Exploring eviction in Fragile Families data

Gregory Gundersen
Princeton University
ggundersen@princeton.edu

Abstract

We explore eviction in the Fragile Families and Child Wellbeing Study dataset and find that it is difficult to predict this outcome. In the training dataset, eviction is reported just 2% of the time. As a result, most classifiers have high accuracy and low mean squared error but their precision and recall scores are typically less than 10%. We experimented with a variety of different feature selection techniques, classifiers, hyperparameters, and class imbalance methods such as oversampling and outlier detection. No combination we tried resulted in markedly improved performance. We present our efforts in detail in hopes that it clarifies the problem for future researchers.

1 Introduction

1.1 Background

Eviction is a traumatic experience that often forces children to move schools and lose friends, consumes a parent or caregiver's energy, and impacts access to credit and public housing. Matthew Desmond, a sociologist at Harvard who studies eviction in the United States, argues that eviction is not just caused by poverty but is a cause of poverty. The problem is also widespread. For example, Desmond estimates that 1 in 8 renters in Milwaukee experience a forced move every 2 years [2].

In this project, we explore eviction in the Fragile Families and Child Wellbeing Study (FFCWS) dataset. The FFCWS is a longitudinal study of 4242 children born between 1998 and 2000 in cities in the United States. A "fragile family" is one in which the parents are unmarried at the time of the child's birth and are statistically at greater risk of both separating and living in poverty [5]. The study consists of interviews of the child's mother, father, and caregiver (if not the mother or father) at time of birth and then follow up surveys at ages 1, 3, 5, 9, and 15. The data from the survey at age 15 has been held out as testing data, and the FFCWS researchers have challenged the scientific community to fit models to the training data and then make predictions of one, several, or all of six response variables: GPA, grit, material hardship, eviction, layoff, and job training [6]. The goal is to discover what makes some children succeed and to help those who are at greater risk.

In this paper, we explore eviction. The outcome for eviction was measured from the following question, which was asked to each child's primary caregiver during year 15:

> Since MONTH AND YEAR COHORT CITY FIELDED IN YR 9, were you evicted from your home or apartment for not paying the rent or mortgage?

The possible answer choices were "Yes", "No", "Refused", and "Don't know". Those who refused or did not know where both encoded as "NA" [4].

2 Methods

Throughout this paper, we refer to the scripts used for different processing steps. The code can be found at: `https://github.com/gwgundersen/fragile-families`.

2.1 Data processing

Description. The FFCWS training dataset consists of 4242 children and 12,945 features. The training dataset has 2121 samples, while the other 2121 samples were held out for testing. Since we were only intereseted in predicting eviction, we dropped any samples for which the response variable was unknown. This reduced the number of training samples from 2121 to 1459. Eviction is an extremely rare event in the FFCWS dataset. Out of the 1459 training samples, only 87 respondents reported being evicted. Features are primarily questions given to the father, mother, or primary caregiver of the child at years 1, 3, 5, 9, and 15. Two example questions, one for the mother and another for the father, are:

> (m2b15) Since birth how many times have you and child been separated for a week or more?
>
> (f4k19) How much did you earn from all reg jobs in past 12 months (amount)?

The data matrix feature labels are encoded, e.g. “m2b15” and “f4k19” above, but can be translated to English using FFCWS-provided ”cookbooks”. We wrote a script to map codes to descriptions for easy, on-the-fly decoding (Script: `preprocess.descriptions`).

Cleaning. 811 columns had all missing values; these were dropped. Columns with categorical data were converted to numerical values using the `pandas` (version 0.19.2) [7] built-in `factorize` function (Script: `preprocess.factorize`).

Imputation. The data matrix is missing 9,614,356 out of 54,912,690 values. This means that roughly 17.5% of the data is missing. Missing data was imputed in the following manner. If a column contained numerical values, the mean value was used. If a column contained non-numerical values, the mode value was used (Script: `preprocess.impute`).

2.2 Feature selection

In order to explore the effects of feature selection, we created four different datasets with features selected in different ways:

- **FULL.** The FULL dataset contains all features after data imputation and cleaning.
- **RLOG.** Randomized logistic regression is a method in which the training data is randomly subsampled many times and fit with a logistic regression model with an L1 penalty for sparsity. The features which appeared most frequently as coefficients in the fitted models were then used as features in the RLOG dataset. Only 15 features were nonzero. See Table 1 for results (Script: `features.rlogistic`). The left column indicates the percentage of time that that feature was selected for by logistic regression. Only one feature was fitted more than a quarter of the time.
- **HC.** The “handcrafted” HC dataset was generated by selecting only questions which contained certain keyword stems. We used `grep` to quickly search for the set of feature codes and their descriptions (See `data/feature_codes_to_names.txt` in the GitHub repository). For example, the stem “abus” would return questions that contained both “abusive” and “abuse”. See Table 2 for word stems and their frequencies in questions. The total number of unique features was 970. (Script: `features.handcrafted`)
- **PCA.** The PCA dataset contains features which are the principal components from Principal Component Analysis (PCA) performed on the FULL dataset. The number of components was programmatically chosen by finding the minimum number of components needed to explain 99% of the variance in the data (Script: `features.pca`).

%	Feature code	Feature question
0.005	m2j15a	During those 2 wks, did you feel more tired and low on energy than usual?
0.005	m3b32g	Days/week: tell stories to child?
0.005	m3f2d10	What is tenth person's relationship to you?
0.01	m2j15c	During those 2 wks did you have more trouble falling asleep than usual?
0.01	m2j15f	During those 2 wks, did you think a lot about death?
0.01	m3a3e	About how many days did you see child in past 30 days?
0.015	m2j15b	During those 2 wks, did you gain or lose weight?
0.015	m2k17e1	How much did you receive past 12 mths working under the table?-($ Range)
0.02	m3f2f2	Is second person currently working?
0.035	m3a3b1	Does (person in A3A) receive any payment for taking care of child?
0.04	hv4l50	(He/She) steals outside the home
0.11	m3a3c	About how many months has child been living there? (months)
0.115	ffcc_centsurvey_b2	How many different classrooms has (CHILD) been in since first starting here?
0.125	hv3pvnceil	If no PPVT ceiling value was reached (child).
0.305	m4a3	How many months ago did he/she stop living with you (monthst of the time)?

Table 1: Features that were selected a nonzero number of times using `RandomizedLogisticRegression` in `scikit-learn` (version 0.18.1) [8]. The percentage indicates the percent of times that that feature was selected for out of all randomized trials.

Abus	Boyfriend	Disab	Employ	Evict	Gun	Job	Money	Mortgage	Pregnan	Rent	Violen	Weapon	Welfare
34	8	28	79	9	5	310	266	30	29	43	32	9	181

Table 2: Word stems and their frequencies in questions.

2.3 Classifiers

We ran experiments with the following `scikit-learn` classifiers. Unless otherwise mentioned, the default hyperparameters for each classifier were used:

- **LogisticRegression** (LR): Regression with binary classification. The `class_weight` hyperparameter was used with weights 0.94 and 0.06 for classes 0 and 1 respectively.
- **LinearSVC** (LSVM): Support vector machine with linear kernel. The `loss` hyperparameter was set to `hinge` for standard SVM loss. The `class_weight` hyperparameter was used with weights 0.94 and 0.06 for classes 0 and 1 respectively.
- **GaussianNB** (GNB): Naive Bayes classifier that assumes features are drawn from a Gaussian distribution. Class priors of 0.94 and 0.06 were used.
- **RandomForestClassifier** (RFC): Ensemble method that builds multiple decision trees and outputs the label which is the mode of all the trees. We used hyperparameters of `n_estimators` set to 1000 and `max_depth` set to 10. These hyperparameters were chosen using cross validation in an attempt to avoid overfitting.

We also ran experiments on our own neural network implementation which was developed for Princeton's COS 402: *Artificial Intelligence*:

- **NeuralNetwork** (NN): A feedforward artificial neural network for nonlinear function approximation. The neural network had 3 layers. For each dataset, the number of neurons in the input layer was the number of features; the number of neurons in the hidden layer was roughly half the number of features; and the number of neurons in the output layer was 2 (evicted or not)

The `scikit-learn` models and their configurations can be found in module `models`. The neural network implementation can be found in `models.neural_network` and `predict.nn`.

	LR			LSVM			GNB			RFC			NN		
Dataset	Acc	Pre	Rec	Acc	Pre	Rec	Acc	Pre	Rec	Acc	Pre	Rec	Acc	Pre	Rec
FULL	0.891	0.036	0.035	0.832	0.040	0.084	0.650	0.048	0.263	0.950	0.000	0.000	0.938	0.000	0.000
RLOG	0.948	0.000	0.000	0.946	0.250	0.028	0.896	0.215	0.245	0.938	0.250	0.040	0.931	0.100	0.013
HC	0.941	0.000	0.000	0.842	0.068	0.071	0.455	0.053	0.608	0.955	0.000	0.000	0.941	0.000	0.000
PCA	0.700	0.036	0.206	0.709	0.047	0.193	0.655	0.104	0.367	0.938	0.000	0.000	0.943	0.000	0.000

Table 3: Accuracy (Acc), precision (Pre) and recall (Rec) on datasets all features (FULL), features generated by PCA (PCA), handcrafted features (HC), and features generated by randomized logistic regression (RLOG).

	LR			LSVM			GNB			RFC		
Dataset	Acc	Pre	Rec	Acc	Pre	Rec	Acc	Pre	Rec	Acc	Pre	Rec
FULL-ov	0.827	0.036	0.064	0.790	0.039	0.139	0.660	0.083	0.367	0.937	0.000	0.000
RLOG-ov	0.942	0.000	0.000	0.945	0.000	0.000	0.897	0.207	0.279	0.901	0.123	0.115
HC-ov	0.899	0.104	0.047	0.599	0.058	0.398	0.589	0.073	0.520	0.936	0.000	0.000
PCA-ov	0.722	0.045	0.220	0.516	0.071	0.537	0.481	0.065	0.583	0.840	0.025	0.046

Table 4: Accuracy (Acc), precision (Pre) and recall (Rec) on datasets all features (FULL), features generated by PCA (PCA), handcrafted features (HC), and features generated by randomized logistic regression (RLOG).

3 Experiments

3.1 Standard prediction

Given the class imbalance, we first predicted all zeros to check for a baseline mean squared error (MSE). This gave us 0.0566 on the Fragile Families competition form (username: `ggundersen`). No other method gave us a superior MSE score, so we switched to measuring the accuracy, precision, and recall.

We could only verify the mean squared error of our predictions using the Fragile Families Challenge website [6], so we created our own test dataset. Following the convention of an 80/20 split, we divided the provided training dataset into two datasets, one for training and another for testing. The new training dataset has 1167 samples while the new test dataset has 292 samples. We then fit each classifier in Section 2.3, made predictions, and then computed the accuracy, precision, and recall. We performed each experiment 5 times and averaged the scores for each trial. We performed these experiments on all four datasets mentioned in Section 2.2. The results are in Table 3 (Script: `predict.standard`). No classifier–feature-set combination gave us decent results.

3.2 Prediction with oversampling

A common solution to the problem of class imbalance is to generate synthetic samples until the proportion is split 50/50. A popular algorithm for oversampling is SMOTE (Synthetic Minority Oversampling Technique) [1]. We used the implementation of SMOTE from `imbalanced-learn` [3], a Python library with many algorithms for dealing with imbalanced classes. For each experiment, we split the original training dataset into 80/20 training and testing. We then oversampled the new training dataset and tested on the held out data that had not been oversampled. The results are in Table 4.

3.3 Outlier prediction

A common solution to imbalanced classes in outlier or anomaly detection. The logic is that rather than train a model with very few examples of one class, train a model only on a single class and then look for outliers. We use performed experiments using scikit-learn's `OneClassSVM` and `IsolationForest` with default parameters for both. In both cases, the performance was not better than what we had seen in previous experiments, and the results are not shown here.

4 Discussion

In this work, we did not find any method that performed well at predicting eviction from the FFCWS dataset. But our results do suggest areas for future work:

- **More real samples of eviction.** According to Matt Desmond's work, eviction rates should be much higher than 6% of the data set. Perhaps following up with parents and caregivers would help fill missing response variables. Perhaps some people's notion of eviction is less strong than Desmond's, which is any forced move, even if it is not one forced through the court system.
- **Better feature selection.** A domain expert could provide feedback on the best way to perform feature selection. Randomized logistic regression suggests that children who moved between households are more likely to be evicted in the future. Perhaps we can better capture this in our feature selection.
- **Better imputation.** To impute missing values, we took the mean of numerical values and the mode of categorical values. One can easily imagine scenarios in which this greatly biases a column accidentally. For example, a single outlier can dramatically affect the mean. Or if a column is missing many values, replacing them all with the mode may skew the data. Taking more time to properly impute data is promising given how much data is missing.
- **Precision and recall on real test data.** In this work, we used our own test dataset because the Fragile Families Challenge website did not provide precision, recall, and accuracy. This reduced our already small number of positive samples to even fewer. Perhaps being able to train the model on all 2121 training samples would improve results.

In conclusion, predicting eviction from the FFCWS dataset is an outstanding and important problem. We explored a number of different combinations of features, classifiers, hyperparameters, and methods to predict eviction and were unable to achieve decent precision and recall scores on a local test dataset. We suggest more thought and effort goes in to missing data imputation and feature selection and that if possible, filling missing response variables.

Acknowledgments

We benefited from class notes, precept discussion, and office hours for Princeton's COS 424. The neural network implementation was done as coursework for Princeton's COS 402.

References

[1] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[2] Matthew Desmond. *Evicted: Poverty and profit in the American city*. Broadway Books, 2016.

[3] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *CoRR*, abs/1609.06570, 2016.

[4] Ian Lundberg. Eviction. `http://www.fragilefamilieschallenge.org/eviction/`. Accessed: 2017-03-27.

[5] Ian Lundberg. Precept notes in COS 424 Fundamentals of Machine Learning, March 2017.

[6] Ian Lundberg and Matthew J. Salganik. Fragile Families Challenge. `http://www.fragilefamilieschallenge.org/`. Accessed: 2017-03-27.

[7] Wes McKinney et al. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. van der Voort S, Millman J, 2010.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.