# DS-GA 1019: Advanced Python for Data Science Optimizing Models for Predicting Bullying in Adolescents

Yasi Asgari	gari YA2193@NYU.ED	
Victor Cui	VYC8567@NYU.EDU	
Aryann Dharamsey	asd484@nyu.edu	
Nikki Gharachorloo	nng235@nyu.edu	
Alex Herron	AH5865@NYU.EDU	
Hailie Nguyen	HLN2020@NYU.EDU	

# 1. Background

According to research done by cyberbullying.org, the frequency of bullying in schools increased by 35% between 2016 and 2019. Traumatic stress reactions, such as Post-Traumatic Stress Disorder, can result from bullying. According to the National Child Traumatic Stress Network, 27.6% of male students and 40.5% of female students among all bullied students had PTSD levels that fell into the clinical range.

Bullying must be stopped in order to avoid these consequences. To decrease the number of students experiencing bullying, we aim to predict if a student will be bullied using machine learning algorithms written from scratch, then optimize those algorithms for speed.

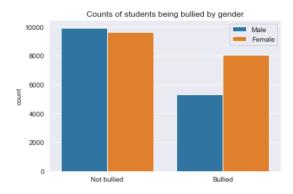


Figure 1: Distribution of Bullying by Gender

#### 2. Data

#### 2.1. Kaggle Dataset

The dataset used for our project was collected by the Global School-Based Student Health Survey (GSHS), focusing on bullying among the youth population. The data was collected over a self-administered questionnaire in schools in Argentina in 2018 on 56,981 students, with an overall response rate of 63%.

There are 18 columns in the dataset. Some key features included age, sex, weight, whether bullying occurred within the past year, whether bullying occurred on school grounds, whether the parents are sympathetic, the number of missed school days, and the number of close friends of a given student.

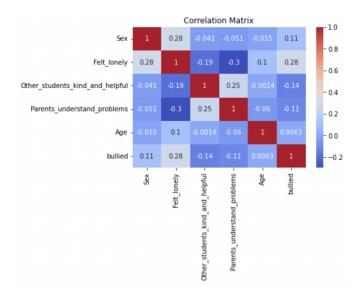


Figure 2: Correlation Matrix of Features in Dataset

## 2.2. Data Preprocessing

There was a significant amount of missing data for this dataset. The vast majority of these missing values were unable to be imputed with median and mean values due to the categorical options for the survey questions. As a result, the rows with null values have been removed, leaving 32,938 records available for modeling purposes.

In addition to removing rows with null values, we also converted certain cateogorical variables to integers. For example, frequency categories were converted in the following way: Never - 0, Rarely - 1, Sometimes - 2, Most of the time - 3, Always - 4. Additionally, we created a target variable for any kind of bullying (accounting for if the student was either bullied on-campus, off-campus, or cyber-bullied).

Finally, we dropped the ID column, which left us with 14 predictors and 4 potential targets. We explore key features in our data distribution in the next section.

#### 2.3. Exploratory Data Analysis

Our data shows that 40.6% of the students reported being bullied. Our class distribution (by bullying type) is relatively balanced. 20.9% of the students reported being bullied on school property, 22.6% reported being bullied outside of school, and 22.6% reported being cyber-bullied in the past 12 months. It's important to note that these groups can overlap. 2,216 students experienced all three types of bullying (in school, outside of school, and online) over the course of a year.

Plotting the variable "bullied" (which includes any bullying type) against various predictors, we can see that some features might play an important role in predicting whether a student is bullied. For example, the proportion of female students being bullied appears about 1.5 times higher than that of male students (Figure 1). A correlation matrix shows that the most correlated predictors of bullying are gender, whether parents understand the student's problems, whether other students are helpful, and whether the student feels lonely (Figure 2). Age does not seem to be a useful predictor of bullying.

Lastly, we calculated variance inflation factors (VIF) to detect multicollinearity among the independent variables. All variables have an VIF of less than 3, which suggests to us that there is no significant multicollinearity that needs to be corrected.

## 3. Implementation

## 3.1. Modeling with Scikit-learn

For our preliminary modeling effort, we used the following Scikit-learn models: logistic regression, decision tree classifier, support vector machine, random forest classifier, and XGBoost classifier. These models were trained and tested using a 80:20 split of the data. While we ran these models on all three types of bullying and total bullying, we decided to focus on modeling total bullying from this section onwards. Making predictions for total bullying had the highest AUC, and it would be the most valuable for parents and school administrators to accurately predict total bullying than predict specific kinds of bullying with lower predicting power. Table 1 shows the metrics and running speed for all implemented Scikit-learn models.

## 3.2. Modeling from scratch

We decided to implement logistic regression and decision tree algorithms from scratch. We chose logistic regression for its simplicity and high predicting power (which was the highest out of the Scikit-learn models selected). Additionally, we chose to create a decision tree model, as its increased complexity provided additional pathways for speed improvement. See Table 2 and 3 for exact model speed and accuracy.

#### 3.2.1. Logistic Regression Model

We implemented logistic regression from scratch with a basic full-batch gradient descent approach using a sigmoid activation function, with all matrix/vector computations done in Numpy. During optimization, we set the learning rate to 0.0001 with 100,000 iterations. We used line\_profiler to identify that our gradient descent implementation is taking up over 99% of runtime (Figure 3 in Appendix). We attempted to optimize it using Numba, which improved the speed by 17.73%. We further attempted to change our approach from full-batch gradient descent to mini-batch stochastic gradient descent (SGD). This allowed us to improve the speed by 96.40% without compromising accuracy. Using line\_profiler on our new approach, we identified that the Numpy.randint function used to pick random samples at the beginning of each mini-batch was a new bottleneck. We further optimized it by instead only calling the function once every epoch. After adapting this optimized version of SGD in Numba, we were able to improve the model speed by another 90.71%, which in total is 99.67% faster than our original implementation. Importantly, the model accuracy stayed consistent after optimization. We also attempted threading, model input normalization, and matrix multiplications using Numba.CUDA kernels instead of Numpy, including using shared memory, but none of them made any noticeable improvement in model performance.

#### 3.2.2. Decision Tree Model

We also created a decision tree model from scratch, which we optimized using line\_profiler as a starting point. In our initial implementation, we calculated the information gain for every possible split. Line\_profiler identified the evaluate\_split() function as taking up 97% of runtime (Figure 4 in Appendix). We attempted to improve its speed by generating random binary splits of the data. This reduced the number of splits that need to be evaluated, while still allowing the algorithm to find a diverse range of splits. This improved speed by 81.41%. We also attempted to combine random split with threading for both the evaluate\_split() and build\_decision\_tree() functions. While our performance improved by 74.12% compared to our initial implementation, threading did not help speed up our random split implementation. Finally, we added parameters for the number of features and number of values for our decision tree. This, combined with random split, allowed us to improve our speed by 98.68% compared to our initial implementation. Lastly, we performed hyperparameter tuning and concluded that num\_features = 2 and num\_values = 2 provided the best combination of predictive power and speed. Our model accuracy before and after optimization is comparable to Scikit-learn results.

#### 4. Outcome

#### 4.1. Modeling with Scikit-learn

Table 1: Scikit-learn Modeling Predicting Any Kind of Bullying

Model	Speed (s)	Accuracy	Precision	Recall	F-1 Score	AUC
Logistic Regression	0.094	0.671	0.633	0.417	0.503	0.629
Decision Tree	0.069	0.625	0.536	0.439	0.482	0.593
SVM	38.142	0.667	0.653	0.350	0.456	0.614
Random Forest Classifier	1.689	0.631	0.542	0.484	0.511	0.606
XGBoost Classifier	0.782	0.658	0.599	0.428	0.499	0.619

# 4.2. Modeling from scratch

## 4.2.1. Logistic Regression Model

Table 2: Optimizing Logistic Regression Models from Scratch

Model Variant	Speed (seconds)	Accuracy
Standard Logistic Regression from Scratch	50.820	0.671
Stochastic Gradient Descent	1.830	0.671
Stochastic Gradient Descent + Normalized Inputs	1.870	0.670
Stochastic Gradient Descent + Threading	38.385	0.671
Numba	41.810	0.671
Stochastic Gradient Descent + Numba	0.170	0.670

### 4.2.2. Decision Tree Model

Table 3: Optimizing Decision Tree Models from Scratch

Model Variant	Speed (seconds)	Accuracy
Standard Decision Tree from Scratch	267.235	0.604
Evaluation Splits on Random Feature Subset	49.655	0.625
Threading	370.378	0.622
Threading + Random Feature Subset	69.144	0.622
Add Params for Features/Values + Random Feature Subset	3.513	0.640

# 5. Summary

Throughout our modeling process, we realized the challenge in accurately predicting which students would be most vulnerable to bullying. The highest AUC achieved was 0.629 for logistic regression. Bullying could be related to a variety of factors, which might not necessarily be captured in our dataset. When comparing our various optimization techniques, we found that our logistic regression model optimized using SGD in combination with Numba gave us the best improvement in speed, allowing our runtime for logistic regression from scratch to be improved by 99.67%. Additionally, we found that we could improve our speed for the decision tree model by 98.68% using random subsets of features as well as utilizing adjustable numbers of features and variables. Potential future work could be done using the techniques described above on additional algorithms, such as support vector machine or random forest, and comparing their performance with the models above or Scikit-learn implementation.

# 6. Appendix

```
Line #
                        Hits
                                                  Time Per Hit
                                                                                % Time Line Contents
                                                                                                          def fit(self, X, y):
    if self.fit_intercept:
        X = self.__add_intercept(X)
                                    10000.0 10000.0
10880000.0 10880000.0
                                                                                            0.0
                                                                                                                  # Initialize the weights
self.theta = np.zeros(X.shape[1])
                             1
                                           29000.0 29000.0
                                                                                                                   # Gradient descent
                                                                                                                 # Gradient descent
for i in range(self.num_iter):
    z = np.dot(X, self.theta)
    h = self._sigmoid(z)
    gradient = np.dot(X.T, (h - y)) / y.size
    self.theta -= self.lr * gradient

    100000
    72649000.0
    726.5

    100000
    844905000.0
    84490.1

    100000
    35674645000.0
    356746.5

    100000
    19988088000.0
    199880.9

                   100000
                                                                                        1.3
                   100000 869217000.0 8692.2
                     99990 113912000.0
                                                                                                                          if i % 10000 == 0:
                                     2016000.0 201600.0
3395000.0 339500.0
                                                                                                                                  z = np.dot(X, self.theta)
h = self.__sigmoid(z)
                       10
10
```

Figure 3: Logistic regression line\_profiler results (original implementation)

Figure 4: Decision tree line\_profiler results (original implementation)

## 7. References

- 1. https://www.kaggle.com/datasets/leomartinelli/bullying-in-schools
- 2. https://cyberbullying.org/school-bullying-rates-increase-by-35-from-2016-to-2019
- 3. https://news.illinois.edu/view/6367/462003792
- 4. https://www.nctsn.org/what-is-child-trauma/trauma-types/bullying/effects
- 5. https://www.stopbullying.gov/bullying/bullying-and-trauma
- 6. https://www.pacer.org/bullying/info/stats.asp