MaSh: Machine Learning for Sledgehammer

Daniel Kühlwein[1], Jasmin Christian Blanchette[2], Cezary Kaliszyk[3], and Josef Urban[1]

[1] ICIS, Radboud Universiteit Nijmegen, Netherlands
[2] Fakultät für Informatik, Technische Universität München, Germany
[3] Institut für Informatik, Universität Innsbruck, Austria

In memoriam Piotr Rudnicki 1951–2012

Abstract. Sledgehammer integrates automatic theorem provers in the proof assistant Isabelle/HOL. A key component, the relevance filter, heuristically ranks the thousands of facts available and selects a subset, based on syntactic similarity to the current goal. We introduce MaSh, an alternative that learns from successful proofs. New challenges arose from our "zero-click" vision: MaSh should integrate seamlessly with the users' workflow, so that they benefit from machine learning without having to install software, set up servers, or guide the learning. The underlying machinery draws on recent research in the context of Mizar and HOL Light, with a number of enhancements. MaSh outperforms the old relevance filter on large formalizations, and a particularly strong filter is obtained by combining the two filters.

1 Introduction

Sledgehammer [27] is a subsystem of the proof assistant Isabelle/HOL [25] that discharges interactive goals by harnessing external automatic theorem provers (ATPs). It heuristically selects a number of relevant facts (axioms, definitions, or lemmas) from the thousands available in background libraries and the user's formalization, translates the problem to the external provers' logics, and reconstructs any machine-found proof in Isabelle (Sect. 2). The tool is popular with both novices and experts.

Various aspects of Sledgehammer have been improved since its introduction, notably the addition of SMT solvers [7], the use of sound translation schemes [8], close cooperation with the first-order superposition prover SPASS [9], and of course advances in the underlying provers themselves. Together, these enhancements increased the success rate from 48% to 64% on the representative "Judgment Day" benchmark suite [9, 10].

One key component that has received little attention is the relevance filter. Meng and Paulson [23] designed a filter, MePo, that iteratively ranks and selects facts similar to the current goal, based on the symbols they contain. Despite its simplicity, and despite advances in prover technology [9, 14, 30], this filter greatly increases the success rate: Most provers cannot cope with tens of thousands of formulas, and translating so many formulas would also put a heavy burden on Sledgehammer. Moreover, the translation of Isabelle's higher-order constructs and types is optimized globally for a problem—smaller problems make more optimizations possible, which helps the automatic provers.

Coinciding with the development of Sledgehammer and MePo, a line of research has focused on applying machine learning to large-theory reasoning. Much of this work

has been done on the vast Mizar Mathematical Library (MML) [1], either in its original Mizar [22] formulation or in first-order form as the Mizar Problems for Theorem Proving (MPTP) [33]. The MaLARea system [34, 38] and the competitions CASC LTB and Mizar@Turing [31] have been important milestones. Recently, comparative studies involving MPTP [2, 20] and the Flyspeck project in HOL Light [17] have found that fact selectors based on machine learning outperform symbol-based approaches.

Several learning-based advisors have been implemented and have made an impact on the automated reasoning community. In this paper, we describe a tool that aims to bring the fruits of this research to the Isabelle community. This tool, MaSh, offers a memoryful alternative to MePo by learning from successful proofs, whether human-written or machine-generated.

Sledgehammer is a one-click technology—fact selection, translation, and reconstruction are fully automatic. For MaSh, we had four main design goals:

- *Zero-configuration*: The tool should require no installation or configuration steps, even for use with unofficial repository versions of Isabelle.
- *Zero-click*: Existing users of Sledgehammer should benefit from machine learning, both for standard theories and for their custom developments, without having to change their workflow.
- *Zero-maintenance*: The tool should not add to the maintenance burden of Isabelle. In particular, it should not require maintaining a server or a database.
- *Zero-overhead*: Machine learning should incur no overhead to those Isabelle users who do not employ Sledgehammer.

By pursuing these "four zeros," we hope to reach as many users as possible and keep them for as long as possible. These goals have produced many new challenges.

MaSh's heart is a Python program that implements a custom version of a weighted sparse naive Bayes algorithm that is faster than the naive Bayes algorithm implemented in the SNoW [12] system used in previous studies (Sect. 3). The program maintains a persistent state and supports incremental, nonmonotonic updates. Although distributed with Isabelle, it is fully independent and could be used by other proof assistants, automatic theorem provers, or applications with similar requirements.

This Python program is used within a Standard ML module that integrates machine learning with Isabelle (Sect. 4). When Sledgehammer is invoked, it exports new facts and their proofs to the machine learner and queries it to obtain relevant facts. The main technical difficulty is to perform the learning in a fast and robust way without interfering with other activities of the proof assistant. Power users can enhance the learning by letting external provers run for hours on libraries, searching for simpler proofs.

A particularly strong filter, MeSh, is obtained by combining MePo and MaSh. The three filters are compared on large formalizations covering the traditional application areas of Isabelle: cryptography, programming languages, and mathematics (Sect. 5). These empirical results are complemented by Judgment Day, a benchmark suite that has tracked Sledgehammer's development since 2010. Performance varies greatly depending on the application area and on how much has been learned, but even with little learning MeSh emerges as a strong leader.

2 Sledgehammer and MePo

Whenever Sledgehammer is invoked on a goal, the MePo (*Me*ng–*P*aulson) filter selects n facts $\phi_1, \ldots, \phi_n$ from the thousands available, ordering them by decreasing estimated relevance. The filter keeps track of a set of relevant symbols—i.e., (higher-order) constants and fixed variables—initially consisting of all the goal's symbols. It performs the following steps iteratively, until n facts have been selected:

1. Compute each fact's score, as roughly given by $r/(r+i)$, where r is the number of relevant symbols and i the number of irrelevant symbols occurring in the fact.
2. Select all facts with perfect scores as well as some of the remaining top-scoring facts, and add all their symbols to the set of relevant symbols.

The implementation refines this approach in several ways. Chained facts (inserted into the goal by means of the keywords `using`, `from`, `then`, `hence`, and `thus`) take absolute priority; local facts are preferred to global ones; first-order facts are preferred to higher-order ones; rare symbols are weighted more heavily than common ones; and so on.

MePo tends to perform best on goals that contain some rare symbols; if all the symbols are common, it discriminates poorly among the hundreds of facts that could be relevant. There is also the issue of starvation: The filter, with its iterative expansion of the set of relevant symbols, effectively performs a best-first search in a tree and may therefore ignore some relevant facts close to the tree's root.

The automatic provers are given prefixes $\phi_1, \ldots, \phi_m$ of the selected n facts. The order of the facts—the estimated relevance—is exploited by some provers to guide the search. Although Sledgehammer's default time limit is 30 s, the automatic provers are invoked repeatedly for shorter time periods, with different options and different number of facts $m \leq n$; for example, SPASS is given as few as 50 facts in some slices and as many as 1000 in others. Excluding some facts restricts the search space, helping the prover find deeper proofs within the allotted time, but it also makes fewer proofs possible.

The supported ATP systems include the first-order provers E [29], SPASS [9], and Vampire [28]; the SMT solvers CVC3 [4], Yices [13], and Z3 [24]; and the higher-order provers LEO-II [5] and Satallax [11].

Once a proof is found, Sledgehammer minimizes it by invoking the prover repeatedly with subsets of the facts it refers to. The proof is then reconstructed in Isabelle by a suitable proof text, typically a call to the built-in resolution prover Metis [16].

Example 1. Given the goal

$$\mathsf{map}\ f\ xs = ys \Longrightarrow \mathsf{zip}\ (\mathsf{rev}\ xs)\ (\mathsf{rev}\ ys) = \mathsf{rev}\ (\mathsf{zip}\ xs\ ys)$$

MePo selects 1000 facts: *rev_map*, *rev_rev_ident*, ..., *add_numeral_special*(3). The prover E, among others, quickly finds a minimal proof involving the 4th and 16th facts:

$$\begin{aligned} \textit{zip\_rev}&: \mathsf{length}\ xs = \mathsf{length}\ ys \Longrightarrow \mathsf{zip}\ (\mathsf{rev}\ xs)\ (\mathsf{rev}\ ys) = \mathsf{rev}\ (\mathsf{zip}\ xs\ ys) \\ \textit{length\_map}&: \mathsf{length}\ (\mathsf{map}\ f\ xs) = \mathsf{length}\ xs \end{aligned}$$

Example 2. MePo's tendency to starve out useful facts is illustrated by the following goal, taken from Paulson's verification of cryptographic protocols [26]:

$$\mathsf{used}\ [] \subseteq \mathsf{used}\ evs$$

A straightforward proof relies on these four lemmas:

$$\begin{aligned}
\textit{used_Nil}&: \ \mathsf{used}\ [] = \bigcup_B \mathsf{parts}\ (\mathsf{initState}\ B)\\
\textit{initState_into_used}&: \ X \in \mathsf{parts}\ (\mathsf{initState}\ B) \Longrightarrow X \in \mathsf{used}\ \textit{evs}\\
\textit{subsetI}&: \ (\bigwedge x.\ x \in A \Longrightarrow x \in B) \Longrightarrow A \subseteq B\\
\textit{UN_iff}&: \ b \in \bigcup_{x \in A} B\ x \longleftrightarrow \exists x \in A.\ b \in B\ x
\end{aligned}$$

The first two lemmas are ranked 6807th and 6808th, due to the many initially irrelevant constants ($\bigcup$, parts, initState, and $\in$). In contrast, all four lemmas appear among MaSh's first 45 facts and MeSh's first 77 facts.

3 The Machine Learning Engine

MaSh (*Ma*chine Learning for *S*ledge*h*ammer) is a Python program for fact selection with machine learning.[1] Its default learning algorithm is an approximation of naive Bayes adapted to fact selection. MaSh can perform fast model updates, overwrite data points, and predict the relevance of each fact. The program can also use the much slower naive Bayes algorithm implemented by SNoW [12].

3.1 Basic Concepts

MaSh manipulates theorem proving concepts such as facts and proofs in an agnostic way, as "abstract nonsense":

- A *fact* ϕ is a string.
- A *feature* f is also a string. A positive *weight* w is attached to each feature.
- *Visibility* is a partial order $\prec$ on facts. A fact ϕ is visible *from* a fact ϕ' if $\phi \prec \phi'$, and visible *through* the set of facts Φ if there exists a fact $\phi' \in \Phi$ such that $\phi \preceq \phi'$.
- The *parents* of a fact are its (immediate) predecessors with respect to $\prec$.
- A *proof* Π for ϕ is a set of facts visible from ϕ.

Facts are described abstractly by their feature sets. The features may for example be the symbols occurring in a fact's statement. Machine learning proceeds from the hypothesis that facts with similar features are likely to have similar proofs.

3.2 Input and Output

MaSh starts by loading the persistent model (if any), executes a list of commands, and saves the resulting model on disk. The commands and their arguments are

`learn` *fact parents features proof*
`relearn` *fact proof*
`query` *parents features hints*

The `learn` command teaches MaSh a new fact ϕ and its proof Π. The parents specify how to extend the visibility relation for ϕ, and the features describe ϕ. In addition to the supplied proof $\Pi \vdash \phi$, MaSh learns the trivial proof $\phi \vdash \phi$; hence something is learned even if $\Pi = \emptyset$ (which can indicate that no suitable proof is available).

[1] The source code is distributed with Isabelle2013 in the directory `src/HOL/Tools/Sledgehammer/MaSh/src`.

The `relearn` command forgets a fact's proof and learns a new one.

The `query` command ranks all facts visible through the given parents by their predicted relevance with respect to the specified features. The optional hints are facts that guide the search. MaSh temporarily updates the model with the hints as a proof for the current goal before executing the query.

The commands have various preconditions. For example, for `learn`, ϕ must be fresh, the parents must exist, and all facts in Π must be visible through the parents.

3.3 The Learning Algorithm

MaSh's default machine learning algorithm is a weighted version of sparse naive Bayes. It ranks each visible fact ϕ as follows. Consider a `query` command with the features $f_1, \ldots, f_n$ weighted $w_1, \ldots, w_n$, respectively. Let P denote the number of proofs in which ϕ occurs, and $p_j \leq P$ the number of such proofs associated with facts described by f_j (among other features). Let π and σ be predefined weights for known and unknown features, respectively. The estimated relevance is given by

$$r(\phi, f_1, \ldots, f_n) = \ln P + \sum_{j\colon p_j \neq 0} w_j\big(\ln(\pi p_j) - \ln P\big) + \sum_{j\colon p_j = 0} w_j \sigma$$

When a fact is learned, the values for P and p_j are initialized to a predefined weight τ. The models depend only on the values of P, p_j, π, σ, and τ, which are stored in dictionaries for fast access. Computing the relevance is faster than with standard naive Bayes because only the features that describe the current goal need to be considered, as opposed to all features (of which there may be tens of thousands). Experiments have found the values $\pi = 10$, $\sigma = -15$, and $\tau = 20$ suitable.

A crucial technical issue is to represent the visibility relation efficiently as part of the persistent state. Storing all the ancestors for each fact results in huge files that must be loaded and saved, and storing only the parents results in repeated traversals of long parentage chains to obtain all visible facts. MaSh solves this dilemma by complementing parentage with a cache that stores the ancestry of up to 100 recently looked-up facts. The cache not only speeds up the lookup for the cached facts but also helps shortcut the parentage chain traversal for their descendants.

4 Integration in Sledgehammer

Sledgehammer's MaSh-based relevance filter is implemented in Standard ML, like most of Isabelle.[2] It relies on MaSh to provide suggestions for relevant facts whenever the user invokes Sledgehammer on an interactive goal.

4.1 The Low-Level Learner Interface

Communication with MaSh is encapsulated by four ML functions. The first function resets the persistent state; the last three invoke MaSh with a list of commands:

[2] The code is located in Isabelle2013's files `src/HOL/Tools/Sledgehammer/sledgehammer_mash.ML`, `src/HOL/TPTP/mash_export.ML`, and `src/HOL/TPTP/mash_eval.ML`.

MaSh.unlearn ()
MaSh.learn [(*fact*$_1$, *parents*$_1$, *features*$_1$, *proof*$_1$), ..., (*fact*$_n$, *parents*$_n$, *features*$_n$, *proof*$_n$)]
MaSh.relearn [(*fact*$_1$, *proof*$_1$), ..., (*fact*$_n$, *proof*$_n$)]
suggestions = MaSh.query *parents features hints*

To track what has been learned and avoid violating MaSh's preconditions, Sledgehammer maintains its own persistent state, mirrored in memory. This mainly consists of the visibility graph, a directed acyclic graph whose vertices are the facts known to MaSh and whose edges connect the facts to their parents. (MaSh itself maintains a visibility graph based on `learn` commands.) The state is accessed via three ML functions that use a lock to guard against race conditions in a multithreaded environment [41] and keep the transient and persistent states synchronized.

4.2 Learning from and for Isabelle

Facts, features, proofs, and visibility were introduced in Sect. 3.1 as empty shells. The integration with Isabelle fills these concepts with content.

Facts. Communication with MaSh requires a string representation of Isabelle facts. Each theorem in Isabelle carries a stable "name hint" that is identical or very similar to its fully qualified user-visible name (e.g., *List.map.simps_2* vs. *List.map.simps*(2)). Top-level lemmas have unambiguous names. Local facts in a structured Isar proof [40] are disambiguated by appending the fact's statement to its name.

Features. Machine learning operates not on the formulas directly but on sets of features. The simplest scheme is to encode each symbol occurring in a formula as its own feature. The experience with MePo is that other factors help—for example, the formula's types and type classes or the theory it belongs to. The MML and Flyspeck evaluations revealed that it is also helpful to preserve parts of the formula's structure, such as subterms [3, 17].

Inspired by these precursors, we devised the following scheme. For each term in the formula, excluding the outer quantifiers, connectives, and equality, the features are derived from the nontrivial first-order patterns up to a given depth. Variables are replaced by the wildcard _ (underscore). Given a maximum depth of 2, the term g (h *x* a), where constants g, h, a originate from theories *T*, *U*, *V*, yields the patterns

T.g(_) T.g(U.h(_,_)) U.h(_,_) U.h(_,V.a) V.a

which are simplified and encoded into the features

T.g T.g(U.h) U.h U.h(V.a) V.a

Types, excluding those of propositions, Booleans, and functions, are encoded using an analogous scheme. Type variables constrained by type classes give rise to features corresponding to the specified type classes and their superclasses. Finally, various pieces of metainformation are encoded as features: the theory to which the fact belongs; the kind of rule (e.g., introduction, simplification); whether the fact is local; whether the formula contains any existential quantifiers or λ-abstractions.

Guided by experiments similar to those of Sect. 5, we attributed the following weights to the feature classes:

Fixed variable	20	Type	2	Presence of $\exists$	2
Constant	16	Theory	2	Presence of λ	2
Localness	8	Kind of rule	2	Type class	1

Example 3. The lemma $\mathsf{transpose}\,(\mathsf{map}\,(\mathsf{map}\,f)\,xs) = \mathsf{map}\,(\mathsf{map}\,f)\,(\mathsf{transpose}\,xs)$ from the *List* theory has the following features and weights (indicated by subscripts):

List_2	$\mathsf{List.transpose}_{16}$
$\mathsf{List.list}_2$	$\mathsf{List.transpose(List.map)}_{16}$
$\mathsf{List.map}_{16}$	$\mathsf{List.map(List.transpose)}_{16}$
$\mathsf{List.map(List.map)}_{16}$	$\mathsf{List.map(List.map, List.transpose)}_{16}$

Proofs. MaSh predicts which facts are useful for proving the goal at hand by studying successful proofs. There is an obvious source of successful proofs: All the facts in the loaded theories are accompanied by proof terms that store the dependencies [6]. However, not all available facts are equally suitable for learning. Many of them are derived automatically by definitional packages (e.g., for inductive predicates, datatypes, recursive functions) and proved using custom tactics, and there is not much to learn from those highly technical lemmas. The most interesting lemmas are those stated and proved by humans. Slightly abusing terminology, we call these "Isar proofs."

Even for user lemmas, the proof terms are overwhelmed by basic facts about the logic, which are tautologies in their translated form. Fortunately, these tautologies are easy to detect, since they contain only logical symbols (equality, connectives, and quantifiers). The proofs are also polluted by decision procedures; an extreme example is the Presburger arithmetic procedure, which routinely pulls in over 200 dependencies. Proofs involving over 20 facts are considered unsuitable and simply ignored.

Human-written Isar proofs are abundant, but they are not necessarily the best raw material to learn from. They tend to involve more, different facts than Sledgehammer proofs. Sometimes they rely on induction, which is difficult for automatic provers; but even excluding induction, there is evidence that the provers work better if the learned proofs were produced by similar provers [20, 21].

A special mode of Sledgehammer runs an automatic prover on all available facts to learn from ATP-generated proofs. Users can let it run for hours at at time on their favorite theories. The Isar proof facts are passed to the provers together with a few dozens of MePo-selected facts. Whenever a prover succeeds, MaSh discards the Isar proof and learns the new minimized proof (using MaSh.relearn). Facts with large Isar proofs are processed first since they stand to gain the most from shorter proofs.

Visibility. The loaded background theories and the user's formalization, including local lemmas, appear to Sledgehammer as a vast collection of facts. Each fact is tagged with its own abstract theory value, of type theory in ML, that captures the state of affairs when it was introduced. Sledgehammer constructs the visibility graph by using the (very fast) subsumption order $\trianglelefteq$ on theory.

A complication arises because $\trianglelefteq$ lifted to facts is a preorder, whereas the graph must encode a partial order $\preceq$. Antisymmetry is violated when facts are registered together. Despite the simultaneity, one fact's proof may depend on another's; for example, an inductive predicate's definition *p_def* is used to derive introduction and elimination rules *pI* and *pE*, and yet they may share the same theory. Hence, some care is needed when constructing $\preceq$ from $\trianglelefteq$ to ensure that $p\_def \preceq pI$ and $p\_def \preceq pE$.

When performing a query, Sledgehammer needs to compute the current goal's parents. This involves finding the maximal vertices of the visibility graph restricted to the facts available in the current Isabelle proof context. The computation is efficient for graphs with a quasi-linear structure, such as those that arise from Isabelle theories: Typically, only the first fact of a theory will have more than one parent. A similar computation is necessary when teaching MaSh new facts.

4.3 Relevance Filters: MaSh and MeSh

Sledgehammer's MaSh-based relevance filter computes the current goal's parents and features; then it queries the learner program (using MaSh.query), passing the chained facts as hints. This process usually takes about one second on modern hardware, which is reasonable for a tool that may run for half a minute. The result is a list with as many suggestions as desired, ordered by decreasing estimated relevance.

Relying purely on MaSh for relevance filtering raises an issue: MaSh may not have learned all the available facts. In particular, it will be oblivious to the very latest facts, introduced after Sledgehammer was invoked for the last time, and these are likely to be crucial for the proof. The solution is to enrich the raw MaSh data with a proximity filter, which sorts the available facts by decreasing proximity in the proof text.

Instead of a plain linear combination of ranks, the enriched MaSh filter transforms ranks into probabilities and takes their weighted average, with weight 0.8 for MaSh and 0.2 for proximity. The probabilities are rough approximations based on experiments. Fig. 1 shows the mathematical curves; for example, the first suggestion given by MaSh is considered about 15 times more likely to appear in a successful proof than the 50th.

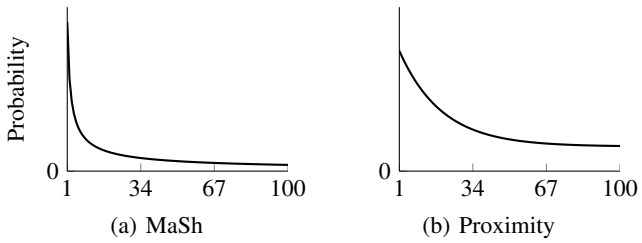

Fig. 1. Estimated probability of the jth fact's appearance in a proof

This notion of combining filters to define new filters is taken one step further by MeSh, a combination of *Me*Po and Ma*Sh*. Both filters are weighted 0.5, and both use the probability curve of Fig. 1(a).

Ideally, the curves and parameters that control the combination of filters would be learned mechanically rather than hard-coded. However, this would complicate and possibly slow down the infrastructure.

4.4 Automatic and Manual Control

All MaSh-related activities take place as a result of a Sledgehammer invocation. When Sledgehammer is launched, it checks whether any new facts, unknown to the visibility graph, are available. If so, it launches a thread to learn from their Isar proofs and update the graph. The first time, it may take about 10 s to learn all the facts in the background theories (assuming about 10 000 facts). Subsequent invocations are much faster.

If an automatic prover succeeds, the proof is immediately taught to MaSh (using MaSh.learn). The discharged (sub)goal may have been only one step in an unstructured proof, in which case it has no name. Sledgehammer invents a fresh, invisible name for it. Although this anonymous goal cannot be used to prove other goals, MaSh benefits from learning the connection between the formula's features and its proof.

For users who feel the need for more control, there is an `unlearn` command that resets MaSh's persistent state (using MaSh.unlearn); a `learn_isar` command that learns from the Isar proofs of all available facts; and a `learn_prover` command that invokes an automatic prover on all available facts, replacing the Isar proofs with successful ATP-generated proofs whenever possible.

4.5 Nonmonotonic Theory Changes

MaSh's model assumes that the set of facts and the visibility graph grow monotonically. One concern that arises when deploying machine learning—as opposed to evaluating its performance on static benchmarks—is that theories evolve nonmonotonically over time. It is left to the architecture around MaSh to recover from such changes. The following scenarios were considered:

- *A fact is deleted.* The fact is kept in MaSh's visibility graph but is silently ignored by Sledgehammer whenever it is suggested by MaSh.
- *A fact is renamed.* Sledgehammer perceives this as the deletion of a fact and the addition of another (identical) fact.
- *A theory is renamed.* Since theory names are encoded in fact names, renaming a theory amounts to renaming all its facts.
- *Two facts are reordered.* The visibility graph loses synchronization with reality. Sledgehammer may need to ignore a suggestion because it appears to be visible according to the graph.
- *A fact is introduced between two facts ϕ and ϕ'.* MaSh offers no facility to change the parent of ϕ', but this is not needed. By making the new fact a child of ϕ, it is considered during the computation of maximal vertices and hence visible.
- *The fact's formula is modified.* This occurs when users change the statement of a lemma, but also when they rename or relocate a symbol. MaSh is not informed of such changes and may lose some of its predictive power.

More elaborate schemes for tracking dependencies are possible. However, the benefits are unclear: Presumably, the learning performed on older theories is valuable and should be preserved, despite its inconsistencies. This is analogous to teams of humans developing a large formalization: Teammates should not forget everything they know each time a colleague changes the capitalization of some basic theory name. And should users notice a performance degradation after a major refactoring, they can always invoke `unlearn` to restart from scratch.

5 Evaluations

This section attempts to answer the main questions that existing Sledgehammer users are likely to have: How do MaSh and MeSh compare with MePo? Is machine learning really helping? The answer takes the form of two separate evaluations.[3]

5.1 Evaluation on Large Formalizations

The first evaluation measures the filters' ability to re-prove the lemmas from three formalizations included in the Isabelle distribution and the *Archive of Formal Proofs* [19]:

Auth	Cryptographic protocols [26]	743 lemmas
Jinja	Java-like language [18]	733 lemmas
Probability	Measure and probability theory [15]	1311 lemmas

These formalizations are large enough to exercise learning and provide meaningful numbers, while not being so massive as to make experiments impractical. They are also representative of large classes of mathematical and computer science applications.

The evaluation is twofold. The first part computes how accurately the filters can predict the known Isar or ATP proofs on which MaSh's learning is based. The second part connects the filters to automatic provers and measures actual success rates.

The first part may seem artificial: After all, real users are interested in any proof that discharges the goal at hand, not a specific known proof. The predictive approach's greatest virtue is that it does not require invoking external provers; evaluating the impact of parameters is a matter of seconds instead of hours. MePo itself has been fine-tuned using similar techniques. For MaSh, the approach also helps ascertain whether it is learning the learning materials well, without noise from the provers. Two (slightly generalized) standard metrics, full recall and AUC, are useful in this context.

For a given goal, a fact filter (MePo, MaSh, or MeSh) ranks the available facts and selects the n best ranked facts $\Phi = \{\phi_1, \ldots, \phi_n\}$, with $\mathit{rank}(\phi_j) = j$ and $\mathit{rank}(\phi) = n+1$ for $\phi \notin \Phi$. The parameter n is fixed at 1024 in the experiments below.

The known proof Π serves as a reference point against which the selected facts and their ranks are judged. Ideally, the selected facts should include as many facts from the proof as possible, with as low ranks as possible.

Definition 1 (Full Recall). The *full recall* is the minimum number $m \in \{0, \ldots, n\}$ such that $\{\phi_1, \ldots, \phi_m\} \supseteq \Pi$, or $n+1$ if no such number exists.

[3] Our empirical data are available at `http://www21.in.tum.de/~blanchet/mash_data.tgz`.

		MePo		MaSh		MeSh	
		Full rec.	AUC	Full rec.	AUC	Full rec.	AUC
Isar proofs	*Auth*	430	79.2	190	93.1	**142**	**94.9**
	Jinja	472	73.1	307	90.3	**250**	**92.2**
	Probability	742	57.7	384	88.0	**336**	**89.2**
ATP proofs	*Auth*	119	93.5	198	92.0	**68**	**97.0**
	Jinja	163	90.4	241	90.6	**84**	**96.8**
	Probability	428	74.4	368	85.2	**221**	**91.6**

Fig. 2. Average full recall and average AUC (%) with Isar and ATP proofs

Definition 2 (AUC). The *area under the receiver operating characteristic curve* (*AUC*) is given by

$$\frac{\left|\{(\phi,\phi') \in \Pi \times (\Phi - \Pi) \mid \mathit{rank}(\phi) < \mathit{rank}(\phi')\}\right|}{|\Pi| \cdot |\Phi - \Pi|}$$

Full recall tells how many facts must be selected to ensure that all necessary facts are included—ideally as few as possible. The AUC focuses on the ranks: It gives the probability that, given a randomly drawn "good" fact (a fact from the proof) and a randomly drawn "bad" fact (a selected fact that does not appear in the proof), the good fact is ranked before the bad fact. AUC values closer to 1 (100%) are preferable.

For each of the three formalizations (*Auth*, *Jinja*, and *Probability*), the evaluation harness processes the lemmas according to a linearization (topological sorting) of the partial order induced by the theory graph and their location in the theory texts. Each lemma is seen as a goal for which facts must be selected. Previously proved lemmas, and the learning performed on their proofs, may be exploited—this includes lemmas from imported background theories. This setup is similar to the one used by Kaliszyk and Urban [17] for evaluating their Sledgehammer-like tool for HOL Light. It simulates a user who systematically develops a formalization from beginning to end, trying out Sledgehammer on each lemma before engaging in a manual proof.[4]

Fig. 2 shows the average full recall and AUC over all lemmas from the three formalizations. For each formalization, the statistics are available for both Isar and ATP proofs. In the latter case, Vampire was used as the ATP, and goals for which it failed to find a proof are simply ignored. Learning from ATP proofs improves the machine learning metrics, partly because they usually refer to fewer facts than Isar proofs.

There is a reversal of fortune between Isar and ATP proofs: MaSh dominates MePo for the former but performs slightly worse than MePo for the latter on two of the formalizations. The explanation is that the ATP proofs were found with MePo's help. Nonetheless, the combination filter MeSh scores better than MePo on all the benchmarks.

Next comes the "in vivo" part of the evaluation, with actual provers replacing machine learning metrics. For each goal from the formalizations, 13 problems were gener-

[4] Earlier evaluations of Sledgehammer always operated on individual (sub)goals, guided by the notion that lemmas can be too difficult to be proved outright by automatic provers. However, lemmas appear to provide the right level of challenge for modern automation, and they tend to exhibit less redundancy than a sequence of similar subgoals.

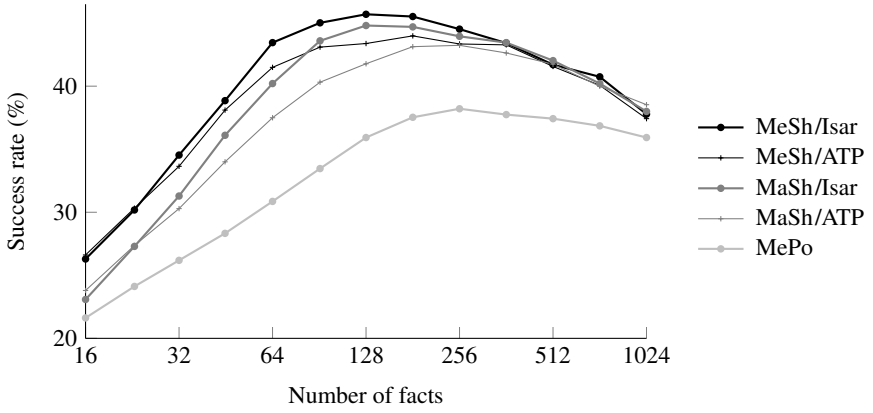

Fig. 3. Success rates for a combination of provers on *Auth* + *Jinja* + *Probability*

ated, with 16, 23 ($\approx 2^{4.5}$), 32, ..., 724 ($\approx 2^{9.5}$), and 1024 facts. Sledgehammer's translation is parameterized by many options, whose defaults vary from prover to prover and, because of time slicing, even from one prover invocation to another. As a reasonable uniform configuration for the experiments, types are encoded via the so-called polymorphic "featherweight" guard-based encoding (the most efficient complete scheme [8]), and λ-abstractions via λ-lifting (as opposed to the more explosive SK combinators).

Fig. 3 gives the success rates of a combination of three state-of-the-art automatic provers (Epar 1.6 [36], Vampire 2.6, and Z3 3.2) on these problems. Two versions of MaSh and MeSh are compared, with learning on Isar and ATP proofs. A problem is considered solved if it is solved within 10 s by any of them, using only one thread. The experiments were conducted on a 64-bit Linux server equipped with 12-core AMD Opteron 6174 processors running at 2.2 GHz. We observe the following:

- MaSh clearly outperforms MePo, especially in the range from 32 to 256 facts. For 91-fact problems, the gap between MaSh/Isar and MePo is 10 percentage points. (The curves have somewhat different shapes for the individual formalizations, but the general picture is the same.)
- MaSh's peak is both higher than MePo's (44.8% vs. 38.2%) and occurs for smaller problems (128 vs. 256 facts), reflecting the intuition that selecting fewer facts more carefully should increase the success rate.
- MeSh adds a few percentage points to MaSh. The effect is especially marked for the problems with fewer facts.
- Against expectations, learning from ATP proofs has a negative impact. A closer inspection of the raw data revealed that Vampire performs better with ATP (i.e., Vampire) proofs, whereas the other two provers prefer Isar proofs.

Another measure of MaSh and MeSh's power is the total number of goals solved for any number of facts. With MePo alone, 46.3% of the goals are solved; adding MaSh and MeSh increases this figure to 62.7%. Remarkably, for *Probability*—the most difficult formalization by any standard—the corresponding figures are 27.1% vs. 47.2%.

	MePo	MaSh	MeSh
E	55.0	49.8	**56.6**
SPASS	57.2	49.1	**57.7**
Vampire	55.3	49.7	**56.0**
Z3	53.0	51.8	**60.8**
Together	65.6	63.0	**69.8**

Fig. 4. Success rates (%) on Judgment Day goals

5.2 Judgment Day

The Judgment Day benchmarks [10] consist of 1268 interactive goals arising in seven Isabelle theories, covering among them areas as diverse as the fundamental theorem of algebra, the completeness of a Hoare logic, and Jinja's type soundness. The evaluation harness invokes Sledgehammer on each goal. The same hardware is used as in the original Judgment Day study [10]: 32-bit Linux servers with Intel Xeon processors running at 3.06 GHz. The time limit is 60 s for proof search, potentially followed by minimization and reconstruction in Isabelle. MaSh is trained on 9852 Isar proofs from the background libraries imported by the seven theories under evaluation.

The comparison comprises E 1.6, SPASS 3.8ds, Vampire 2.6, and Z3 3.2, which Sledgehammer employs by default. Each prover is invoked with its own options and problems, including prover-specific features (e.g., arithmetic for Z3; sorts for SPASS, Vampire, and Z3). Time slicing is enabled. For MeSh, some of the slices use MePo or MaSh directly to promote complementarity.

The results are summarized in Fig. 4. Again, MeSh performs very well: The overall 4.2 percentage point gain, from 65.6% to 69.8%, is highly significant. As noted in a similar study, "When analyzing enhancements to automatic provers, it is important to remember what difference a modest-looking gain of a few percentage points can make to users" [9, § 7]. Incidentally, the 65.6% score for MePo reveals progress in the underlying provers compared with the 63.6% figure from one year ago.

The other main observation is that MaSh underperforms, especially in the light of the evaluation of Sect. 5.1. There are many plausible explanations. First, Judgment Day consists of smaller theories relying on basic background theories, giving few opportunities for learning. Consider the theory *NS_Shared* (Needham–Shroeder shared-key protocol), which is part of both evaluations. In the first evaluation, the linear progress through all *Auth* theories means that the learning performed on other, independent protocols (certified email, four versions of Kerberos, and Needham–Shroeder public key) can be exploited. Second, the Sledgehammer setup has been tuned for Judgment Day and MePo over the years (in the hope that improvements on this representative benchmark suite would translate in improvements on users' theories), and conversely MePo's parameters are tuned for Judgment Day.

In future work, we want to investigate MaSh's lethargy on these benchmarks (and MeSh's remarkable performance given the circumstances). The evaluation of Sect. 5.1 suggests that there are more percentage points to be gained.

6 Related Work and Contributions

The main related work is already mentioned in the introduction. Bridges such as Sledgehammer for Isabelle/HOL, Miz$\mathbb{AR}$ [37] for Mizar, and HOL(y)Hammer [17] for HOL Light are opening large formal theories to methods that combine ATPs and artificial intelligence (AI) [20, 35] to help automate interactive proofs. Today such large theories are the main resource for combining semantic and statistical AI methods [39].[5]

The main contribution of this work has been to take the emerging machine-learning methods for fact selection and make them incremental, fast, and robust enough so that they run unnoticed on a single-user machine and respond well to common user-interaction scenarios. The advising services for Mizar and HOL Light [17, 32, 33, 37] (with the partial exception of MoMM [32]) run only as remote servers trained on the main central library, and their solution to changes in the library is to ignore them or relearn everything from scratch. Other novelties of this work include the use of more proof-related features in the learning (inspired by MePo), experiments combining MePo and MaSh, and the related learning of various parameters of the systems involved.

7 Conclusion

Relevance filtering is an important practical problem that arises with large-theory reasoning. Sledgehammer's MaSh filter brings the benefits of machine learning methods to Isabelle users: By decreasing the quantity and increasing the quality of facts passed to the automatic provers, it helps them find more, deeper proofs within the allotted time. The core machine learning functionality is implemented in a separate Python program that can be reused by other proof assistants.

Many areas are calling for more engineering and research; we mentioned a few already. Learning data could be shared on a server or supplied with the proof assistant. More advanced algorithms appear too slow for interactive use, but they could be optimized. Learning could be applied to control more aspects of Sledgehammer, such as the prover options or even MePo's parameters. Evaluations over the entire *Archive of Formal Proofs* might shed more light on MaSh's and MePo's strengths and weaknesses. Machine learning being a gift that keeps on giving, it would be fascinating to instrument a user's installation to monitor performance over several months.

Acknowledgment. Tobias Nipkow and Lawrence Paulson have, for years, encouraged us to investigate the integration of machine learning in Sledgehammer; their foresight has made this work possible. Gerwin Klein and Makarius Wenzel provided advice on technical and licensing issues. Tobias Nipkow, Lars Noschinski, Mark Summerfield, Dmitriy Traytel, and the anonymous reviewers suggested many improvements to earlier versions of this paper. The first author is supported by the Nederlandse organisatie voor Wetenschappelijk Onderzoek (NWO) project Learning2Reason. The second author is supported by the Deutsche Forschungsgemeinschaft (DFG) project Hardening the Hammer (grant Ni 491/14-1).

[5] It is hard to envisage all possible combinations, but with the recent progress in natural language processing, suitable ATP/AI methods could soon be applied to another major aspect of formalization: the translation from informal prose to formal specification.

References

1. The Mizar Mathematical Library. http://mizar.org/
2. Alama, J., Heskes, T., Kühlwein, D., Tsivtsivadze, E., Urban, J.: Premise selection for mathematics by corpus analysis and kernel methods. J. Autom. Reasoning, to appear, http://arxiv.org/abs/1108.3446
3. Alama, J., Kühlwein, D., Urban, J.: Automated and human proofs in general mathematics: An initial comparison. In: Bjørner, N., Voronkov, A. (eds.) LPAR-18. LNCS, vol. 7180, pp. 37–45. Springer (2012)
4. Barrett, C., Tinelli, C.: CVC3. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 298–302. Springer (2007)
5. Benzmüller, C., Paulson, L.C., Theiss, F., Fietzke, A.: LEO-II—A cooperative automatic theorem prover for higher-order logic. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNAI, vol. 5195, pp. 162–170. Springer (2008)
6. Berghofer, S., Nipkow, T.: Proof terms for simply typed higher order logic. In: Aagaard, M., Harrison, J. (eds.) TPHOLs 2000. LNCS, vol. 1869, pp. 38–52. Springer (2000)
7. Blanchette, J.C., Böhme, S., Paulson, L.C.: Extending Sledgehammer with SMT solvers. J. Autom. Reasoning, to appear, http://www21.in.tum.de/~blanchet/jar-smt.pdf
8. Blanchette, J.C., Böhme, S., Popescu, A., Smallbone, N.: Encoding monomorphic and polymorphic types. In: Piterman, N., Smolka, S. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 493–507. Springer (2013)
9. Blanchette, J.C., Popescu, A., Wand, D., Weidenbach, C.: More SPASS with Isabelle—Superposition with hard sorts and configurable simplification. In: Beringer, L., Felty, A. (eds.) ITP 2012. LNCS, vol. 7406, pp. 345–360. Springer (2012)
10. Böhme, S., Nipkow, T.: Sledgehammer: Judgement Day. In: Giesl, J., Hähnle, R. (eds.) IJCAR 2010. LNAI, vol. 6173, pp. 107–121. Springer (2010)
11. Brown, C.E.: Satallax: An automatic higher-order prover. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS, vol. 7364, pp. 111–117. Springer (2012)
12. Carlson, A.J., Cumby, C.M., Rosen, J.L., Roth, D.: SNoW user guide. Tech. rep., C.S. Dept., University of Illinois at Urbana-Champaign (1999), http://cogcomp.cs.illinois.edu/papers/CCRR99.pdf
13. Dutertre, B., de Moura, L.: The Yices SMT solver. http://yices.csl.sri.com/tool-paper.pdf (2006)
14. Hoder, K., Voronkov, A.: Sine qua non for large theory reasoning. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE-23. LNAI, vol. 6803, pp. 299–314. Springer (2011)
15. Hölzl, J., Heller, A.: Three chapters of measure theory in Isabelle/HOL. In: van Eekelen, M.C.J.D., Geuvers, H., Schmaltz, J., Wiedijk, F. (eds.) ITP 2011. LNCS, vol. 6898, pp. 135–151. Springer (2011)
16. Hurd, J.: First-order proof tactics in higher-order logic theorem provers. In: Archer, M., Di Vito, B., Muñoz, C. (eds.) Design and Application of Strategies/Tactics in Higher Order Logics. pp. 56–68. NASA Tech. Reports (2003)
17. Kaliszyk, C., Urban, J.: Learning-assisted automated reasoning with Flyspeck. CoRR abs/1211.7012 (2012), http://arxiv.org/abs/1211.7012
18. Klein, G., Nipkow, T.: Jinja is not Java. In: Klein, G., Nipkow, T., Paulson, L. (eds.) Archive of Formal Proofs. http://afp.sf.net/entries/Jinja.shtml (2005)
19. Klein, G., Nipkow, T., Paulson, L. (eds.): Archive of Formal Proofs. http://afp.sf.net/
20. Kühlwein, D., van Laarhoven, T., Tsivtsivadze, E., Urban, J., Heskes, T.: Overview and evaluation of premise selection techniques for large theory mathematics. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS, vol. 7364, pp. 378–392. Springer (2012)

21. Kühlwein, D., Urban, J.: Learning from multiple proofs: First experiments. In: Fontaine, P., Schmidt, R., Schulz, S. (eds.) PAAR-2012. pp. 82–94 (2012)
22. Matuszewski, R., Rudnicki, P.: Mizar: The first 30 years. Mechanized Mathematics and Its Applications 4(1), 3–24 (2005)
23. Meng, J., Paulson, L.C.: Lightweight relevance filtering for machine-generated resolution problems. J. Applied Logic 7(1), 41–57 (2009)
24. de Moura, L.M., Bjørner, N.: Z3: An efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer (2008)
25. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic, LNCS, vol. 2283. Springer (2002)
26. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. J. Comput. Secur. 6(1-2), 85–128 (1998)
27. Paulson, L.C., Blanchette, J.C.: Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In: Sutcliffe, G., Ternovska, E., Schulz, S. (eds.) IWIL-2010 (2010)
28. Riazanov, A., Voronkov, A.: The design and implementation of Vampire. AI Comm. 15(2-3), 91–110 (2002)
29. Schulz, S.: System description: E 0.81. In: Basin, D., Rusinowitch, M. (eds.) IJCAR 2004. LNAI, vol. 3097, pp. 223–228. Springer (2004)
30. Schulz, S.: First-order deduction for large knowledge bases. Presentation at Deduction at Scale 2011 Seminar, Ringberg Castle (2011), http://www.mpi-inf.mpg.de/departments/rg1/conferences/deduction10/slides/stephan-schulz.pdf
31. Sutcliffe, G.: The 6th IJCAR automated theorem proving system competition—CASC-J6. AI Communications, to appear
32. Urban, J.: MoMM—Fast interreduction and retrieval in large libraries of formalized mathematics. Int. J. AI Tools 15(1), 109–130 (2006)
33. Urban, J.: MPTP 0.2: Design, implementation, and initial experiments. J. Autom. Reasoning 37(1-2), 21–43 (2006)
34. Urban, J.: MaLARea: A metasystem for automated reasoning in large theories. In: Sutcliffe, G., Urban, J., Schulz, S. (eds.) ESARLT 2007. CEUR Workshop Proceedings, vol. 257. CEUR-WS.org (2007)
35. Urban, J.: An overview of methods for large-theory automated theorem proving. In: Höfner, P., McIver, A., Struth, G. (eds.) ATE-2011. CEUR Workshop Proceedings, vol. 760, pp. 3–8. CEUR-WS.org (2011)
36. Urban, J.: BliStr: The blind strategymaker. CoRR abs/1301.2683 (2013), http://arxiv.org/abs/1301.2683
37. Urban, J., Rudnicki, P., Sutcliffe, G.: ATP and presentation service for Mizar formalizations. J. Autom. Reasoning 50(2), 229–241 (2013)
38. Urban, J., Sutcliffe, G., Pudlák, P., Vyskočil, J.: MaLARea SG1—Machine learner for automated reasoning with semantic guidance. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNAI, vol. 5195, pp. 441–456. Springer (2008)
39. Urban, J., Vyskočil, J.: Theorem proving in large formal mathematics as an emerging AI field. In: Bonacina, M.P., Stickel, M.E. (eds.) Automated Reasoning and Mathematics—Essays in Memory of William McCune. LNAI, vol. 7788, pp. 240–257. Springer (2013)
40. Wenzel, M.: Isabelle/Isar—A generic framework for human-readable proof documents. In: Matuszewski, R., Zalewska, A. (eds.) From Insight to Proof—Festschrift in Honour of Andrzej Trybulec. Studies in Logic, Grammar, and Rhetoric, vol. 10(23). Uniwersytet w Białymstoku (2007)
41. Wenzel, M.: Parallel proof checking in Isabelle/Isar. In: Dos Reis, G., Théry, L. (eds.) PLMMS 2009. pp. 13–29. ACM Digital Library (2009)