Lean Theorem Classification

Naveen Kannan Sai Konkimalla Kaden Nguyen

{naveenkc, saik, kadenn}@stanford.edu

Predicting

We created a model that classifies theorems in Lean into math categories (Algebra, Analysis, Geometry, Group Theory, etc). Our idea of creating a lean theorem classifier is related to previous work on machine learning automated theorem provers as well as research on natural language processing; there is extensive data and research for machine learning theorem provers, but none for organizing theorems into mathematical categories. Our work aims to further organize and automate research on automated theorem provers. Training on theorems in the math lean library, we created CNN and Multivariate Logistic regression models. Our baseline of training on a logistic regression had an accuracy of 26.16% accuracy; across the 28 given categories in the math lean library, our model had 93.85% accuracy.

Lean Dataset

To train our theorem classifier, we took advantage of the large set of verified proofs publicly available on mathlib, constructed from Lean repositories on Github [2]. From here, we extracted $80,000+$ complete and verified Lean proofs; we implemented a split between training data, validation data, and test data of 60 percent, 20 percent, and 20 percent respectively. An example of such a complete lean theorem and proof is shown below.

```
theorem additive_inverse_exists : ∀ (n : int), ∃ m : int,
                                  n + m = 0 :=
begin
  intro n,
  use (-n), -- posit -n for m
  simp, -- simp tactic to simplify our expression.
end
```

Category	Algebra	Analysis	Topology	Order	Measure Theory
# Theorems	10320	9794	8918	6996	5373
Category	Ring Theory	Linear Algebra	Category Theory	Group Theory	Set Theory
# Theorems	4411	4345	4049	3054	1879

Table 1. Ten most common proof categories in Mathlib Lean4

Processing

Pre-Processing

From Mathlib's raw files we are given theorems interleaved in definitions, proofs, comments, and instructions. Our initial pass through filtered out theorems to be saved as a csv. The files are conveniently stored within directories that correspond to math categories (algebra geometry, etc.) which were also saved as labels.

Post-Processing

After extracting theorems in Lean, our task is to make the text readable by a CNN model effectively. Within theorems, we chose to remove irrelevant characters like parentheses, punctuation and variable names. These added complexity to statements without information regarding the core theorem.

Features

Finally, we transform our filtered text into a vector of real numbers. We proceed in two ways.

For our baseline, we find the natural language GloVe embeddings of the raw Lean statements [3]. These vector embeddings are then passed into a Logistic Regression model.

For our CNN model, we tokenize the filtered text. Using the Keras inbuilt tokenizer, we map unique character sequences in the dataset to unique integers [1]. Since theorems have variable length, these integer lists have variable size as well. To mitigate this issue, we pad the shorter theorems with zeros.

Model Architecture

1-Dimensional Convolutional Neural Network

The first model we make use of is a 1-dimensional convolutional neural network (CNN), which works by weighting each of the values within a kernel "neighborhood" around a selected value in the input vector. In our use of the model, each of the values within this neighborhood is weighted equally.

$$y_j = \sum_{i=-k}^{k} x_{j-i} w_i \qquad w_i = \frac{1}{2k+1} \tag{1}$$

After the CNN layer, we make use of a max-pooling layer and a dense layer with softmax activation function to format the final output vector to represent probabilities.

Multinomial Logistic Regression

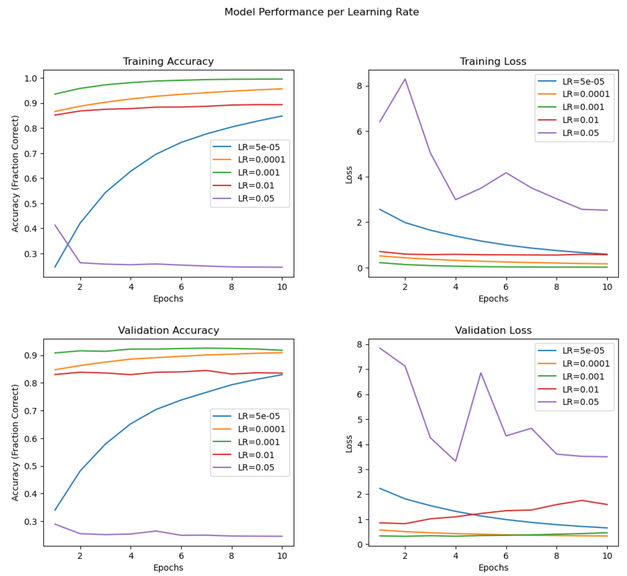
The standard logistic regression formulation deals with the binary classification problem, where there are two classes for the output data. We extend this to the multiclass classification by maintaining different parameters for each of the K classes. The final probability uses softmax to ensure that each parameter refers to a valid probability value for each of the classes.

$$P(y = k \mid x; \theta_k) = \frac{e^{\theta_k^\top x}}{\sum_{i=1}^{K} e^{\theta_i^\top x}} = \text{softmax}\left(k, \theta_1^\top x, \ldots, \theta_K^\top x\right) \tag{2}$$

Results

For a baseline result, this logistic regression classifies poorly with an accuracy of 26.16% across all theorems.

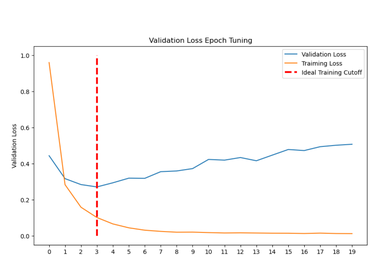
For our CNN algorithm, our pipeline is to first optimize for our hyperparameters of learning rate and training time on our validation set, and then to run the model on the test set for our final accuracy.

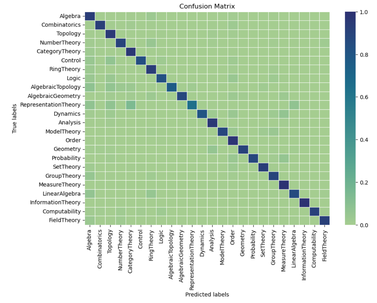


Results Cont.

In this experiment, the learning rate of 0.001 performed the best on the validation set and will be our saved parameter moving forward.

Now, we fine tune our training time by graphing the validation loss vs. training loss for this specific learning rate over an extended period:





With this experiment we can conclude that the ideal training time for optimizing our validation loss is 3 epochs. Using these parameters, on the test dataset we obtain an accuracy of 93.85%.

Discussion

To start, while our baseline model had better accuracy than a random guess between the different categories, we hypothesize that the logistic regression model did not perform well due to the fact that the word embedding model (GloVe) was trained on natural language and could not accurately categorize Lean statements which included a range of mathematical symbols and code.

Qualitatively, we can observe that in all of our training processes, validation loss never diverges excessively from training loss. As a result, we see that over-fitting wasn't an issue.

One example of misclassification in our validation test was with the algebra category, which overlapped the most with algebraic topology, representation theory, and linear algebra. In particular, many other categories in the dataset had a very small number of theorems to reference, like information theory and representation theory. This inherently skewed distribution of categories in the dataset may have also contributed to some of the inaccuracies.

Future Work

With surprisingly good accuracy and rapid training, the model can be integrated into future Lean dataset curation pipelines with little human oversight. With more computational resources, we can run even more expressive models (LSTM, Transformers, etc.) to better capture the logic behind the Lean statements. Eventually, this classifier could also be used in automated theorem prover models to add extra information about theorems.

References

[1] François Chollet et al.
Keras.
`https://keras.io`, 2015.

[2] The mathlib Community.
The lean mathematical library.
In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020, page 367–381, New York, NY, USA, 2020. Association for Computing Machinery.

[3] Jeffrey Pennington, Richard Socher, and Christopher D. Manning.
Glove: Global vectors for word representation.
In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.