Machine Learning on Knowledge Bases

Marcus Pfeiffer

19.06.2018

Contents

1 Introduction

In this paper, we will get to know machine learning based algorithms for improving the performance of Automatic Theorem Provers (ATPs) and Interactive Theorem Provers (ITPs) (in particular of Isabelle/HOL and Mizar). One problem in Theorem Proving where machine learning concepts can fruitfully be applied to is the problem of finding the premises needed to prove a given goal in a large set of available facts. This problem is known as *Premise Selection* or *Fact Selection*. It can be seen as a ranking problem of the available facts in the knowledge base in terms of usefulness, where the highest ranked ones are selected [BGK+16].

The importance of good Premise Selection comes from the fact, that automatic theorem proving is usually based on saturation/resolution [Sch13] implementing the superposition calculus or Satisfiability Modulo Theories (SMT) and although there are search heuristics for the proofs done by the ATPs this is still a hard problem [UHV10]. Thus it is crucial for finding a proof, that the premise set, that the ATP deals with, is as small as possible while containing the premises that are sufficient to prove the goal.

One important task that the success of Premise Selectors implementing machine learning algorithms relies on is the good extraction of features from the given data that the algorithms then deal with. In our case the data will be large knowledge bases and the features will be the proof dependencies as well as the symbols and structure of the formulas themselves. This data preprocessing will be the content of the first part of the paper.

In the second part we will consider as examples of machine learning algorithms the Naive Bayes algorithm, a k-Nearest Neighbor algorithm and a kernel-based algorithm applied to the Premise Selection problem. The machine learning algorithms get the features that we extracted in the first part, a proof goal and visible facts as input and return a predetermined number of facts ordered by their estimated relevance. They can be handed to the ATP directly respectively after translation to the appropriate syntax. By using machine learning, we hope to achieve improvements in the performance of ATPs compared to previous Premise Selection algorithms.

The presented algorithms are used to automatically reason in the Mizar Mathematical Library (MML) [AKT+11] and the Isabelle/HOL library [BGK+16]. In Isabelle, the tool that does the Premise selection, translation to the ATPs syntax and reconstruction of found proofs is called Sledgehammer.

Finally, we compare the performance of MePo, a purely symbol based Fact Selector with the performance of MaSh [KBKU13], the machine learning Fact Selector option in Sledgehammer, and with the one of MeSh, a combination of MaSh and a symbol based Premise Selector similar to MePo.

The main sources of this paper are [BGK+16] and [AKT+11]

2 Machine Learning for Premise Selection

Let us first state a comprehensive definition what Premise Selection is.

Definition 1 (Premise selection Problem)**.** [AKT+11, 195] Given an ATP A, a large number of premises $\mathcal{P}$ (stored in the knowledge base) and a new conjecture c, predict those premises from $\mathcal{P}$, that are likely of use to A for constructing a proof for c.

A first example illustrates the problem and emphasizes the importance of careful feature extraction and the reason to use machine learning methods. We want to evaluate what the relations between the goal and its premises are to find out what the indicators for useful facts given the proof goal are.

Example 1. The following proof goal is taken from the verification of cryptographic protocols by Paulson [Pau98].

$$\text{used}[] = \text{used}\, evs$$

A straightforward proof uses the following lemmas:

$$used\_Nil : \text{used}[] = \bigcup_B \text{parts}(\text{initState}\, B) \tag{1}$$

$$initState\_into\_used : X \in \text{parts}(\text{initState}\, B) \implies X \in \text{used}\, evs \tag{2}$$

$$subsetI : (\bigwedge x. x \in A \implies x \in B) \implies A \subseteq B \tag{3}$$

$$UN\_iff : b \in \bigcup_{x \in A} Bx \longleftrightarrow \exists x \in A. b \in Bx \tag{4}$$

By looking at the premises that the proof relies on, we can imagine why the symbol-based heuristic from MePo underestimates their relevance. Guessing lemmas by only looking at the symbols as indicators for useful premises might be misleading, since parts, $\bigcup$ and initState do not appear as symbols in the goal. In fact, MePo ranks (1) 3742th and hence does not select it. It is not directly clear from this small context why, but MaSh and MeSh rank (1),(2),(3) and (4) within their first 35 respectively 77.

Nevertheless, in the algorithms that are presented later we will still take into account the intuition that facts that share similar features and have a similar structure might be useful to prove each other, although this is not the case in this example.

2.1 Data preprocessing

Having this example in mind, we try to find features in the knowledge base that improve the accuracy of the ranking by the Premise Selection algorithm. The features that we will consider are proof dependencies as well as syntactic features such as the structure of the theorems or symbols occurring in the facts.

Large knowledge bases today can be general ontologies such as SUMO [SUM] and CYC [CYC]. We will however have mathematics-specific knowledge bases such as the Mizar Mathematical library [MML] and the library of Isabelle/HOL in mind and will test the performance on them. In this part, we are not yet concerned with the actual proof goal.

2.1.1 Dependencies in the knowledge base

The first information included in the knowledge base that we consider is the information which lemmas and theorems a theorem needs as premises to be proven. The idea is to capture the structure of the proof for all facts in the knowledge base. The algorithms will work on the assumption that the proofs for theorems that have a similar structure and similar symbols are similar. In other words, facts that are needed to prove a theorem similar to the proof goal might be helpful to prove the proof goal as well. The following definition makes the concept of a dependency precise:

Definition 2 (Dependency [AKT+11] [AMU11])**.** A definition or theorem T depends on some definition, lemma or theorem T' (T' is a dependency of T), if T' is needed for T in terms of well-formedness, justification, or provability of T

A subtle assumption when using the proofs and dependencies that a human has written as information for the machine learning algorithms is, that these dependencies are useful for finding an ATP proof as well [AKT+11, 195].

Proof dependencies are stored and computed differently in different systems. We go into detail on extracting the proof dependencies at the example of the Mizar system.

When looking at a Mizar proof we can distinguish between the steps a human has explicitly written down and the steps that the Mizar proof assistant has implicitly added to ensure the correctness.

To find out which premises are needed for the proof, we have to look at the premises needed in each step, however this task is not trivial since we do not know which premises the Mizar prover has taken into account.

A first heuristic to guarantee a sufficient premise set would be the following: Take all the premises in the human written proof and then add the facts that could have been added by all kinds of possible mechanisms that the prover has build in to find the missing prove steps.

This first heuristic may include a lot of premises that are not actually needed for the proof because we do not need all the premises that the mechanisms could generate in general. Furthermore in Mizar all definitions and theorems are collected in articles and for a needed premise, the whole article is included as dependency.

A procedure suggested in [AKT+11] to remove unnecessary premises from the set of dependencies first orders the premises given by the first heuristic which we call in the following context items. Then for each item we check whether the verification works without it and if it does, we remove it and leave it in the set otherwise. This is still an overestimation, i.e. we might have redundant premises left (e.g. the algorithm depends on the initial enumeration), but clearly we do not make the dependency set bigger.

We express the information about the dependencies in a matrix to make use of them in the training part of the machine learning algorithm:

Definition 3 (Proof matrix)**.** Let Γ be the set of (first order) formulas that appear in the knowledge base. For $p, c \in \Gamma$ define $\mu : \Gamma \times \Gamma \to \{0, 1\}$ by

$$\mu(c, p) = \begin{cases} 1 & \text{if } p \text{ is used to prove } c, \\ 0 & \text{otherwise} \end{cases}$$

i.e. if p is used to prove c, then there is a 1 in column c and row p and we can additionally define the set of used premises for a particular formula

$$usedPremises(c) := \{p \,|\, \mu(c, p) = 1\}$$

The definition of dependency can be refined by the definition of a visibility, which captures the fact that we do not need to consider those facts as possible premises from the set of facts from the knowledge base that appear after the goal in the proof text. This distinction enables improvements in the algorithms.

Definition 4 (Visibility)**.** Visibility is a partial order on the facts and encodes the fact, that a goal cannot be proved using facts that appear later in the proof development.

In Isabelle it is computed by using the theory extension order on the theory type and then using this preorder to make it into a partial order. The set of dependencies is a subset of the visible facts, namely of those facts, that are used to prove the goal. [BGK+16]

2.1.2 Feature extraction

Besides the information that we get from looking at the proof dependencies, we will use the features that we get from analyzing the formulas themselves. These kind of features will be symbols and (sub)terms appearing in the formulas. It is important to note, that by using the formulas symbols and structure as features, we assume that theorems with similar symbols and structure are likely to be useful for each others proofs. As one might expect, the information gained from this analysis depends strongly on the choice of features. In addition to the symbols, we can use types or type classes to which terms in the formula belong to. Furthermore, the theory in which the formula occurs may also be a relevant feature. In [BGK+16] we find a more concrete receipt how we can use the formula's structure as a feature, namely by considering all patterns up to a given depth and replacing variables with their type. The following two examples illustrate this structure extraction first in a general function setting and second in the syntax of Isabelle.

Example 2 (General)**.** For given depth 2, $g\,(h\,x\,a)$ with x being a variable of type τ we get the following structural features:

x	a	$g\_\_$	$g(h\_\_)$
$h\_\_$	$h\,x\_$	h_a	$h\,x\,a$

which are simplified to

τ	a	g	$g(h)$
h	$h(\tau)$	$h(a)$	$h(\tau, a)$

The simplification is done because variable names may be different in different theories and names do not give us information about the fact.

Example 3 (Concrete)**.** Additionally we can use the features from the facts that the human has explicitly given in the proof so far which helps if there are not many features in the proof goal itself . In Isabelle this would be the facts after the keywords "using", "from", "hence",... [BGK+16]

For using these syntactic information in the algorithms later, we again store them in a matrix.

transpose(map(map f) xss) = map(map f)(transpose xss), a lemma for lists has the features:

map	map(list_list)	fun
map(fun)	map(map, list list)	list
map(map)	transpose	list_list
map(transpose)	transpose(map)	List
map(map, transpose)	transpose(list_list)	

Additionally we can use the features from the facts that the human has explicitly given in the proof text so far. This helps if there are not many features in the proof goal itself. In Isabelle this would be the facts after the keywords "using", "from", "hence",... [BGK+16]

For using these syntactic information in the algorithms later, we again store them in a matrix.

Definition 5 (Feature Matrix)**.** Let $T = \{t_1, ..., t_m\}$ be a fixed enumeration of the set of features. Let Γ being the set of all formulas, define $\Phi : \Gamma \times \{1, ..., m\} \to \{0, 1\}$ by

$$\Phi(c, i) = \begin{cases} 1 & \text{if } t_i \text{ appears in } c, \\ 0 & \text{otherwise.} \end{cases}$$

To consider information of the expressed features of a particular formula $c \in \Gamma$, we define the feature function that maps the formula to its vector $\phi : \Gamma \to \{0, 1\}^m$ given by:

$$\phi_i^c = 1 \iff \Phi(c, i) = 1,$$

and the set of expressed features of a formula c

$$e(c) := \{t_i \mid \Phi(c, i) = 1\}.$$

Furthermore, we define the extended features of c to be the expressed features together with the features of the facts that were proved using c

$$\bar{e}(c) = e(c) \cup \bigcup_{\text{d such that c} \in usedPremises(d)} e(d)$$

for being able to define a "finer" Naive Bayes algorithm.

As one additional information that the algorithms will make use of we assign weights to the features, which express the rarity [BGK+16]. The assumption behind this is, that facts that share rare features have a higher probability of being useful than facts that share rather common features. For example, a function that we defined just in a particular context in our self written theory and that appears in the theorem that we want to prove should be taken much higher into account than a lemma about some frequently appearing function, that also appears in the goal. A canonical weight is the inverse document frequency [BGK+16, 8]. For a feature t, we write $w(t)$ for the weight of the feature.

Regarding the question how these information are actually stored, [BGK+16] suggests to store them as a tuple of the form (c, par(c), e(c), $usedPremises(c)$) where par(c)

specifies the predecessors of c w.r.t the visibility relation and shows how to extend it to get all visible facts so we do not need to store all of them.

In general the choice is not canonical and the exact way of extracting the features differs from one Premise Selection algorithm to another [BGK+16] [AKT+11] [USPV08] [HV11].

2.2 Machine Learning Algorithms for Premise Selection

The Premise Selection problem can be treated as ranking problem by considering the task as providing a ranking of formulas by their predicted usefulness. For $p, c \in \Gamma$ we denote the classifier function by $\text{Relevance}_c(p) \in \mathbb{R}$ which is the usefulness of p for proving c. The Premise Selector selects p if $\text{Relevance}_c(p)$ is above a certain threshold. It is important to note that we define for each formula p an own classifier Relevance_p which gets as input all formulas which have p in their visibility set. To test how good a classifier Relevance_p is for the facts in the knowledge base, we compare the result of the classifier to the actual usefulness, given by the corresponding entry in the dependency matrix. The loss function is a function $l : \mathbb{R} \times \mathbb{R} \to \mathbb{R}^+$. Examples are the 0-1 loss, which is 1 if $x = y$ and 0 otherwise and the squared loss which is given by $l(x, y) = (x - y)^2$ Given the loss function l we define the expected loss $E : \mathbb{R} \to \mathbb{R}$ for a premise p by

$$E(\text{Relevance}_c(p)) = \sum_{c \in \Gamma} l(\text{Relevance}_c(p), \mu(c, p))$$

In all the following algorithms, the goal is to find $\text{Relevance}_c(p)$ that takes a high value whenever p is used to prove c and close to 0 whenever p is not used to prove x which is just another way of saying that we want to minimize the expected loss.

2.2.1 k-Nearest Neighbors

In the setting of the k-Nearest Neighbors (k-NN) algorithm, we select those premises, that are used to prove the facts, that are in some sense close to the goal (hence the name) and also near goals themselves. We usually capture the nearness by defining a metric which has a small value if two values are close and a large one if they are away. In our case it is easier to make it the other way around and define nearness where higher values correspond to closer formulas. This nearness depends, as one might expect, on the common features and their weight. We define, for a given parameter τ_1, a constant determined by experiments, the nearness of two formulas a and b as:

$$n(a, b) = \sum_{t \in e(a) \cap e(b)} w(t)^{\tau_1}$$

where $e(a)$ are the expressed features and $w(t)$ the weight of feature t as defined above.

Let N be the set of the k formulas that have highest nearness to a given goal c. We call the elements of N neighbors.

For a given parameter τ_2 that captures the difference of importance of the premises used to prove the near neighbors and the near neighbors themselves, we define the relevance of a visible formula p as follows:

$$\text{Relevance}_c(p) = \left(\tau_2 \sum_{q \in N \,|\, p \in usedPremises(q)} \frac{n(q,c)}{|usedPremises(q)|} \right) + \begin{cases} n(p,c) & \text{if } p \in N \\ 0 & \text{otherwise} \end{cases}$$

There are different things to note here: We take into account all of our information that we described in the previous chapter. We therefore do not only use k-NN, which only uses the features but extend it with the information about dependencies in the following way: Each neighbor of the goal positively affects all of the neighbors dependencies, expressed by the left summand, and, expressed by the right summand, positively affects the neighbor itself, corresponding to the fact that dependencies as well as the theorem itself are useful for proving a theorem [BGK+16]. This captures the intuition, that proofs of similar theorems are often similar.

2.2.2 Naive Bayes

In the Naive Bayes (NB) setting we are interested in the probability

$$P(\text{p is used in the proof of c})$$

To be able to compute that probability, we characterize c by its features $e(c)$ and approximate it by the conditional probability

$$P(\text{p is used to prove c'} \,|\, \text{c' has features e(c)})$$

The formulas c', that we now use are, in contrast to a new goal c, proved and thus we can use the information stored for them. We now restrict (for computational reasons) the features that are allowed to appear in c' and get

$$P(\text{p is used in a proof of c'} \,| \\ \text{features in e(c) appear in c' , the ones in } \bar{e}(\text{p})\text{-e(c) do not})$$

Naive Bayes is called naive because we assume the conditional independence of the features, i.e. the appearance of two features/symbols in a formula is not related, which of course is not true in general. We apply Bayes' formula and use the independence assumption to get the following product of probabilities:

$$P(\text{p is used in the proof of c'}) \\ \cdot \prod_{t \in e(c) \cap \bar{e}(p)} P(\text{c' has feature t} \,|\, \text{p is used in the proof of c'}) \\ \cdot \prod_{t \in e(c) - \bar{e}(p)} P(\text{p has feature t} \,|\, \text{p is not used in the proof of c'}) \\ \cdot \prod_{t \in \bar{e}(p) - e(c)} P(\text{c' does not have feature t} \,|\, \text{t is used in the proof of c'})$$

We can compute each of these four expressions from known dependencies. To do so efficiently, the following factors are stored.

$s(q,t)$ stores the number of times a fact q occurs as a dependency of a fact described by feature t.

$r(q)$ stores the number of times a fact q occurs as a dependency and K is the total number of known proofs.

$$P(\text{p is used in the proof of c'}) = \frac{r(p)}{K}$$

$$P(\text{c' has feature t} \,|\, \text{p is used in the proof of c'}) = \frac{s(p,t)}{r(p)}$$

$$P(\text{c' does not have feature t} \,|\, \text{p is used in the proof of c'}) = 1 - \frac{s(p,t)}{r(p)}$$

Finally, $P(\text{p has feature t} \,|\, \text{p is not used in the proof of c'})$ is the a priori probability of p being used in a proof. Since this is very unlikely, we estimate it by a low, fixed probability.

Now, taking the weights of features from the last step into account and by applying the logarithm, we get the following:

$$\begin{aligned}\text{Relevance}_c(p) = \sigma_1 \ln(r(p)) + &\sum_{t \in e(c') \cap \bar{e}(p)} w(f) \ln \frac{\sigma_2 s(p,t)}{r(p)} \\ &+ \sigma_3 \sum_{t \in \bar{e}(p) - e(c)} w(f) \ln\left(1 - \frac{s(p,t)}{r(t)}\right) + \sigma_4 \sum_{t \in e(c) - \bar{e}(p)} w(f)\end{aligned}$$

where σ_i are factors which were determined by experiments from the authors of [BGK+16].

An implementation of naive Bayes is SNoW.

2.2.3 Kernel-based Learning

In contrast to the concept of the Bayes approach, we do not estimate some probability when using kernels. Instead, we construct a function space, that depends on the kernel and then choose the classifier function from that space that minimizes the expected loss. In contrast to arbitrary functions kernels allows us to use linear optimization. From a different perspective, the kernel function measures similarity between the two inputs, in our case the formulas. It gives us the opportunity to use classifier functions that are not linear in the features, i.e. to capture non-linear dependencies, while still giving the opportunity to use linear optimization to compute the factors.

[KvLT+12]

Definition 6 (Kernel)**.** A function $k : \Gamma \times \Gamma \to \mathbb{R}$ is a kernel, if there is a function $\phi : \Gamma \to F$ which is sometimes called basis function from Γ to an inner product space F, e.g. $\mathbb{R}^n$, s.t.

$$k(x,y) = \langle \phi(x), \phi(y) \rangle.$$

An example of a nonlinear kernel (which will be used) is the Gaussian kernel with parameter σ, where $\langle \cdot, \cdot \rangle$ is the standard scalar product in $\mathbb{R}^n$.

$$k_{gauss}(x,y) := exp\left(-\frac{\langle \phi^x, \phi^x \rangle - 2\langle \phi^x, \phi^x \rangle + \langle \phi^y, \phi^y \rangle}{\sigma^2}\right)$$

We consider a space of functions which are candidates for our classifier function:

Definition 7 (Kernel Function Space)**.** Given a kernel k, we define

$$\mathcal{F}_k := \left\{ f \in \mathbb{R}^\Gamma \mid f(x) = \sum_{v \in \Gamma} \alpha_v k(x, v)\, \alpha_v \in \mathbb{R},\ ||f|| < \infty \right\}$$

where $||f|| = \sum_{u,v \in \Gamma} \alpha_u \alpha_v k(u, v)$.

From these functions we choose the one which has the lowest expected loss w.r.t the square loss for the proofs of which we already know the dependencies. We prevent overfitting by additionally penalize functions with high metric, which corresponds to complicated functions, that are too adjust to the training set and do not generalize to unseen goals. It is important to see that we get a different expected loss for different premises. For each formula p we compute its classifier w.r.t its specific data as follows.

$$\text{Relevance}_p = \underset{f \in \mathcal{F}_k}{arg\,min} E(f) + \lambda ||f||^2$$

This is a minimization problem and we do not go into detail. A solution to this particular minimization problem can be found in [AKT+11].

There is one important drawback in using the above algorithms as only indicator of usefulness: In MaSh, for example, the facts and proofs typed after the last Sledgehammer run might not even considered on the next Sledgehammer run due to an implementation choice. And furthermore they appear in very few proofs only but may be very important due to the fact that they were written in the same theory as the proof goal. For this reason, the MaSh selector is in practice combined with a proximity selector that ranks the facts by decreasing order in the proof text and contributes with factor 0.2 to the result [BGK+16, 10].

Another issue that occurs in practice is the problem of non-monotonic theory changes, i.e. renaming, deletion, reordering and modifying of formulas and theories. This obviously means changes in dependencies and naming. Due to implementation choices, this is not always corrected, but we will not go into detail about these issues. [BGK+16, 12]

2.3 Performance in Comparison

In this section we evaluate how the Premise Selection algorithms using machine learning perform in comparison to ones not using machine learning. For this, we shortly introduce suitable metrics to compare them in a way, in which we actually understand, what goes on on the selection level.

2.3.1 Metrics

The following metric expresses, how good the premises in a known proof can be rediscovered by the Fact Selectors.

Definition 8 (Full recall)**.** Let $\{p_1, ..., p_n\}$ be the highest ranked facts by a Premise Selector and c the goal.

The full recall is the smallest natural number k for which $usedPremises(c) \subset \{p_1, ..., p_k\}$ and set to $n+1$ if the highest ranked facts do not include the needed premises. It indicates how many facts the selection algorithm would have needed. Smaller numbers indicate better selection [BGK+16].

2.3.2 MaSh/MeSh vs MePo in Sledgehammer

MePo, another Premise Selector for Isabelle uses the following heuristic for Fact Selection. It is further refined, for example by weighing the features by their rarity as we do in the other algorithms as well. It iteratively applies the following steps until it has selected the desired number of facts: We start the iteration with initializing the set of known symbols to the set of symbols in the proof goal.

1. For each fact, compute as its score $k/(k+u)$ where k is the number of known symbols occurring in the fact and u the number of unknown symbols.

2. Select the facts with the highest score and add the features of the selected facts to the set of known symbols.

MaSh is the name for both the implementation of the k-NN and Naive Bayes algorithms in Sledgehammer [BGK+16]. MeSh is the algorithm that combines the result of MaSh with the ones from the selector SInE (Sumo Inference Engine) [HV11] which is similar to MePo, by taking both results into account where both are weighted 0.5. MeSh performs even better overall than MaSh. As stated before, MaSh is in practice combined with a proximity selector, that takes into account, that new lemmas and theorems, defined in the same theory as the goal might not have enough impact on the knowledge base but are likely to play a role in the proof of the goal.

In the table we consider formalizations of different matter with each having around 750-1500 goals with about 6 dependencies per goal each, from 31 to 42 average features per fact and total numbers of 13-65000 total facts. Figure 1 [BGK+16] shows the full recall for these formalization. The lowest average full recall is highlighted. We see that MaSh and MeSh have a lower full recall by factors between two and six.

		MaSh		MeSh	
Formalization	MePo	NB	kNN	NB	kNN
Auth	647	104	143	**96**	112
IsaFoR	1332	**513**	604	517	570
Jinja	839	244	306	**229**	256
List	1083	**234**	263	259	271
Nominal2	1045	**220**	276	229	264
Probability	1324	424	422	**393**	395

Figure 1: Average full recall

Figure 2 [BGK+16] shows the success rate of the ATPs. In particular from the range between 100 and 300 given facts we see that the machine learning based algorithms outperform MePo. Another observation is, that the success rate decreases when the ATPs are given more than 300 facts. This really confirms, that a good Premise Selection is important, since the provers do not find as many proofs, when given many facts.

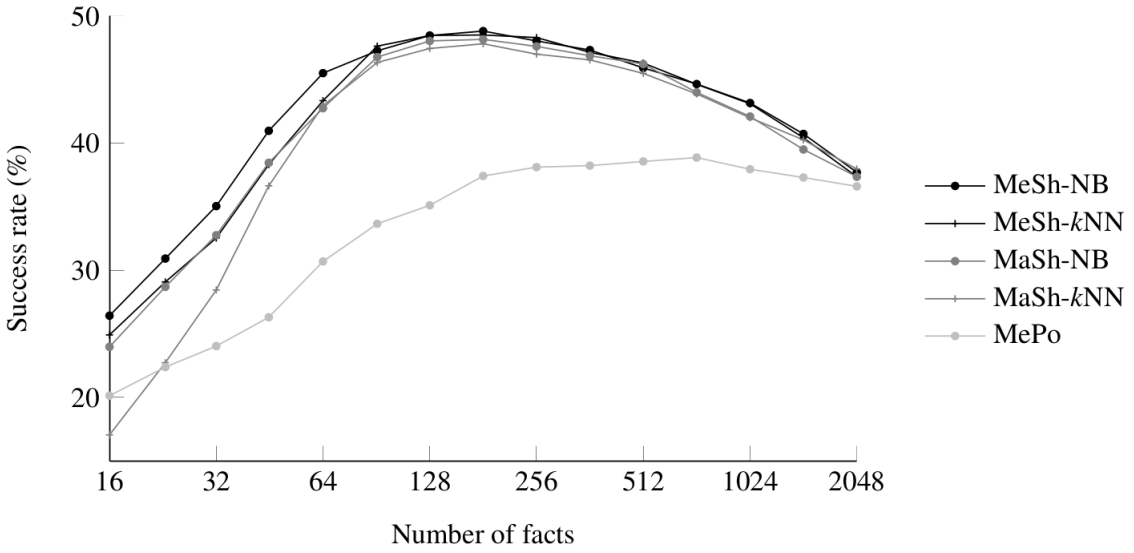

Figure 2: Success rates within 10s of a combination of four provers on the goals of the above formalizations

3 Summary and Outlook

We saw, that MaSh and MeSh clearly outperformed MePo in terms of the chosen metric and the considered formalizations. The metrics capture our guess about what should be useful and what we wanted the Premise Selectors to do. In a different comparison on different goals in [BGK+16], Sledgehammer with MaSh and MeSh still finds more proofs than with MePo, but the difference is smaller. MaSh is even outperformed by MePo using the E-prover and Vampire. A hypothesis is that MaSh tends to select facts, that "mislead" the ATPs heuristics to find the proof. Another reason could be that goals tend to rely on local facts for which we do not have enough data [BGK+16, 20].

As features, we used only symbols and terms occurring in formulas. [AKT+11] suggests to look for different ideas. In [ACI+16] instead of the above discussed learning methods, deep learning is used for Premise Selection.

Interesting related subjects include the applications of Machine Learning to Theory Exploration and Heuristic/Strategy selection for ATPs. Theory Exploration is the problem of automatically finding lemmas and propositions that can be rather straight-forward deduced, which in turn can be used to generate missing lemmas needed for a proof. This is suggested in [JRSC14]. Heuristic/Strategy Selection happens in the part, where the actual reasoning is carried out in ATPs and is mostly about picking the next clauses to use in the step of the saturation/resolution in a clever way to actually find the proof. [MP09] [BHP14] We saw, that a good feature extraction combined with machine learning can lead to significantly better results for Premise Selection in terms of the metrics. Even though this does not necessarily need to improve the performance of ATPs for particular cases, there is a lot of potential and progress has been made after the used papers came out.

References

[ACI+16] Alexander A. Alemi, François Chollet, Geoffrey Irving, Christian Szegedy, and Josef Urban. Deepmath - deep sequence models for premise selection. *CoRR*, abs/1606.04442, 2016.

[AKT+11] Jesse Alama, Daniel Kühlwein, Evgeni Tsivtsivadze, Josef Urban, and Tom Heskes. Premise selection for mathematics by corpus analysis and kernel methods. *CoRR*, abs/1108.3446, 2011.

[AMU11] Jesse Alama, Lionel Mamane, and Josef Urban. Dependencies in formal mathematics. *CoRR*, abs/1109.3687, 2011.

[BGK+16] Jasmin Christian Blanchette, David Greenaway, Cezary Kaliszyk, Daniel Kühlwein, and Josef Urban. A learning-based fact selector for isabelle/hol. *J. Autom. Reason.*, 57(3):219–244, October 2016.

[BHP14] James P. Bridge, Sean B. Holden, and Lawrence C. Paulson. Machine learning for first-order theorem proving. *Journal of Automated Reasoning*, 53(2):141–172, Aug 2014.

[CYC] Cyc: Logical reasoning with the world's largest knowledge base. `http://www.cyc.com/`. Accessed: 2018-05-13.

[HV11] Kryštof Hoder and Andrei Voronkov. Sine qua non for large theory reasoning. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *Automated Deduction – CADE-23*, pages 299–314, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[JRSC14] Moa Johansson, Dan Rosén, Nicholas Smallbone, and Koen Claessen. Hipster: Integrating theory exploration in a proof assistant. In Stephen M. Watt, James H. Davenport, Alan P. Sexton, Petr Sojka, and Josef Urban, editors, *Intelligent Computer Mathematics*, pages 108–122, Cham, 2014. Springer International Publishing.

[KBKU13] Daniel Kühlwein, Jasmin Christian Blanchette, Cezary Kaliszyk, and Josef Urban. Mash: Machine learning for sledgehammer. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Interactive Theorem Proving*, pages 35–50, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[KvLT+12] Daniel Kühlwein, Twan van Laarhoven, Evgeni Tsivtsivadze, Josef Urban, and Tom Heskes. Overview and evaluation of premise selection techniques for large theory mathematics. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Automated Reasoning*, pages 378–392, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[MML] Mizar mathematical library. `http://mizar.org/library/`. Accessed: 2018-05-13.

[MP09] Jia Meng and Lawrence C. Paulson. Lightweight relevance filtering for machine-generated resolution problems. *Journal of Applied Logic*, 7(1):41 – 57, 2009. Special Issue: Empirically Successful Computerized Reasoning.

[Pau98] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *J. Comput. Secur.*, 6(1-2):85–128, January 1998.

[Sch13] Stephan Schulz. System description: E 1.8. In Ken McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 735–743, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[SUM] Suggested upper merged ontology (sumo). `http://www.adampease.org/OP/`. Accessed: 2018-05-13.

[UHV10] Josef Urban, Krystof Hoder, and Andrei Voronkov. Evaluation of automated theorem proving on the mizar mathematical library. In Komei Fukuda, Joris van der Hoeven, Michael Joswig, and Nobuki Takayama, editors, *Mathematical Software – ICMS 2010*, pages 155–166, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[USPV08] Josef Urban, Geoff Sutcliffe, Petr Pudlák, and Jiří Vyskočil. Malarea sg1 - machine learner for automated reasoning with semantic guidance. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Automated Reasoning*, pages 441–456, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.