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.

Whole processing is in *'notebooks/training.ipynb'* (some parts were done with LGBM CLI after saving datasets to LIBSVM format, because lgbm python interface takes much more RAM).

Results are analysed in *'notebooks/compare_results.ipynb'*

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

6 LGBM models and 1 RF models are tested. 5 LGBM models and RF model are on raw data, 1 LGBM model is on AutoEncoder Embeddings. One of the LGBM models is on embeddings from AutoEncoder

LGBM differ in hyperparameters, hyperparameters sets: (shallow, deep, naive)

NAIVE Hyperparameters

```
output_model=LGBM_params_naive.bst
objective = binary
metric = auc
is_unbalance = true
boosting = gbdt
max_depth = 80
learning_rate = 0.05
num_rounds = 300
data = train.libsvm
```

SHALLOW Hyperparameters

```
output_model=LGBM_params_shallow.bst
objective = binary
metric = auc
is_unbalance = true
boosting = gbdt
num_leaves = 30
feature_fraction = 0.75
bagging_fraction = 0.8
bagging_freq = 20
learning_rate = 0.035
lambda_l2 = 0.06
verbose = 1
nthreads = 6
max_bin = 255
num_rounds = 5000
data = train.libsvm
```

DEEP Hyperparameters are basically the same but without restriction on num leaves

Following models are tested: (If not stated otherwise there are 8 false clauses per every positive clause)

- lgbmnaive- lightgbm trained with NAIVE hyperparameters
- lgbmshallow - lightgbm trained with SHALLOW hyperparameters
- lgbmdeep - lightgbm trained with DEEP hyperparameters
- lgbmdeepshallowmean - average of predictions of lgbmdeep and lgbmshallow
- lgbmdeepshallowmean_16 - average of predictions of lgbmdeep (trained on dataset with 16 false clauses per positive clause) and lgbmshallow (trained on datasest with 16 false clauses per positive clause)
- lgbmembed - lightgbm trained only on embeddings from AutoEncoder
- rf - RandomForest
- total - not a model but a sum of all theorems proved by any of the models.
- random - chooses random list of premises, used for benchmark (just an estimate of a benchmark, for a better benchmark it should be checked few times on many permutations, here results are just for one random permutation.)

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 %
lgbmnaive	0.92015	53.4
lgbmshallow	0.93747	46.0
lgbmdeep	0.94295	52.0
lgbmdeepshallowmean	0.94435	53.4
lgbmdeepshallowmean_16	0.91452	52.6
lgbmembed	0.91920	45.2
rf	0.84701	39.8
total	–	62.6
random	–	17.4

5.2 Model similarities

	lgbmshallow	lgbmnaive	lgbmdeep	lgbmdeepshallowmean	lgbmdeepshallowmean_16	lgbmembed	rf	random
lgbmshallow	1.000000	0.952174	0.976923	0.958801	0.931559	0.938053	0.904523	0.988506
lgbmnaive	0.820225	1.000000	0.834615	0.816479	0.825095	0.858407	0.829146	0.919540
lgbmdeep	0.951311	0.943478	1.000000	0.958801	0.908745	0.929204	0.874372	0.977011
lgbmdeepshallowmean	0.958801	0.947826	0.984615	1.000000	0.931559	0.938053	0.899497	0.988506
lgbmdeepshallowmean_16	0.917603	0.943478	0.919231	0.917603	1.000000	0.929204	0.909548	0.988506
lgbmembed	0.794007	0.843478	0.807692	0.794007	0.798479	1.000000	0.839196	0.954023
rf	0.674157	0.717391	0.669231	0.670412	0.688213	0.738938	1.000000	0.896552
random	0.322097	0.347826	0.326923	0.322097	0.326996	0.367257	0.391960	1.000000

Rysunek 1: $a_{i,j} = \frac{|T_i \cap T_j|}{|T_j|}$, T_i - Set of theorems proved by model i

.

Figure 1 shows how many (%) of j theorems (proved by model j) are proved by model i. $a_{i,j} = 1$ means that model i is at every aspect better (not worse) then model j (proved all his theorems).There is no nontrivial occurences of that.

As you can see even random permutation of premises isn't totally covered by any of the models and none of them is actually even really close. This suggests a lot of space for improvements.

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).

7 Conclusion

Despite using only syntactic features which are obfuscated and many shortcuts taken along the way models seem to be quite useful in helping selecting the best premises. There is still a huge space for improvement which is shown in previous chapter. I estimate that with these improvements model could reach 85% - 90% of theorems coverage.