Applying Machine Learning to Automated Theorem Proving

Elizabeth Chen

Brown University Data Science Initiative

February 18, 2021

`https://github.com/elizabeth-c-chen/data1030-ML-theorem-proving`

Introduction

The dataset I studied for this project is the First-Order Theorem Proving dataset from the UCI Machine Learning Repository. The dataset was generated by feeding a set of about 6,000 conjectures from first-order logic to the "E" automated theorem prover (ATP), analyzing the features of the conjecture before and during the proof process, and recording the time taken to prove the conjecture using five different heuristics. In addition to the five heuristics from E, there is a sixth heuristic generated by the author of the research paper, Dr. James P. Bridge, which indicates that there is no best heuristic for that problem; this happens if none of the other five heuristics were able to prove the theorem in under 100 seconds.

Dr. Bridge conducted extensive research to determine whether machine learning could successfully relate the measurable features to determine correct classifications. He was able to show experimentally that selecting a heuristic based on features of the conjecture and associated axioms will do better than any single heuristic applied to all problems.

The raw dataset contains 14 "static" features (base features of the conjecture) and 39 "dynamic" features (features that arise during the early stages of the proof process). Descriptions of the features can be found in the Appendix in Tables 1 and 2. It also contains five additional columns representing the time taken by the five heuristics native to E. Initially, the time measurement columns were translated into a target variable labeled 0 through 5 to verify the column conversions from time to target labels as found in the author's training data.

My original plan was to solve the problem of determining which single heuristic out of the original six would work the best for a given problem, based on the features it possesses. However, upon closer examination of the target variable and consultation of the research paper associated with the dataset, I decided to work on one of Dr. Bridge's suggestions for further research, an alternative class labeling that takes into account the case when multiple heuristics were equally fast. I decided to modify my research problem because I saw that in many cases, with the level of precision the data was reported with, multiple heuristics could have proven the theorem very quickly and in the same amount of time, but only the lowest-indexed one would be chosen as the "best" in this case. I was curious to see how the results might compare to the original problem I had chosen, so I also ran the same algorithms on the original target and will compare the results later in this report. At the time of writing this report, I did not see any other public uses of this same dataset.

Using machine learning to determine if there is an underlying relationship between structure of a conjecture and the optimal heuristic to select is particularly useful as these relationships are often not obvious, even to experts in the field. The implications of successfully using machine learning to detect these relationships may give hints to developing new heuristics for ATPs or could help optimize the heuristic selection process and the total time taken to complete the proof.

Exploratory Data Analysis

Figure 1 shows the distribution of heuristics for the original multiclass data (green) and the modified multilabel to multiclass data (purple), which was created by allowing multiple heuristics to tie for the fastest time. A tolerance of 0.01 seconds was allowed—equal to the level of precision in the raw data time columns. We note the most significant difference in class 1, which loses about 57% of instances. The other classes 2, 3, 4, and 5 lose 50%, 30%, 41%, and 32% of their instances with the new labeling, respectively. This transformation demonstrates the effect of programmatically calculating the best heuristic via the minimum argument: the lowest indexed result will always be chosen first, and looking at the first label in many of the larger new classes, we see 1, 2, and 4 (the original classes with the top three biggest losses) make frequent appearances.

Figure 1

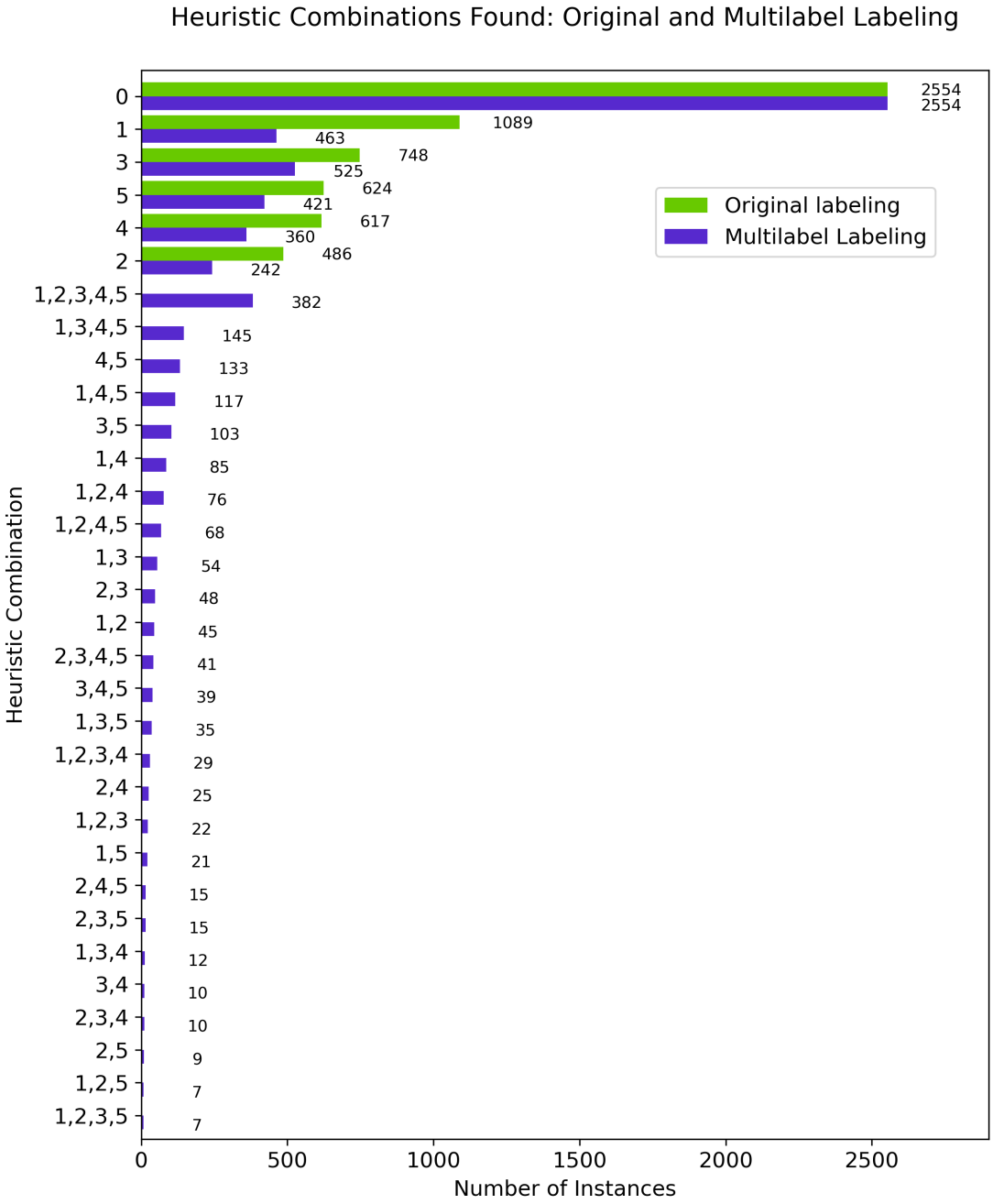


Figure 2 shows two plots of the fraction of clauses that are demodulators (S4) against the maximum clause weight (S13). The first plot shows the complete dataset, which includes some very extreme outliers that can be misleading. The second plot shows the data after filtering out approximately the largest 1% of values from S13; we are more easily able to see the distribution of the majority of data points by removing just 34 outlier points. I looked at every plot in this manner and while some were almost exactly the same, others did look very different. In cases where the plots from the full dataset appeared to have a possible pattern, there was nothing interesting to see in the plot once outliers were removed.

Figure 2

Heatmap of Static Feature 4 vs. Static Feature 13

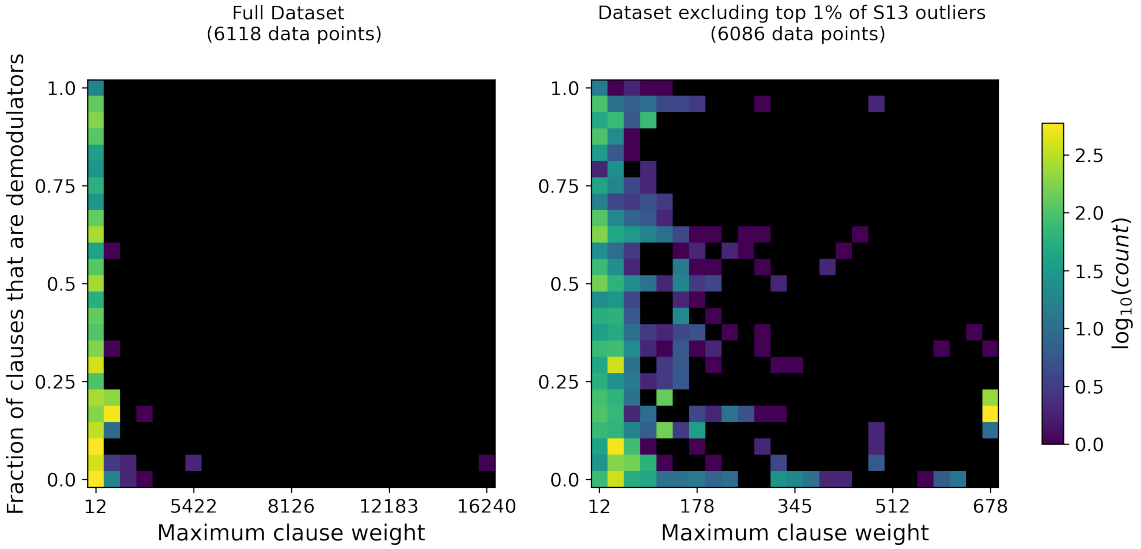
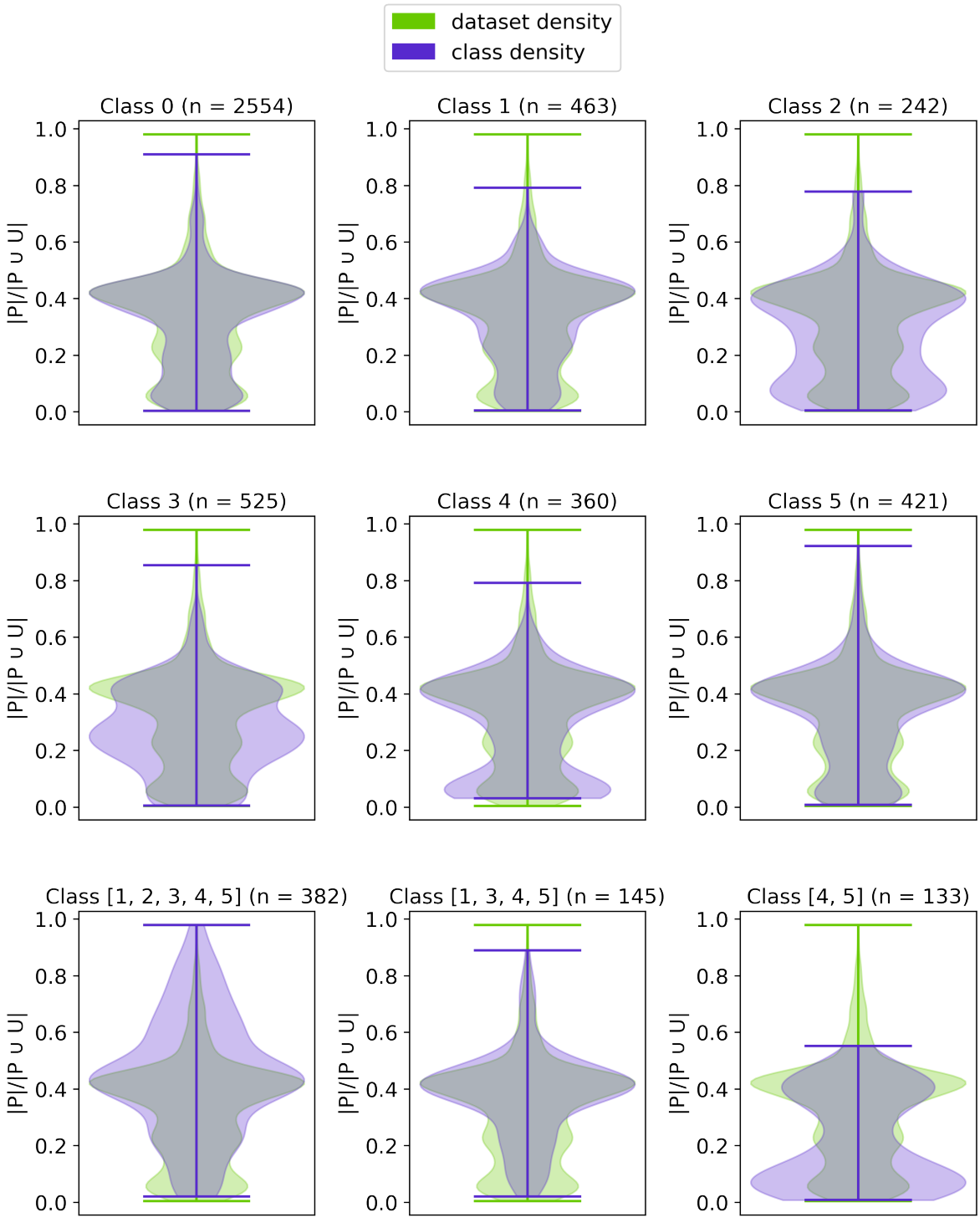


Figure 3 shows violin plots of the distribution of Dynamic Feature 3, $|P|/|P \cup U|$, for the largest nine classes of the modified dataset. The name of the feature represents the number of processed clauses divided by the number of processed and unprocessed clauses combined, which represents a snapshot taken at the time when the number of processed clauses is exactly 100. The minimum value of this feature is 0.00315, and the maximum value is 0.97917, where a small number indicates the existence of very many unprocessed clauses, and a large number indicates very few unprocessed clauses. The green density shows the overall density of the dataset, while the purple density shows the density of the specific class. It is interesting to note the similarities between most of the plots and the original density, such as the peak slightly above 0.4. However, each individual class shows variation from the rest and from the density and it will be interesting to find the importance value of this feature determined by the best model.

Figure 3


Distribution by Class of Feature D3: |P|/|P ∪ U|
dataset density
class density
Class 0 (n = 2554)
Class 1 (n = 463)
Class 2 (n = 242)
Class 3 (n = 525)
Class 4 (n = 360)
Class 5 (n = 421)
Class [1, 2, 3, 4, 5] (n = 382)
Class [1, 3, 4, 5] (n = 145)
Class [4, 5] (n = 133)
|P|/|P ∪ U|
0.0
0.2
0.4
0.6
0.8
1.0

Methods

In splitting the data, I took into consideration the fact that the data is IID with no group or time structure. With the goal of studying a multilabel classification problem, the training set would need to include as much variety in the data as possible, so as to include data points of varying features and target classes. This gave substantial freedom to split the data into the three groups. I performed a basic split of 80-20 to separate out the test set, and then used four-folds during cross validation to separate the training and validation data, for a total split of 60-20-20 for training, validation, and test sets, respectively. I decided to use Standard Scaler on all features as a majority of features followed skewed distributions and standardization of all features was needed in particular for Support Vector Classification.

I also removed two features from the raw dataset while constructing the data used for the train, test, and validation sets: Static Feature 5 and Dynamic Feature 21. Each of those features took on the same value for all data points, so they did not provide any real information to the model. This left a total of 51 features in the dataset used for preprocessing.

The machine learning algorithms I tried on my dataset were Random Forest, K Nearest Neighbors, Support Vector Machines, and Logistic Regression. For Random Forest, I tuned the maximum depth in the range of 15 to 40 and maximum number of features ranging between 15 and 51. For K Nearest Neighbors, I tuned the number of neighbors, ranging between 5 and 60, and tested out both the uniform and distance weighting methods. For Support Vector Classifier, I tuned the value of C between 10 and 10,000 in log increments, and tolerance between 0.0001 and 0.01. Finally, for Logistic Regression, I used the L2 penalty and tried the sag and lbfgs solvers, and ranged C between 1 and 10,000 in log increments.

For evaluating the machine learning algorithms, I decided to use the F1 score with micro averaging as the main scoring metric. I felt that the false positives and false negatives were both equally valuable in evaluating the model's success, since being able to not only identify classes but also distinguish the classes from each other is the ultimate goal. I chose to use micro averaging due to the fact that the modified class labeling had 32 classes with slightly over half the classes containing fewer than 50 data points and seven classes containing just 15 or fewer points. While I cared about the algorithm performing well on all classes, giving equal weight to both the largest classes and the smallest classes (i.e., macro averaging) did not seem valuable since it would be more difficult for the model to learn to identify a class based on just a few data points from the training data.

Uncertainties due to splitting were relatively small: for Random Forest it was 0.007, for K Nearest Neighbors it was 0.010, for Support Vector Classifier it was 0.014, and for Logistic Regression it was 0.016.

Results

The best machine learning algorithm turned out to be Random Forest, with a best test F1 score of 0.696 and mean test score of 0.682 ± 0.007. In comparison to the mean result achieved by a dummy classifier, the mean test score represents an improvement of 34 standard deviations above the best dummy model which gave a mean score of 0.346. K Nearest Neighbors also performed quite well, achieving a mean score of 0.666 ± 0.010. Support Vector Classifier performed slightly worse, achieving a mean score 0.626 ± 0.014. Logistic Regression performed the most poorly, achieving a mean score of 0.394 ± 0.016.

The top five most important features found were D4, D18, S13, S3, and S14. The first three mentioned impacted the score most dramatically compared to the rest, while S3 and S14 were more in line with the rest of the features. The feature D3 which was shown in the exploratory data analysis scored 19th most important out of the 51 features. The least important features were D17, D7, D22 and D36, which all barely changed or even minutely improved the test score when permuted.

Compared to the original dataset which used six classes, the modified dataset performed significantly better. Using the same four algorithms on the original data resulted in a best score given by Random Forest as well, but an overall lower mean score of 0.626 ± 0.010. The dummy classifier gave a mean score of 0.419 ± 0.016, so the random forest score represents an improvement of 11 standard deviations. Since there were six classes in this dataset compared to 32 classes in the modified dataset, this is not much of an achievement in comparison to the best model for the modified dataset.

Outlook

Originally my intention was to solve the multiclass problem using the original dataset, which contained six total classes. As the initial results were not so interesting or impressive, this led me to consider working on the alternative labeling with ties for time to see if any improvements could be made, and to get a sense for how the programmatic selection of the minimum time influenced the model. I am interested in furthering the work done in this project by testing alternative time tolerance levels other than 0.01 seconds in order to delve deeper into the mechanics of class labeling. This would require more research to better understand the E Prover's process and the machine used to generate the data. In addition to altering labels based on time variation, it would also be interesting to analyze relations and similarities between the different classes and use this data to inform the labeling process. Overall, there seem to be many more opportunities to continue improving on classification performance.

Appendix

Table 1: Descriptions of Static Features

Feature Number	Description
1	Fraction of clauses that are unit clauses
2	Fraction of clauses that are Horn clauses
3	Fraction of clauses that are ground Clauses
4	Fraction of clauses that are demodulators
5	Fraction of clauses that are rewrite rules (oriented demodulators)
6	Fraction of clauses that are purely positive
7	Fraction of clauses that are purely negative
8	Fraction of clauses that are mixed positive and negative
9	Maximum clause length
10	Average clause length
11	Maximum clause depth
12	Average clause depth
13	Maximum clause weight
14	Average clause weight

Table 2: Descriptions of Dynamic Features

Feature number	Description
1	Proportion of generated clauses kept (trivial clauses are discarded)
2	Sharing factor (A measure of the number of shared terms)
3	$\|P\|/\|P \cup U\|$
4	$\|U\|/\|A\|$
5	Ratio of longest clause lengths in P and A
6	Ratio of average clause lengths in P and A
7	Ratio of longest clause lengths in U and A
8	Ratio of average clause lengths in U and A
9	Ratio of maximum clause depths in P and A
10	Ratio of average clause depths in P and A
11	Ratio of maximum clause depths in U and A
12	Ratio of average clause depths in U and A
13	Ratio of maximum clause standard weights in P and A
14	Ratio of average clause standard weights in P and A
15	Ratio of maximum clause standard weights in U and A
16	Ratio of average clause standard weights in U and A
17	Ratio of the number of trivial clauses to |P|
18	Ratio of the number of forward subsumed clauses to |P|
19	Ratio of the number of non-trivial clauses to |P|
20	Ratio of the number of other redundant clauses to |P|
21	Ratio of the number of non-redundant deleted clauses to |P|
22	Ratio of the number of backward subsumed clauses to |P|
23	Ratio of the number of backward rewritten clauses to |P|
24	Ratio of the number of backward rewritten literal clauses to |P|
25	Ratio of the number of generated clauses to |P|
26	Ratio of the number of generated literal clauses to |P|
27	Ratio of the number of generated non-trivial clauses to |P|
28	`context_sr_count`/|P|
29	Ratio of paramodulations to |P|
30	`factor_count`/|P|
31	`resolv_count`/|P|
32	Fraction of unit clauses in U
33	Fraction of Horn clauses in U
34	Fraction of ground clauses in U
35	Fraction of demodulator clauses in U
36	Fraction of rewrite rule clauses in U
37	Fraction of clauses with only positive literals in U
38	Fraction of clauses with only negative literals in U
39	Fraction of clauses with positive and negative literals in U

The set of processed clauses is denoted by P and the set of unprocessed clauses by U. The set of axioms is denoted by A.
`context_sr_count`, `factor_count`, and `resolv_count` are variables used by E.

References

[1] J. P. Bridge. *Machine learning and automated theorem proving*. Tech. rep. University of Cambridge, 2010.

[2] J.P. Bridge, S.B. Holden, and L.C. Paulson. "Machine Learning for First-Order Theorem Proving". In: *Journal of Automated Reasoning* 53 (Feb. 2014), 141–172.