École normale supérieure
Department of Computer Science

Suggesting relevant lemmas by learning from succesful proofs

INTERNSHIP REPORT

Jonathan Laurent
jonathan.laurent@ens.fr

Under the supervision of:

Hugo Herbelin
hugo.herbelin@inria.fr

August 2016

1. Introduction

In the last decade, proof assistants have become mature enough to enable the formalization of large mathematical theories [6] and the certification of industrial-scale software like the *Compcert C compiler* [12]. Despite these achievements, writing formal proofs remains a long and tedious process and the developement of better automation tools is likely to be a key ingredient towards a more widespread use of interactive theorem provers among computer scientists and mathematicians.

The Isabelle proof assistant features a tool named *Sledgehammer*, which when invoked attempts to discharge the current proof obligation in a completely automated way and which appeared to be powerful enough to prove 47% of the lemmas of a formal proof of the Kepler conjecture without any human intervention [7]. Such a tool relies on two key components:

1. An **automated theorem prover**, which is able to search efficiently for a formal derivation of some goal given a small set of premises.

2. A **premise selector**, which heuristically selects a bunch of facts that may be useful to the automated theorem prover from a base of available lemmas.

Although Coq provides basic automated provers such as the `auto` and `firstorder` tactics, which can be extended using the *Ltac* language, it does not come with an automated tool for searching for relevant lemmas. During this internship, we made the following contributions:

1. We developped a tool for Coq that ranks the available lemmas by respect to their relevance in proving a given fact. It uses machine learning techniques so as to capture useful connections between the symbols that appear in a statement and which lemmas are used in its proof, in a similar fashion to the premise selector of Sledgehammer [10].

2. We explored a possible improvement of the current state-of-the-art premise selection methods [1], which involves using topics extraction techniques based on LDA to generate richer features for formulas.

3. We tested our tool on two large corpora of formalized mathematics (namely the *CoRN* repository and the *mathcomp* library) and run a comparative evaluation of the different algorithms it implements that is easily reproducible.

1.1. Presentation of `colt`

During our internship, we developped a set of tools to export large corpora of formalized mathematics in a format that is accessible to machine learning algorithms, train statistical models of lemma relevance on them and use such models within the Coq proof assistant. These tools are gathered under the name of `colt`, which stands for *Coq Learning Tools*.

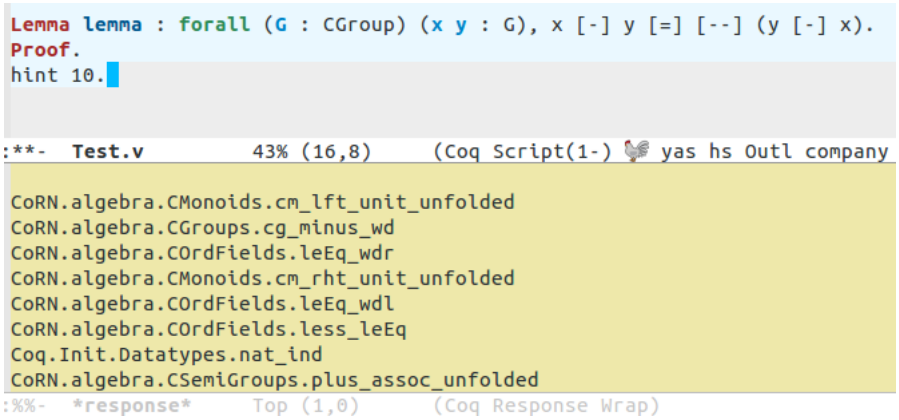


Figure 1: A screenshot of a user triggering the `hint` tactic of `colt`

The most interesting component of `colt` from the user point of view is its `hint` tactic, which when invoked with argument n at the beginning of a proof outputs a list of n suggestions of lemmas, ordered by relevance. An example is shown Figure 1, where we try to prove a simple result on commutative groups using the CoRN library, namely the identity $x - y = -(y - x)$. The first suggestion of `colt` is the lemma `cm_lft_unit_unfolded` from the `CMonoids` module:

```
Lemma cm_lft_unit_unfolded (M : CMonoid) (x : M) : [0] [+] x [=] x.
```

which seems like a coherent guess as a commutative group is also a commutative monoid. That being said, other suggestions like the third one are clearly irrelevant. In fact, the output of `hint` typically contains a lot of noise as `colt` has no notion of the mathematical meaning of the formulas it manipulates. Therefore, its output is better targeted at an automated prover which can easily handle dozens of suggestions than at a human user who wants to get an insight about some statement. We show in section 3 that `hint` covers an average of 85% of the lemmas that are

used in the proof of a random CoRN statement within its 100 first suggestions, which makes for a subset of 2% of all the available lemmas.

Implementation details The current version of `colt` consists in about 1000 lines of python relying on the *numpy* and *scikit-learn* libraries, a 350 lines Coq plugin written in OCaml which implements the `hint` tactic along with features extraction utilities, and also 250 lines bash script for processing large corpora of mathematics that uses both this plugin and the *coq-dpdgraph* program. The whole source package can be found at:

`http://www.eleves.ens.fr/home/jlaurent/software/colt-release.tar.gz`

1.2. About this report

This report was written to be accessible to people with very little knowledge in machine learning. Such readers may want to skip more technical sections labelled with the $(*)$ symbol though.

Acknowledgements I would like to thank Professor Hugo Herbelin for providing me a great scientific environment within the PPS team and Théo, Nicolas and Victor for being great officemates.

2. Machine learning techniques

In this section, we give an overview of the machine learning techniques that are used in `colt` for premise selection. After we give a precise definition of the premise selection problem, we describe how formulas and proof corpora are represented in a way that is accessible to machine learning algorithms. Then, we introduce two simple estimators based on k-nearest-neigbors and Naive Bayes classifiers that are commonly used in the litterature [1, 10, 11]. Finally, we explain how we improved them by using topics extraction techniques.

2.1. Problem definition

We are given a list $(l_1, \cdots, l_k)$ of k lemmas. For any formula f, we want to sort these lemmas by order of relevance in proving f. In order to learn such a sorting

function, we are given a training set $D = ((f_1, y_1), \cdots, (f_n, y_n))$ where f_i is a statement for all i and y_i is a boolean vector of size k called its dependency vector, whose i^{th} coordinate is 1 if and only if the lemma l_i appears in the proof of f_i.

Remarks

1. Rather than learning a sorting function directly, we usually learn a scoring function which maps any formula f to a score vector $s(f)$ of size k whose i^{th} coordinate is as high as l_i is relevant in proving f.

2. As large corpora of formalized mathematics are a scarce resource and the lemmas of a library are usually strongly dependent on each other, the database lemmas $\{l_1, \cdots, l_k\}$ along with their dependencies often constitute a major part of the training set.

2.2. Data and features

Most machine learning algorithms are not adapated to deal directly with structured objects like mathematical formulas or proof trees. Even manipulating variable-length data can be quite tricky and requires advanced techniques like recurrent neural networks. For this reason, we model formulas as finite sets of mathematical symbols. For instance, the fact that $(e^z)^n = e^{nz}$ for n an integer and z a complex number appears in the following lemma of the CoRN library:

```
Lemma ExpCC_pow : forall z n, ExpCC z[^]n [=] ExpCC (nring n[*]z).
```

which is in turn modelled in `colt` by the following set of symbols:

{ `CC`, `ExpCC`, `cr_mult`, `nat`, `nexp_op`, `nring`, `st_eq` }

Then, given an enumeration $\{s_1, \cdots, s_p\}$ of all the symbols occuring in the lemma database or the training set, we define the *feature vector* $\bar{f}$ of a formula f as a boolean vector of size p whose i^{th} coordinate is 1 if and only if s_i occurs in f.

Remarks

1. An alternative to this *set of symbols* model is a *bag of symbols* model where each coordinate of $\bar{f}$ is not a boolean but an integer measuring the number of occurences of some symbol. However, the latter is not used in `colt` as it appears empirically that the importance of a symbol occuring in a formula is not significatively correlated with its exact number of occurence.

2. In the case of Coq formulas, what we mean exactly by *symbols* are constants, inductive types and inductive type constructors. In particular, variable names are not considered to be symbols. Besides, some symbols only appear as implicit arguments like it is the case for `nat` in the example above.

3. It is possible to use much richer features to describe formulas. Many options are surveyed in [1] including: types, variable names (they can hold some useful information when coherent naming conventions are used troughout the training corpus) and term walks of length 2 (for instance `"ExpCC _ [^] _"` in the example above). However, as noted in [11] and [10], constant and constructor symbols remain the most useful features to describe formulas and we did not try to include others in `colt` in an effort to keep the feature vector's dimension reasonably small (see Figure 3).

For a given corpus, we build the lemmas database $\{l_1, \cdots, l_k\}$ as the set of all defined constants whose type has sort `Prop`. One of these lemma is considered to be a dependency of a fact from the training set if it occurs in its proof term. Besides, `colt` uses a blacklist system so as to avoid that very frequent properties like `and_ind` or `False_ind` pollute dependencies.

Overall, the training set D can be conveniently represented by two matrices X and Y of dimensions $n \times p$ and $n \times k$ respectively, where the i^{th} line of X is the feature vector of f_i and the corresponding line of Y is its dependency vector y_i. These matrices are typically very sparse and are consequently manipulated using sparse representations whenever possible. In the next few sections, we present different algorithms for learning scoring functions from them.

2.3. The weighted k-NN estimator

The intuition behind the weighted k-nearest-neighbors estimator (k-NN) is pretty simple: when attempting to prove a result, one should use lemmas that proved to

be useful when establishing similar facts.

More precisely, in order to find good premises for a fact f, one sorts the available lemmas in ascending order by respect to their distance to f. In `colt`, the distance between two formulas is defined as the $\mathcal{L}^1$ distance between their feature vectors. Then, we assign to the i^{th} lemma the weight:

$$w_i = \exp\left(-\lambda \cdot \frac{d_i}{d_k}\right)$$

where λ and k are constants and d_j is defined for all j as the distance between f and the j^{th} best lemma. Finally, the score vector estimate is the weighted sum:

$$s(f) = \sum_i w_i y_i$$

Remarks

1. The weights are defined in such a way that a lemma with distance 0 to f is given weight 1 and the k^{th} best lemma is given the constant weight $e^{-\lambda}$. The hyperparameters k and λ are set empirically using grid search. In the `colt` implementation, their default value is 50 and 6.0 respectively.

2. The k-NN estimator is much more relevant when the feature vectors are weighted using an idf scheme: when the symbol s_i appears in a formula f, $\bar{f}_i$ is not set to 1 but to:

   $$\log \frac{n}{|\{j : s_i \text{ occurs in } f_j\}|}$$

   instead, which corresponds to the logarithm of the inverse frequency of s_i in the whole training set. This way, rare symbols in a formula f are given more importance than very common ones like `conj` or `disj` that carry very little information on what f is about. With such a weighting scheme, k-NN estimators are considered as state-of-the-art for premise selection in a recent survey [1].

2.4. The Naive Bayes estimator

Naive Bayes classifiers are a family of classifiers that rely on a strong (naive) independence hypothesis between the features of the objects being classified. Despite

their simplicity, they often give good performances with few training data, which makes them state-of-the art in many classification tasks. In this section, we describe a Bernoulli Naive Bayes classifier in the context of document classification, where it is more easily motivated. Readers that are familiar with this topic are invited to skip this part. Then, we explain how such classifiers can be applied to the problem of premise selection.

2.4.1. Description in the context of document classification

Suppose we want to classify some documents in K different categories. For instance, let's say the documents are journal articles and the categories correspond to different fields like `biology`, `mathematics` or `sociology`. A document is modelled as a set of words and represented by a feature vector whose j^{th} coordinate is 1 if the j^{th} word of the english language occurs in it and 0 otherwise. In order to classify a document, we want to estimate the probability $\mathbf{P}(C = j \mid X = x)$ that it belongs to the j^{th} category given that it has feature vector x for each $i \in \{1, \ldots, K\}$ and pick the category with the highest probability. Using Bayes law, we have:

$$\mathbf{P}(C = j \mid X = x) = \frac{\mathbf{P}(C = j) \cdot \mathbf{P}(X = x \mid C = j)}{\mathbf{P}(X = x)}$$

In order to estimate the probability $\mathbf{P}(X = x \mid C = j)$ that the document has feature vector x given that it belongs to the j^{th} category, we make the strong assumption that the occurence of two different words are independent events when conditioned to a given category. This makes sense because although the occurences of the words `"cell"` and `"genome"` in a document are strongly correlated in general, they are roughly independent if we already know the document is a biology article. With this independence hypothesis, we have:

$$\mathbf{P}(X = x \mid C = j) = \prod_i \mathbf{P}(X_i = x_i \mid C = j) = \prod_i p_{i,j}^{x_i}(1 - p_{i,j})^{1-x_i}$$

where $p_{i,j}$ is the probability that the i^{th} word of the dictionnary occurs in a document of the j^{th} category. We can learn an empirical approximation of the $(p_{i,j})$ probabilities from a large number of already classified documents. For instance, $p_{\texttt{cell,biology}}$ would be approximated by the number of biology articles containing

the word `"cell"` divided by the total number of biology articles in the training set. Similarly, we can get from the training set an estimate of $q_j := \mathbf{P}(C = j)$ for all j. Then, when analyzing a new unclassified document of feature vector x, the Bernouilli Naive Bayes estimate for its category is:

$$\begin{aligned}\hat{j} &= \operatorname*{argmax}_j \mathbf{P}(C = j \mid X = x) \\ &= \operatorname*{argmax}_j \mathbf{P}(C = j) \cdot \mathbf{P}(X = x \mid C = j) \\ &= \operatorname*{argmax}_j q_j \cdot \prod_i p_{i,j}^{x_i}(1 - p_{i,j})^{1-x_i}\end{aligned}$$

2.4.2. Usage for premise selection

A natural way to formulate premise selection as a classification problem is to introduce one category per lemma in the database, the i^{th} category being the category of formulas which proof may involve l_i. Thus, a formula would typically belong to multiple categories (see the first remark below). Then, given a formula f, the score of l_i is proportional to the estimate of $\mathbf{P}(C = i \mid X = \bar{f})$.

Remarks

1. In most presentations, Naive Bayes classifiers are said to work with disjoint categories. The usual way so-called *multi-label* classification problems are solved using Naive Bayes classifiers is a *one-vs-rest* approach where a binary classifier is trained for each category C that separates objects in two disjoint classes, namely the ones that belong to C and the ones that do not. This approach is not optimal for premise selection [11] as the probability estimates given by each classifier are not easily comparable to each other. Thus, we use a single classifier with non-disjoint categories.

2. As it appears in section 2.4.1, the hypothesis of disjoint categories does not appear explicitely in the justification of Naive Bayes classifiers. Overlapping categories just make the independence hypothesis less natural and more questionable. In our setting, this hypothesis translates to the fact that *the occurence of two different symbols in a formula f is independent given*

that a particular lemma is used to prove f, which is not absurd in the case of highly specialized lemmas but clearly wrong for general lemmas like `nat_ind` that can be applied in many different contexts.

3. (∗) In `colt`, the Naive Bayes classifier do not use a Bernoulli event model like the one described in section 2.4.1 but a multinomial model, which performs better. This may sound a little surprising as Bernoulli models are generally considered better in document classification for small documents of the size of the typical formulas that are encountered in mathematical corpora. One possible explanation for this is that it is better to not penalize explicitly the absence of a lemma in the dependencies of a statement as most facts admit many different but equally good proofs.

2.5. Improvements based on topics extraction techniques

The last two estimators we described are both very simple and well estabished in the premise selection community. In this section, we propose to improve them using topics extraction techniques.

2.5.1. Topics extraction and LDA

Latent Dirichlet Allocation (LDA) is a generative graphical model that is well-used in document classification. It is described in more details in [2] and in Appendix A of this report. In a nutshell, LDA enables the following:

1. From a large corpus of unannotated documents and without any other information, extract a list of recurrent *topics*. A topic is a probability distribution over the words of the dictionnary with high weights on a set of semantically closed words. For instance, one topic may put high weights on the words `"plant"`, `"cell"`, `"genome"`, etc ... and could therefore be named *biology*.

2. Given a document, map it to a probability distribution over these extracted topics according to which it is most likely. For instance, an article about GMOs may be mapped to a distribution with high weights on topics like *biology*, *health*, and *politics*.

Given a mathematical corpus, one may extract a set of *mathematical topics* using LDA. We show Figure 2 an example of four topics that were extracted from the CoRN library. For each topic, we print their most frequent inhabitants. Note that the topic names were not generated by the topics extraction algorithm and only consist in a human interpretation of a distribution of words.

In order to use LDA on a mathematical corpus, it is natural to see each fact as a document whose words are the symbols occuring in it. However, we do something a bit more subtle in `colt`. Indeed, each document corresponds to a fact along with its proof and is made of two kind of words: words that correspond to the symbols of the fact's statement and words that correspond to its dependencies. This has the advantage that topics then capture not only correlations between symbols but also between symbols and dependencies and between different dependencies.

2.5.2. Application to premise selection

There are two ways to apply this topics extraction procedure to premise selection.

Using LDA's generative model directly $(*)$ The extracted topics can be summarized in two matrices β^s and β^d of dimensions $T \times p$ and $T \times k$ respectively with T the total number of topics and where $\beta^s_{i,j}$ is the probability that a word in the i^{th} topic is the j^{th} symbol given that it is a symbol and $\beta^d_{i,j}$ is the probability that a word in the i^{th} topic is the j^{th} dependency given that it is a dependency.

Then, given a fact f, we compute from β_s the most likely topic distribution d for f given $\bar{f}$. For the reader familiar with LDA, this is the E-step of the LDA inference algorithm. Finally, the score of l_j is the probability that a dependency of f is l_j given that f has topic distribution d. In other words,

$$(s(f))_j = \sum_i d_i \beta^d_{i,j}$$

or $s(f) = (\beta^d)^T d$ in matrix form.

Improving other classifiers by creating topic features It is possible to improve a classifier C by learning a list of topics from the training set and then concatenating to every feature vector $\bar{f}$ which is sent to C its most likely topic distribution d. Our experiments suggest that d is very informative relative to its

low dimension (typically about 20). For instance, as illustrated in Figure 4 of section 3, a k-NN classifier that receives d as its only feature has non-optimal but decent performances. Besides, d is not redundant with $\bar{f}$ as is shown in section 3.

2.5.3. `colt`'s best estimator

The best estimator of `colt` is a combination of a k-NN estimator (E_1) and a Naive Bayes estimator (E_2), both enhanced with topic features computed using a LDA model. The score given to a lemma l is equal to:

$$-\left[\alpha r_1 + (1-\alpha) r_2\right]$$

where r_1 and r_2 are the ranks given to l by $E1$ and E_2 respectively and α is a hyperparameter (in the current implementation of `colt`, it is set to 0.7).

3. Evaluation and results

We compared and evaluated the different algorithms of `colt` on the CoRN repository [5], a generic mathematical library which was originally written in order to provide a formal constructive proof of the fundamental theorem of algebra. As shown Figure 3, it is composed of about 5,000 theorems and lemmas. As mentioned in section 2.2, the same lemmas are used for both the lemma database and the training set.

Evaluating a premise selection algorithm is a tough task and the only good way to do it might be to measure how it improves the success rate of state-of-the-art automated provers on a large benchmark. Unfortunately, this was not an option for `colt` as no efficient automated prover has been integrated to Coq so far. As a consequence, we extracted from the CoRN library a test set of about 500 lemmas that were not used for training and tried to predict their dependencies. The results are shown Figure 4. For each estimator, we plot its average n-cover on the test set against n, the n-cover of an estimator on a lemma l being the ratio of the dependencies of l that are covered in its n first suggestions. In particular, Figure 4 tells us that on average, about 85% of the dependencies of a theorem lie in a subset of 100 lemmas that are given the highest rank by the best estimator of `colt` among 5,000 CoRN facts.

TOPIC 2	**TOPIC 6**	**Topic 8**	**Topic 13**
Arithmetic	*Metric topology*	*Orders and lattices*	*Binop properties*
Z	MetricSpace	Lattice	associative
Z0	Complete	TotalOrder	right_unit
mul	UniformlyContinuousSpace	PartialOrder	right_inverse
Zdivides	ball	SemiLattice	left_inverse
le	ucFunction	meet	left_unit
gt	stableMetric	join	commutative
modulo	PrelengthSpace	le	right_distributive
Zgcd	approximate	monotone	right_absorbing
lt	ProductMS	min	left_distributive
add	Compact	max	left_absorbing
Zneg	is_RegularFunction	continuous	transitive

The exhaustive topics listing can be generated by the following command:

```
colt topics ./examples/CoRN/features/ -n 15 -p 30
```

Figure 2: A selection of topics extracted from the CoRN repository

	`CoRN`	`mathcomp`
Number of theorems	4,940	13,095
Number of distinct features	90,378	216,542
Avg. number of deps. per theorem	7.0	6.9

Figure 3: Some statistics about the two corpora on which we evaluated `colt`

The same experiment was conducted on the *mathcomp* library, which is roughly three times bigger, as shown Figure 3. Remarkably enough, we observed that for any value of n, the $3n$-cover on *mathcomp* is very close to the n-cover on *CoRN*. As a consequence, 300 lemmas have to be suggested in order to get a 85% cover on *mathcomp*.

Remarks

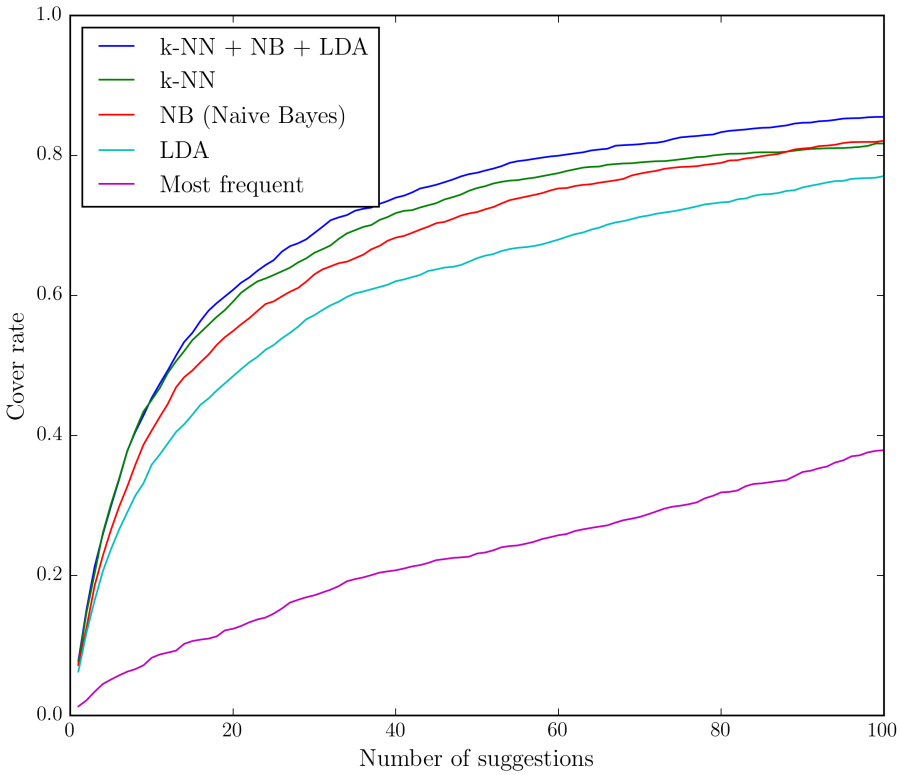
1. The cover curve of an estimator is not a perfect measure of its relevance. Indeed, there is hardly a unique way to prove a fact. Therefore, a suggested lemma not belonging to the dependencies of a statement being proved does not mean it is irrelevant. Besides, it is not necessary that the premise selector reaches a cover of 100% for an automated prover to be succesful.

2. The estimator we described in section 2.5.3 has a 100-cover of about 5 points above simple k-NN and NB estimators. Three of these five points are due to the additional topic features and two others are due to the convex combination between two different classifiers.

3. All the numbers and plots given in this section can be reproduced using the `colt eval` command, which is discussed in Appendix B.

4. Conclusion

Our work adds another evidence to the existing litterature that machine learning provides good heuristics for premise selection [1, 11] and that the Coq proof as-



k-NN+NB+LDA	the best estimator of `colt`, described in section 2.5.3
k-NN	simple k-NN estimator
NB	simple Naive Bayes with multinomial event model
LDA	k-NN with LDA topic distr. as the only feature
Most frequent	always returns the most used CoRN lemmas

Figure 4: Cover curves for different estimators on the CoRN test set

sistant is not very different from Isabelle in this respect [10]. Besides, we have shown that using topics extraction techniques can improve slightly the performance of simple estimators. That being said, the difference is not spectacular and it remains to be seen whether or not it outweights the fact that the modified estimators are about an order of magnitude slower to train.

It may be surprising to the reader that the current state-of-the art techniques in premise selection rely on very simple machine learning techniques like Naive Bayes or k-NN along with very basic features for representing formulas. One possible explanation is that formalized mathematics corpora are scarce resources and that fitting more subtle statistical models of lemma relevance would require more training data than we currently have. Indeed, half of CoRN's lemmas appear in at most three proofs of the same repository, which makes it difficult to learn anything but a very simple model of what they are useful for.

References

[1] Jasmin C Blanchette, Cezary Kaliszyk, Lawrence C Paulson, and Josef Urban. Hammering towards qed. *Journal of Formalized Reasoning*, 9(1):101–148, 2016.

[2] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003.

[3] James P Bridge. *Machine learning and automated theorem proving*. PhD thesis, University of Cambridge, 2010.

[4] Melvin Fitting. First-order logic. In *First-Order Logic and Automated Theorem Proving*. Springer, 1990.

[5] Herman Geuvers, Freek Wiedijk, and Jan Zwanenburg. A constructive proof of the fundamental theorem of algebra without using the rationals. In *International Workshop on Types for Proofs and Programs*, pages 96–111. Springer, 2000.

[6] Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, Ioana Pasca, Laurence Rideau, Alexey Solovyev, Enrico Tassi, and Laurent Théry. A Machine-Checked Proof of the Odd Order Theorem. In Sandrine Blazy, Christine Paulin, and David Pichardie, editors, *ITP 2013, 4th Conference on Interactive Theorem Proving*, volume 7998 of *LNCS*, pages 163–179, Rennes, France, July 2013. Springer.

[7] Thomas C Hales. Developments in formal proofs. *arXiv preprint arXiv:1408.6474*, 2014.

[8] Jónathan Heras and Ekaterina Komendantskaya. Proof pattern search in coq/ssreflect. *arXiv preprint arXiv:1402.0081*, 2014.

[9] Kong Woei Susanto Jia Meng, CLaire Quigley. Linking Isabelle with Automated Theorem Provers. `https://www.cl.cam.ac.uk/~lp15/papers/Automation/`, 2008.

[10] Cezary Kaliszyk, Lionel Mamane, and Josef Urban. Machine learning of coq proof guidance: First experiments. *arXiv preprint arXiv:1410.5467*, 2014.

[11] Daniel Kühlwein, Jasmin Christian Blanchette, Cezary Kaliszyk, and Josef Urban. Mash: machine learning for sledgehammer. In *International Conference on Interactive Theorem Proving*, pages 35–50. Springer, 2013.

[12] Xavier Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7):107–115, 2009.

[13] Alexandre Riazanov and Andrei Voronkov. The design and implementation of vampire. *AI communications*, 15(2, 3):91–110, 2002.

[14] Stephan Schulz. E–a brainiac theorem prover. *Ai Communications*, 15(2, 3):111–126, 2002.

[15] Josef Urban, Geoff Sutcliffe, Petr Pudlák, and Jiří Vyskočil. Malarea sg1-machine learner for automated reasoning with semantic guidance. In *International Joint Conference on Automated Reasoning*, pages 441–456. Springer, 2008.

A. Appendices

A. Description of the LDA model $(*)$

Latent Dirichlet allocation (LDA) is a generative probabilistic model which is used to find underlying topics for a given corpus of documents. More rigorously, a corpus is defined as follows:

- A word is an element of a finite set called *dictionnary*.
- A document is a sequence of N words $w = (w_1, \ldots, w_N)$.
- A corpus is a sequence of M documents $D = (w^1, \ldots, w^M)$.

Denoting by $\mathrm{Dir}(\alpha)$ the Dirichlet distribution of parameter α, the generation of a document is modelled by the following process:

Algorithm 1 LDA: drawing a document

1: The size of the document N is drawn from any fixed distribution
2: A distribution of topics is picked for this document: $\theta \sim \mathrm{Dir}(\alpha)$
3: **for** $1 \leq n \leq N$ **do**
4: Topic $z_n \sim$ Multinomial (θ)
5: $w_n \sim p(\cdot|z_n, \beta) = (\beta(z_n, j))_{1 \leq j \leq V}$

where β denotes a $k \times V$ matrix whose lines are defining topics. A graphical representation of LDA can be found Figure 5. It describes the following factorization:

$$p(\theta, z, w|\alpha, \beta) = \prod_{m=1}^{M} p(\theta^m|\alpha) \prod_{n=1}^{N} p(z_n^m|\theta^m) p(w_n^m|z_n^m, \beta)$$

In order to learn the parameters α and β from a training corpus, we use the EM algorithm, the latent variables being z and θ. For more informations, see [2].

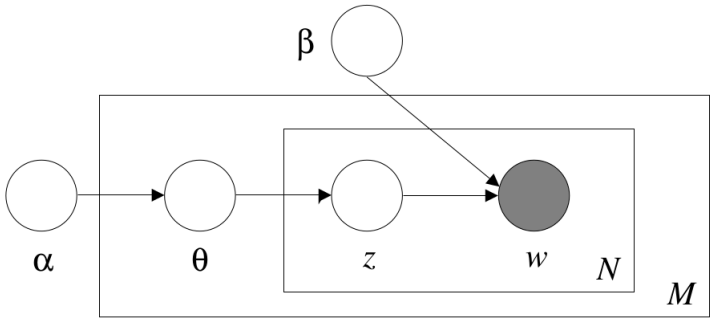


Figure 5: Graphical model for LDA. Source: [2]

B. Installation and basic usage of `colt`

Download The whole source package of `colt` can be found at:

`http://www.eleves.ens.fr/home/jlaurent/software/colt-release.tar.gz`

Requirements

- Coq 8.5 with sources and `coq-dpdgraph` installed
- Ocaml compiler (> 4.02)
- Python (> 3.5.1) with numpy and scikit-learn

Setup In order to build the Coq plugin, just type `make`. The `python3` command should be bound to a python interpreter which is compatible with the requirements above. If not, set the variable `PYTHON` in `colt` manually.

Basic usage with CoRN or mathcomp We give the following instructions for *CoRN* but the same hold for *mathcomp*: just replace `"CoRN"` by `"mathcomp"` everywhere.
First, you need to preprocess the CoRN library. In order to do this, write the path to the CoRN sources directory in the file `examples/CoRN/config/path`. Then, run:

```
./scripts/build.sh ./examples/CoRN
```

After this, you need to fit a premise selector from the sources of CoRN by typing:

```
./colt learn examples/CoRN/features -o ./models/nb
```

Finally, add the following lines in your working Coq file:

- `Colt Set LoadPath "..."` where `...` is replaced by the path of colt on your disk
- `Colt Set Model "nb"`
- `Require Import colt.plugin.plugin`

Now, you can use the tactic `hint` n just after the `Proof` keyword of a statement in order to get n suggestions. An example can be found in the file `examples/Test.v`, which is shown Figure 6. You will need to update the paths in it before you use it though.

Frequently asked questions

1. *How do I use an other library than CoRN or mathcomp ?*
   Just clone the `examples/template` directory and modify its `config` subdirectory. If you don't understand the role of a file, look at the script that uses it in the `scripts` folder.

2. *How can I generate cover curves ?* Use:

   ```
   ./colt eval example/CoRN/features
   ```

   if you followed the CoRN basic usage instructions. Otherwise, consult:

   ```
   ./colt eval --help
   ```

3. *Where can I get more help ?*
   Use `./colt --help` and `./colt [subcommand] --help` or email to `jonathan-laurent@live.fr`

```
(* Directory containing the colt plugin *)
Add LoadPath "~/Stage/colt/plugin".

(* Directory containing the CoRN sources folder:
   not needed if  CoRN was installed using opam *)
Add LoadPath "~/Stage".

Require Import colt.plugin.plugin.

(* Directory containing the 'colt' program *)
Colt Set LoadPath "~/Stage/colt".

(* Name of the mode file that should be loaded in 'colt/models' *)
Colt Set Model "nb".

Require Import CoRN.algebra.CAbGroups.

Lemma lemma (G : CGroup) (x y : G) : x [-] y [=] [--] (y [-] x).
Proof.
hint 10. (* Suggest 10 lemmas using colt *)
Admitted
```

Figure 6: The file `examples/Test.v`