Predicting Community Safety Through Machine Learning

Jake Blancher

***Abstract*—This project leverages machine learning algorithms to predict community safety using the 'Communities and Crime' dataset from UC Irvine's repository. A graphical user interface (GUI) is also implemented, allowing users to input neighborhood characteristics and receive safety predictions before relocating.**

GitHub Repo

I. Introduction

The average American moves houses more than 11 times over the course of their lifetime [Cha15]. When searching for a new home, the safety of the neighborhood in which you are moving into is of paramount importance. For this project, I analyze a UCI dataset containing over 120 features, with violent crime rate as the target variable.

Using a variety of techniques and machine learning algorithms, I hope to create a model that can predict the safety of a community by evaluating certain features. Additionally, I hope to simplify the model (without sacrificing much accuracy) to a point that an average American can understand and apply in their own lives when they are moving.

Determining neighborhood safety often relies on subjective reports, outdated crime maps, or inconsistent statistics. This project provides an objective, data-driven solution by leveraging machine learning. However, working with a high-dimensional dataset that includes socioeconomic, demographic, and law enforcement variables presents unique challenges, particularly in avoiding overfitting, ensuring interpretability, and maintaining fairness across different communities.

The final model is just 8 features, so it is simple and clear which variables are the most important in predicting neighborhood safety.

II. Feature Selection

A. Importance of Feature Selection

The first problem that came to mind when examining this data set is the fact that there are 127 features available. It would simply not be feasible to test the predictive accuracy of all subsets of these 127 characteristics, as $2^{127} - 1$ is an unfathomably large number ($1.70 * 10^{38}$, approximately equal to the amount of sand grains on the earth squared) [Kru12]. This number of iterations would not be complete in my lifetime, even with the most powerful computers available. So, a very important part of this project will be feature selection. Not only is this important to save computational resources, but it will also help with generalization (preventing overfitting) and interpretability.

B. Data Cleaning

First, I can easily remove non-predictive columns which, for example, include the state, county, and community name. Next, I would remove the columns that have missing data. There are 25 columns that are missing data for 1675 of the 1994 instances. The data is stored as a question mark, instead of a NaN. As such, I used np.replace to replace all question marks with NaN, so that I could subsequently use dropna to remove these. Since these columns only have data for about 16% of the instances, they are not worth considering, especially because we have an abundance of other viable features without missing data.

C. Principal Component Analysis (PCA)

In order to help reduce the dimensionality of the data, I considered using principal component analysis (PCA). However, as mentioned in the abstract, I created a simple GUI so that people can input the data of a prospective neighborhood, and the algorithm can predict whether this will be a low or high crime community. As such, I want the user to only have to input a handful of features. However, PCA will create a linear combination of the features, and is unlikely to actually reduce the number of features that the user will need to input. Thus, I am manually reducing the features, to both reduce dimensionality and make the user input to the GUI simple.

D. Feature Correlation to Target Variable

At this point, I had 99 features remaining. Notably, the data had already been normalized into a 0.00 to 1.00 range, using unsupervised equal binning. Thus, no further normalization would need to be done.

I then ran a pairwise correlation between each feature and the target variable. I then dropped all features that have a correlation between -0.5 and 0.5, as evidently they don't strongly correlate with the crime variable. This removed 84 features, meaning there were 15 features remaining.

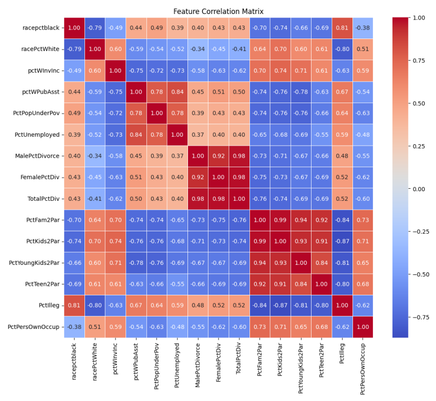
Figure 1: Initial Pairwise Correlation Matrix

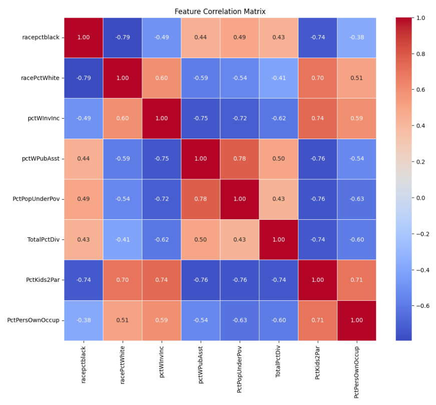
Figure 2: Final Pairwise Correlation Matrix

E. Pairwise Correlation between Features

I then ran a pairwise correlation between the 15 remaining features. The correlation matrix showed us that of the remaining features, there were a few pairs of features that were highly correlated and thus redundant and removable. The high correlation is represented by either the deep red (correlation close to 1) or deep blue (correlation close to -1). These features convey similar information and can introduce instability in models.

For example, the percent of total people divorced, the percent of males divorced, and the percent of females divorced are unsurprisingly highly correlated (Figure 1), each with a correlation of over 0.90 with one another. As such, I removed percent of male divorcees and percent of female divorcees. Another example is that the percentage of families headed by 2 parents, the percentage of kids with 2 parents, the percentage of young kids (under 4) with 2 parents, and the percentage of teens with 2 parents all have correlations over .90 with one another. So, I kept the percentage of kids with two parents feature (as it not only has the highest correlation with the target among the 4 aforementioned features, but all 99 features), and removed the other redundant variables.

Then, I was left with a correlation matrix with the 8 remaining features, as seen in figure 2. There are no two features remaining that are correlated greater than the absolute value of 0.8. The lasso regularizer in linear regression will further help with feature selection.

III. Linear Regression

A. Regularization

For the linear regression, I decided to use a Lasso Regularizer to make the parameters sparse, and further reduce the dimensionality of the data by eliminating relatively unimportant variables. Lasso regression is particularly effective in sparse settings, where the goal is not just predictive performance but also feature elimination. Its L1 penalty term forces some coefficients to zero, effectively removing irrelevant predictors. This aligns well with the project's goal of interpretability and simplicity — an essential feature when the end-user is expected to enter values manually into a GUI. I decided to try and find the optimal linear regression model for three sets of parameters: all 99 features, the 8 aforementioned pruned features, and the 2 most predictive features.

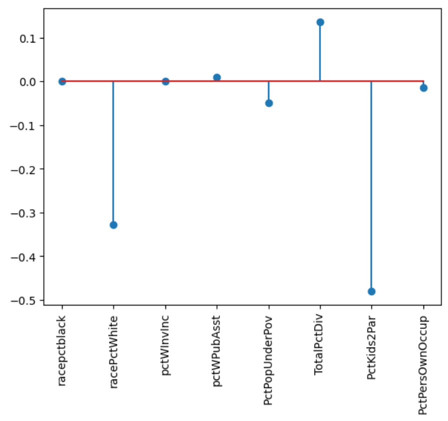

Figure 3: Stem Plot for 8 Features' Weights

B. Training and Testing

The first thing I did was split the data into testing and training, reserving 20% for the latter. I am going to use this same split for all three sets of parameters. Moving forward, I am going to follow the same process for all three sets. The next objective was to find the optimal alpha value, one that maximizes the r^2 values when using cross validation in the training data. I first tested 30 values over a log space between 10^{-5} and 10^{-1}. Then, among these first 30 tested values, I found the alpha that maximizes r^2. Then, I constrained the range to be within one order of magnitude of the best alpha, retrying 30 values of alpha. The alpha yielding the highest r^2 among these 30 values was selected as the final regularization parameter." Finally, I will use this alpha as a regularizer to train the model using the training data, and predict the test set, revealing the r^2 for the set of parameters.

Notably, when training the model with the eight best features, there were two weights (the second and seventh) that are significantly higher than the others. These weights correspond to the percent of Caucasian people and the percentage of kids that live with two parents, respectively. As such, I tested the impact of simplifying the model to just these two features. I considered including the sixth feature, Total Percent Divorced, but decided not to, as it has a high negative correlation with percent of kids that live with two parents.

Features	r^2 of Test Data
All 99 Features	0.6893
8 Selected Features	0.6332
2 Most Predictive	0.6387

Table I: r^2 for Selected Feature Sets.

Unsurprisingly, the test set r^2 decreased when reducing the number of features from 99 to 8, changing from 0.6893 to 0.6332, respectively. However, the r^2 actually increased when further reducing it from 8 features to 2, increasing to 0.6387. I figured, if I was going to reduce the features from 99 to 8 for the linear regression, I might as well reduce it even further and improve the r^2. As such, applying the principles of parsimony, I drastically reduced the dimensionality of the data (by 97 dimensions, from 99 to 2). Although I used just two features for the linear regression because their weights were so comparatively great, I plan to use the 8 feature set (or a subset of the 8 if desirable) moving forward for the classification algorithms and hence in the GUI. I fear the two feature set will be too simple and cause future models to underfit.

IV. Classification Algorithms

A. Binning

Now that the Linear Regression model was successfully implemented, I now needed a way to "bin" my data, converting the continuous "Violent Crime" target variable to a categorical variable. This binning will allow me to implement other types of machine learning algorithms. I decided to use K-means clustering with k=3, so that I could assign instances to be either "low crime" or "high crime" communities. Using K-means clustering allows the data to naturally segment itself into a high and low crime cluster, rather than forcing the data into two groups using either equal-width or equal-frequency binning. The binning is seen in figure 5, below. Additionally, making the

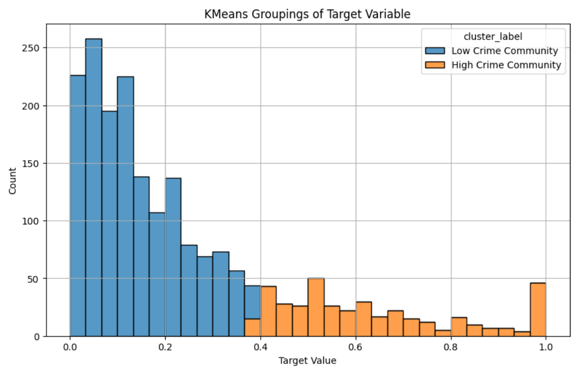

Figure 4: Histogram of Binned Crime Variable

target a binary variable significantly simplifies the algorithms, allowing for higher accuracy and lower loss. Notably, of the 1994 instances, 1593 of them (79.8%) were classified as low crime, meaning there is signifcant class imbalance.

B. Logistic Regression

The first thing I did when trying to make my logistic regression as robust as possible, was run a simple model. I initially used default weights and regularization strength (C) without balancing. This resulted in the scores seen on the top half of the table below. Across the board, the model did a good job of predicting the "0" class, which represents low-crime communities. However, it had a quite poor value for high-crime communities (class 1) recall and f1-score. I thought that this likely had to do with the aforementioned fact that class 1 only represented about 20% of the data. So, I passed the parameter "class_weight"="balanced," so that a higher weight would be put on classifying high crime communities.

Table II: Logistic Regression Accuracy

Before Class Balancing				
	Precision	Recall	F1-score	Support
0	0.89	0.98	0.93	317
1	0.87	0.55	0.67	82
Accuracy			0.89	399
Macro avg	0.88	0.76	0.80	399
Weighted avg	0.89	0.89	0.88	399
Class Weight = 'Balanced'				
	Precision	Recall	F1-score	Support
0	0.95	0.87	0.91	317
1	0.62	0.82	0.71	82
Accuracy			0.86	399
Macro avg	0.78	0.84	0.81	399
Weighted avg	0.88	0.86	0.87	399

The balancing of the class weights improved class 0's precision, class 1's recall, f1-score at the expense of class 0's recall, class 0's f1-score, and most notably the overall accuracy. I then considered what else I could do to improve the model. I decided on using cross-validation to try and optimize the value of C, which equals $\frac{1}{\lambda}$, also known as the inverse of the regularization strength.

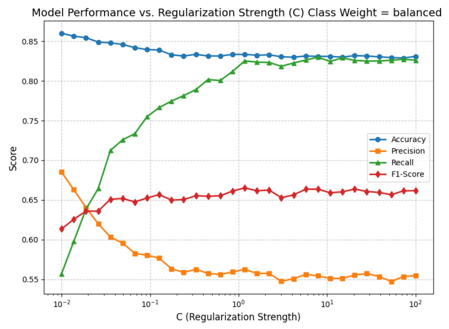

Figure 5: Cross Validation Testing Values of C With Class Weight Balanced

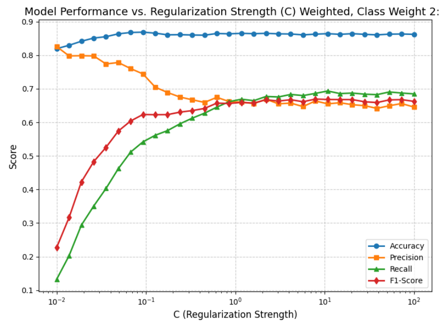

Figure 6: Cross Validation Testing Values of C With Class Weight 2:1

I first tested 30 values over a log space between 10^{-2} and 10^{2}. However, I noticed that no matter the value of C, the precision and f1-scores were quite poor. I theorized that using "class_weight"="balanced," would actually overweight the class 1 points, as it would assign them a weight of roughly 4 times the class 0 weight, since class 1 and class 0 represent about 80% and 20% of the data, respectively. As such, I assigned class 1 to be only double class 0's weight. Consequently, the precision, recall, and f1-score are all stable and roughly equal once C gets greater than 1. Thus, I settled on a C value of 4.64, the smallest C value once the metrics stabilize. This resulted in the final metrics seen below, when using a logistic regression model with a weights ratio of 2:1 and C=4.64.

Table III: Classification Report for Final Logistic Regression Model

Class Weight 2:1

	Precision	Recall	F1-Score	Support
Class 0	0.93	0.95	0.94	317
Class 1	0.80	0.73	0.76	82
Accuracy			0.91	399
Macro Avg	0.87	0.84	0.85	399
Weighted Avg	0.90	0.91	0.91	399

The final model shows a marked improvement in overall performance compared to the version after class balancing. While the recall for high crime communities (Class 1) decreased slightly from 0.82 to 0.73, precision significantly improved from 0.62 to 0.80, reducing false positives. The F1-score for Class 1 also increased from 0.71 to 0.76, indicating a better balance between precision and recall. Additionally, overall accuracy rose from 0.86 to 0.91, and both macro and weighted F1-scores improved, reflecting a more equitable and effective classification across both classes. This suggests the final model achieves a stronger balance between detecting high crime areas and minimizing misclassifications.

C. Support Vector Classifier (SVC)

The next classification algorithm I used was a support vector classifier (SVC). The first SVC that I ran used default C, gamma, and class weight balancing. Consequently, the results were quite poor, as seen in table IV.

Table IV: Initial Confusion Matrix for SVC

True Label \ Predicted Label	**0**	**1**
0	0.99 (TN)	0.013 (FP)
1	0.63 (FN)	0.37 (TP)

Again, we see the effects of the class imbalance. The model did a quite poor job at predicting the positive high-crime communities, accurately classifying only 63% of data points. As such, I tested various weights to assign to the minority class. I want to prioritize minimizing false negatives. It would be more damaging if my model predicts that a community was safe when it was dangerous, than if the model predicts a dangerous neighborhood when it was safe. So, I am going to value specificity higher than sensitivity. Thus, I chose a minority class weight of 4.

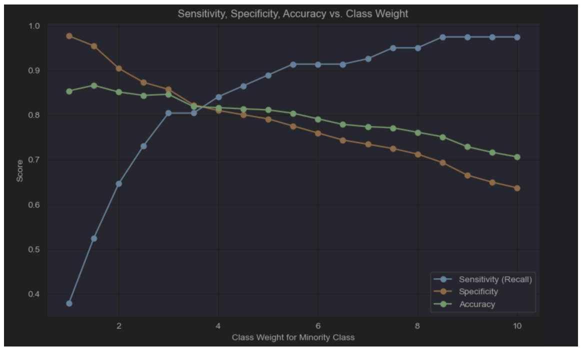

Figure 7: Minority Class Weight vs. Accuracy Metrics

Next, I wanted to optimize the values of C and gamma. I decided to test 30 values for each parameter, with C between 10^{-1} and 10^{-1} and gamma between 10^{-4} and 10^{-1}. So, I created a 30 x 30 array, iterating through the 900 combinations to find the optimal pair, settling on C = 4.52 and gamma = 0.079. Combining this with a minority class weight of 4, I was able to get the confusion matrix below.

Table V: Final Confusion Matrix for SVC

True Label \ Predicted Label	**0**	**1**
0	0.80 (TN)	0.20 (FP)
1	0.13 (FN)	0.87 (TP)

Between the two models, Model 2 is clearly better suited for prioritizing Class 1, where missing a high crime community (false negative) is more critical than incorrectly labeling a low crime one (false positive). Model 1 has a high false negative rate (0.63), meaning it fails to detect most high crime areas, despite having a very low false positive rate (0.013). In contrast, Model 2 reduces false negatives significantly to 0.13 and correctly identifies 87% of high crime communities (true positives), making it far more effective for the intended purpose. Although Model 2 has a higher false positive rate (0.20), this is an acceptable trade-off in this context. Therefore, Model 2 is the preferred choice for applications where identifying high crime areas accurately is critical.

D. Neural Network

The neural network that I implemented in this project is a straightforward feedforward architecture designed for binary classification. It consists of an input layer that takes in the eight selected features selected from the dataset, followed by one hidden layer with 100 neurons. A sigmoid activation function is applied after the hidden layer to introduce non-linearity, which helps the model learn complex patterns in the data. The output layer contains a single neuron, and its output is treated as a logit, passed directly to the loss function without additional activation.

The loss function I chose for this model is BCEWithLogitsLoss, which is particularly well-suited for binary classification tasks. It combines a sigmoid activation and binary cross-entropy loss in a numerically stable way, reducing the risk of vanishing gradient. Again the class 1 data points were assigned a weight of four, as they only comprise of about 20% of the dataset.

The first thing I had to select was the learning rate. I tested three values 10^{-4},10^{-3}, and 10^{-2}. The loss is shown in figure 10.

Based on its results, I decided to use a learning rate equal to 10^{-3}, as it converges quickly (unlike 10^{-4}) without the instability of overshooting like a rate of 10^{-2}.

Applying this learning rate over 100 epochs, resulted in the test accuracy seen in figure 11.

We see the test accuracy a bit volatile in the first few epochs, before stabilizing, resulting in a final test accuracy of 81.7%.

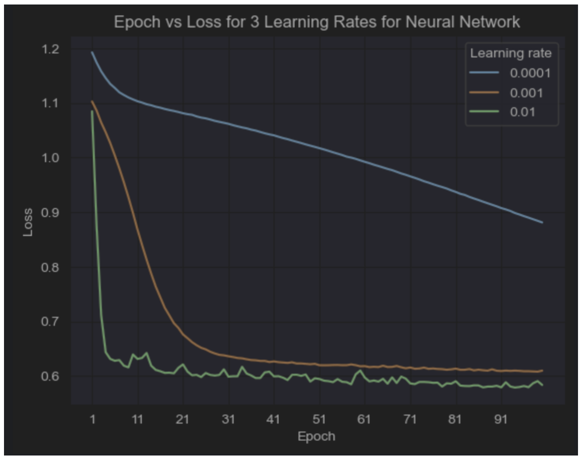

Figure 8: Neural Network Learning Rate vs Loss

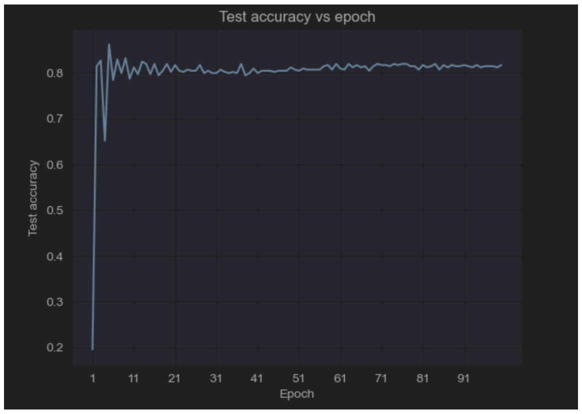

Figure 9: Neural Network Learning Rate vs Loss

E. Deep Neural Network

The main differences between the original neural network (NN) and this deep neural network (DNN) that I created lie in their depth, activation functions, and regularization. The NN consists of a single hidden layer with a Sigmoid activation function, which is simple but can lead to vanishing gradient problems and limited representational power. In contrast, the DNN includes multiple hidden layers (three) with ReLU activations, which are more effective for deep learning by enabling better gradient flow and faster convergence. Additionally, the DNN incorporates dropout layers for regularization, helping reduce overfitting by randomly deactivating neurons during training. The DNN also has significantly more parameters (1001 as compared to 43,521), giving it higher capacity to model complex patterns in the data.

To be consistent, I decided to continue to use a learning rate 10^{-3}, minority class weight of 4, and iterating over 100 epochs. The accuracies are shown in figure 12.

This DNN model had a final test accuracy of 86.7%, about 5% improvement from the NN model.

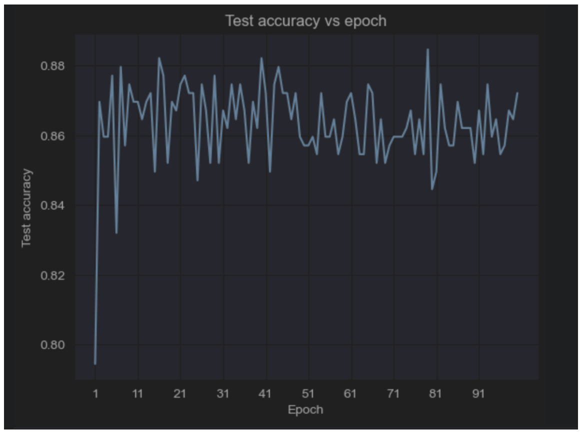

Figure 10: Deep Neural Network Learning Rate vs Loss

V. Ensemble Learner

In order to create the strongest learner, I then created an ensemble of the models I have already created. I decided to use Logistic Regression, SVC, and Deep Neural Network. I excluded the Neural Network for two reasons. Primarily, it is just a simpler and less accurate version of the Deep Neural Network, so including it would be redundant. Second, it would cause the ensemble to have four models, which may cause complications in prediction if two learners predicted class 0 and two learners predicted class 1.

Though likely a faulty assumption (because the models are likely dependent, in the sense that they are likely to overlap in the instances that they misclassify), if assume the models were independent with accuracies of 86.0%, 84.0%, and 86.7% (corresponding to the Logistic Regression, SVC, and DNN, respectively). The ensemble would only misclassify a data point if all 3 models predict them incorrectly, which would occur $(1 - .860) \times (1 - .840) \times (1 - .867) = 0.0029792$, equal to a roughly .3% error rate and a 99.7% accuracy.

However, in reality, the ensemble accurately predicted 88.5% of the test set. This is, of course, below the theoretical accuracy, but it is still almost 2% more accurate than the DNN, the best singular model.

VI. Graphical User Interface (GUI)

The last step of my project was creating a GUI that deployed my ensemble so that a user could predict the crime in their neighborhood. I considered learning Django or Flask so that it could be hosted online. However, I simply didn't have enough time to properly learn a web application framework, so I resorted to the only GUI software I have experience with, Tkinter. I created a bare-bones interface, where users input the values for their eight features, and it outputs both a classification (high or low crime) and a continuous prediction of crime (violent crimes per 100k people). I created a function that converted the raw un-normalized user-input percentages to the normalized values to be used in the algorithms, and one that converted the normalized regression output, to an normalized crimes per 100k value.

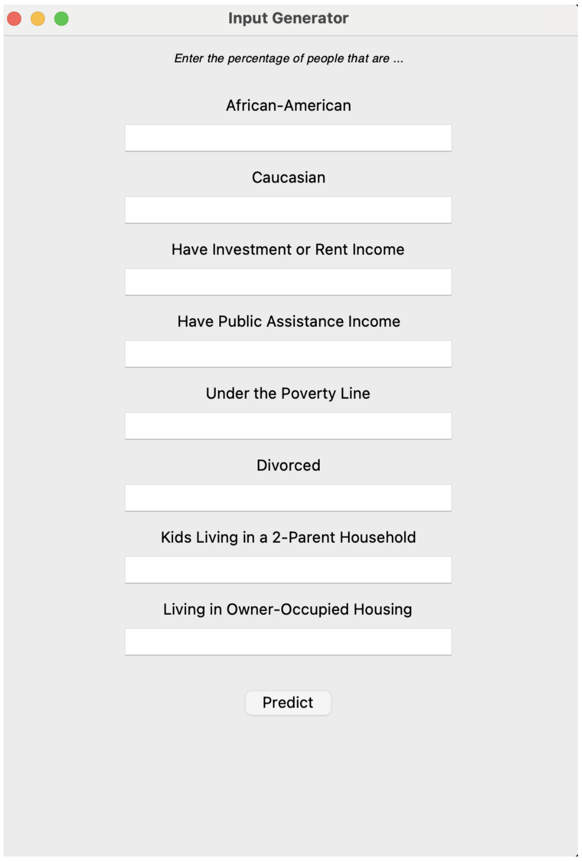
Input Generator
Enter the percentage of people that are ...
African-American
Caucasian
Have Investment or Rent Income
Have Public Assistance Income
Under the Poverty Line
Divorced
Kids Living in a 2-Parent Household
Living in Owner-Occupied Housing
Predict

Figure 11: Tkinter GUI

VII. Discussion

A. Ethical Consideration

Upon examining my 8 pruned features, there are two features, ”racepctblack” and ”racepctWhite”, that could certainly be considered contentious. One might erroneously interpret this dataset as suggesting that predominantly white communities are inherently safer, while minority communities are inherently less safe. For one, this data is, of course, purely correlational and does not imply causation. Additionally, that claim is not only remarkably prejudiced, but simply untrue. Majority minority communities are likely to be victims of over-policing. One study, focused on the policing in Chicago, found that within the ”Near North” precinct, just 7.9% of residents were Black yet 73.5% of investigatory stops targeted Black Chicagoans. Similarly, Black Chicagoans cars were 3.3 times more likely to be searched than White people's cars [Che22]. Bias against Black Americans is not only found in police forces, but also in juries. A 2012 study found that all-White juries were 16% more likely to convict a Black defendant than a White defendant [ABH12]. Thus, the higher crime metrics in majority minority neighborhood is not due to an inherent biological predisposition to crime, but rather a function of law enforcement and juries' distrust and disproportionate force used upon minority communities.

B. Future Work

While the models developed in this study offer strong predictive performance and simplicity, there are a few limitations. First, the dataset is from the 1990s and may not reflect contemporary trends in crime or urban development. Second, simplifying to only a few features may exclude important but subtle interactions between variables. Finally, ethical considerations must be made to ensure that models do not perpetuate systemic biases or stigmatize communities based on socioeconomic indicators.

Future improvements include incorporating up-to-date datasets, experimenting with other models (like those from Data Mining such as KNN, Naive-Bayes, Random Forests, etc), and integrating geospatial data to further enhance predictive accuracy. Additionally, a web-based implementation of the GUI (using Django or Flask) could make the tool widely accessible.

References

[ABH12] Shamena Anwar, Patrick Bayer, and Randi Hjalmarsson. The impact of jury race in criminal trials*. *The Quarterly Journal of Economics*, 127(2):1017–1055, 04 2012.

[Cha15] Mona Chalabi. How many times does the average person move?, Jan 2015.

[Che22] Heather Cherrone. Chicago police more likely to use force against black chicagoans: Watchdog, Mar 2022.

[Kru12] Robert Krulwich. Which is greater, the number of sand grains on earth or stars in the sky?, Sep 2012.