Premise selection with machine learning

Piotr Piękos

1 czerwca 2019

1 Introduction

Task description is in a file Task.pdf

The goal was to create a premise selection machine learning algorithm from the mizar dataset based only on obfuscated binary features.

Several models are created with similar pipeline and next steps are suggested.

2 General Description

Approach is as follows:

1. Features are translated into numeric table (with memory-efficient types to make it possible)
2. Dataset of lists of theorems and premises is split into pairs of theorems and premises, so we ignore "set dependencies" - we don't predict usefulness for sets, just for single premises.
3. features from premise and theorem are concatenated
4. model on that dataset is trained to predict usefulness of the feature.

3 Dataset creation

As mentioned above, lists of premises for theorems are translated into pairs (theorem, pair). Rows created with that manner are labeled with 1 for usefulness.

Additional N rows (N is a hyperparameter) are sampled per every theorem from premises that aren't in "useful" set.

3.1 Embeddings

Additionally dense features are created with autoencoders. They are not used in every model though.

Before training simple autoencoder with two layers (ReLU, sigmoid) on the theorem/feature matrix - the reasoning is that similar features give similar 'context' for premises.

One could also try to do embeddings based theorem-premise relationships

4 Models

Following models are tested:

- LightGBM - raw features
- LightGBM - embeddings
- LightGBM - raw features + embeddings
- RandomForest - raw features

5 Evaluation

5.1 Metrics

Two evaluation methods were applied for each model:

- AUC 75/25: Simple AUC on test set, after training on a train set (split 75/25)
- PROVED %: % of theorems proved with E-PROVER with this model as a premise selector

Model	AUC 75/25	PROVED
1	6	87837
2	7	78
3	545	778
4	545	18744
5	88	788

5.2 Model similarities

T_i - Set of theorems proved by model i.

Diagram of how well model a covered theorems proved by b.

$a_{i,j} = \frac{|T_i \cap T_j|}{|T_j|}$

6 Next Steps

- **More models tested** - For example simple Regression (maybe on embeddings with polynomial features) could model different thing and help increasing at least sum of all theorems proved.
- **Training only on short proofs** - One can think that if there are many premises are in the proof, then they aren't that relevant and probably many of them are useful only with a combination of an other premise. Therefore we can just use short proofs for training that probably use simple relationships for inferences - the type we're looking for.
- **Regression** - This is an extension of the previous one. instead of classification we can train a regressor that in a simplest case scenario just sums up all the occurences of the premise for a given theorem, but one could also weight the premises by the inverse of the number of premises used for a proof, so if proof uses 3 premises to proof theorem then their weights are equal to 1/3. And for a pair (theorem,premise) we sum all the weights and train regression on that.
- **Iterative training** - theorems that are proved can later be used as samples for new training sets, which gets bigger with every iteration.
- **Hyperparameters** - There are a lot of parameters in the training. From lgbm parameters to 'meta-parameters':
  - Ratio of False to True samples in training dataset - setting it higher would probably improve results, as it would better resolve real scenario.
  - number of trees
  - trees depth
  - regularization
- **Embeddings with theorem-premises list** - Instead of creating embeddings with features one could try creating them with list of theorems and their respective premises.
- **More stubborn prediction with dynamic number of premises** - For now the program doesn't even check whether solver gave up (timeout) or didn't manage to solve (not enough premises). It could check it and increase number (start with small numbers) of premises until it finds proof or times out. That's important because for some theorems it may be sufficient to choose 2 premises, but for some many premises are needed. It's impossible to model with current architecture.
- **Ignored Set Dependencies** - for now we look only at pairs (Theorem, Premise), but it might be the case that 2 (or more) premises are useful for some theorem only when they're together.

- **Better Features** - Using semantic dependencies between features or at least extracting more semantic features (as with autoencoder attempt).
- **Recommender System approach** - This setup is very similar to the recommender system approach. We are literally recommending a SAT solver set of premises. I'd like to try some RecSyss approaches which could for example help with the problem of ignored set dependencies (even basic like apriori algorithms, although that definitely be too simple).