameters* 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 (CV) with the `GridSearchCV` function. This function exhaustively creates grids of candidate paramater values that will then be used to fit a model (Pedregosa et al. 2011). Whichever set of parameters produces the smallest MSE in the holdout sets (equivalently, the largest negative MSE) is chosen as the best model.

In addition to the MLP, we also train a support vector machine (SVM), random forest model, and decision tree to compare the performance of classical machine learning methods with the MLP. Similar to the MLP, we will use `GridSearchCV` with 5-fold CV to find the best model. The accuracy of the different approaches will then be compared using the MSE from the 5-fold CV data; the model with the smallest MSE will be considered the best model.

3 Results

Figure 5 shows the final neuron architecture for the MLP on the nonimputed data.

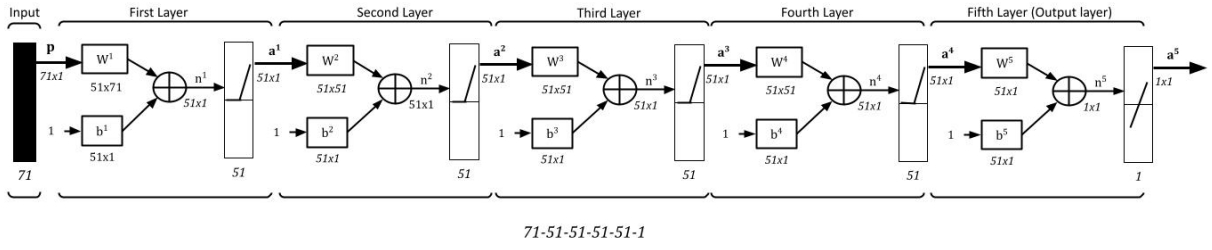

Figure 5: Chosen MLP

Table 1 shows the model results for the classical machine learning methods and the MLP on the non-imputed data. In this case, the 5-fold CV chose a ReLu activation funciton and four hidden layers, each with 51 neurons. This MLP is superior to all the other classical ML models on both the test and training data. Table 2 shows the same thing except on the non-imputed data.

Table 1: Model Performance for Non-Imputed Data

Model Type	Best 5-Fold MSE	Test MSE
SVR	0.0186	0.0203
DTR	0.0356	0.0453
RFR	0.0181	0.0187
MLP	0.0180	0.0130

Table 2: Model Performance for Imputed Data

Model Type	Best 5-Fold MSE	Test MSE
SVR	0.0185	0.0199
DTR	0.0352	0.0389
RFR	0.0178	0.0187
MLP	0.0181	0.0167

We see that, for the imputed data, the MLP model is not the unambiguous winner, and its cross-validated error is higher than the model fit on non-imputed data. Thus, we prefer the model where features with missing values are dropped as it produces the highest accuracy. When fit on the test data holdout, the MLP is also the best performer.

Figure 6 shows a histogram of violent crime rates for the actual and predicted values using the best MLP model from non-imputed data.

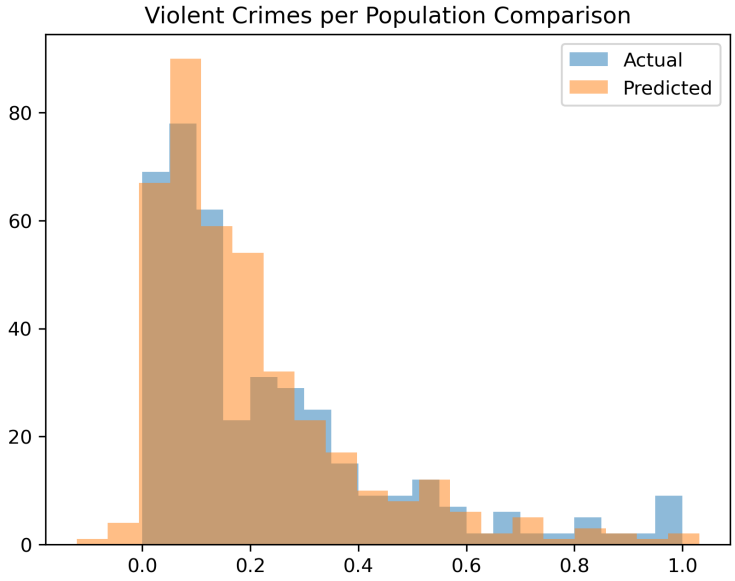

Figure 6: Violent Crime Per Population Comparison (Test Data)

To illustrate a real world application of our model, we next show how it can be used to assess how changes in a feature are associated with changes in the violent crime rate. We construct an "average" community, where all features are set to the average levels in the dataset. Then, we check the predicted values of violent crime rate across a range of values for a particular variables. Thus, in the same way that coefficient estimates in OLS regression allow us to make the *ceteris paribus* interpretations about how the outcome variable changes with changes in a feature, we can see how altering feature values changes the outcome, holding all else constant.

Figure 7 shows the predicted crime rate for the constructed "average" community where three features

are varied across the 0-1 interval. We see, unsurprisingly, that the violent crime rate increases with the unemployment rate (and correspondingly decreases with the employment rate).

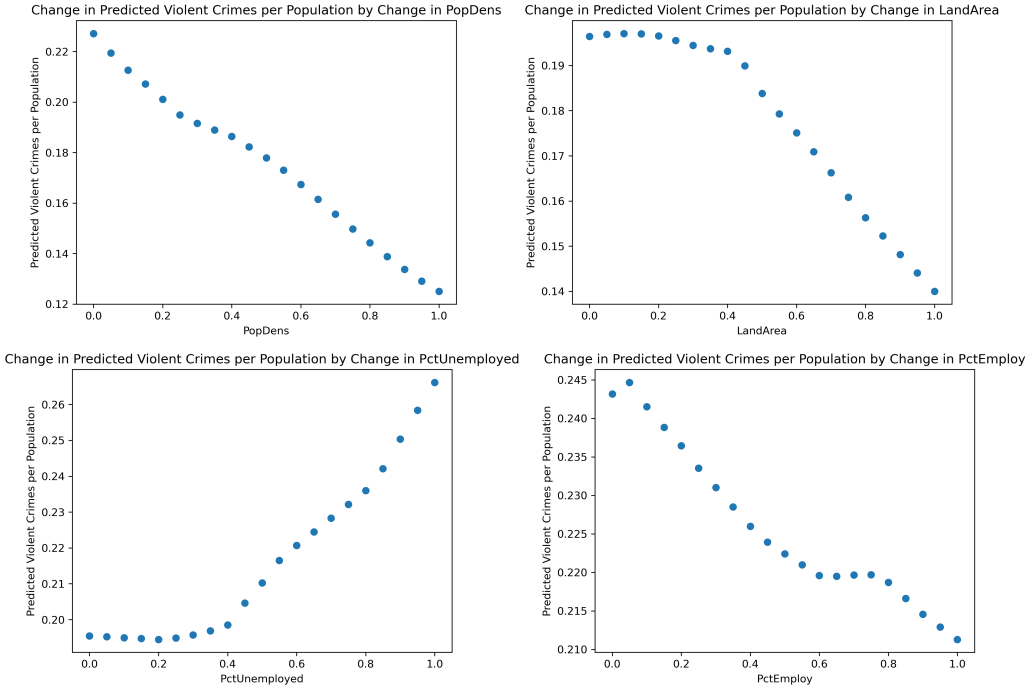

Figure 7: Predicted Violent Crime with Varying Features

4 Summary and Future Work

In this project, we were able to construct an accurate model of violent crime rates using a large variety of features. This model outperformed several other classical ML models and is able to provide insights about the relationship between the features and the violent crime rate outcome variable.

For this project, we specifically chose our candidate parameters in the 5-fold CV to ensure that the Python file did not take more than a few hours to run. In a full project, we would choose a greater range of classical ML models to compare as well as a larger parameter space to explore. In addition, for a longer-term project, we would want to find a better data imputation method. The `IterativeImputer` function is flexible and can be used with a variety of estimation methods (Pedregosa et al. 2011). We would explore alternative estimators and also consider non-parametric approaches, like the `KNNImputer`.

Aonther potential future improvement is to not only include more up-to-date information to make the model more relevant, but also include 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 examine how changes in a policy variable are associated with changes in violent crime rates.

A Appendix B

```
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()] > cutoff).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.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, et al. 2011. "Scikit-Learn: Machine Learning in Python." *Journal of Machine Learning Research* 12: 2825–30.

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.