1 Introduction

Automated theorem proving (ATP) is concerned with the development and use of systems that automate sound reasoning: the derivation of conclusions that follow inevitably from facts. This project is specifically concerned with ATP for classical first-order logic, which has well understood and attractive computational properties. ATP lies at the heart of many important computational tasks, and current ATP systems are capable of solving non-trivial problems, e.g., EQP [McC] solved the Robbins problem [McC97], NASA uses ATP systems to certify aerospace software [DFS04], and Microsoft uses ATP to develop and maintain high-quality software [BDF+04]. In practice, however, the search complexity for finding proofs of most interesting theorems is enormous, and many problems cannot currently be solved within realistic resource limits. Therefore a key concern of ATP research is the development of more powerful systems, capable of proving more difficult theorems within the same resource limits.

In recent years the ability of ATP systems to reason over large theories (LTs) – theories in which there are many functors and predicates, many axioms of which typically only a few are necessary for proving that a given conjecture is a theorem, and many conjectures to be proved from a (largely) common set of axioms, has become increasingly important. LT problems (in the context of this project, an ATP problem consists of axioms and a conjecture to be proved) are becoming more prevalent as large knowledge bases, e.g., ontologies and large mathematical knowledge bases, are translated into forms suitable for automated reasoning, e.g., [RPG05, US07, PS07]. The TPTP problem library [SS98], the standard set of test problems for first-order ATP systems, has enjoyed an increasing number of contributions of LT problems, which has confirmed the importance and helped motivate development in this area.

For traditional ATP, LT problems pose several specific challenges, including: system engineering to load, index, and access the very large numbers of formulae (the axioms and conjecture to be proved); finding appropriate search strategies for the search spaces induced by LTs; extracting a small subset of axioms sufficient for proving that a given conjecture is a theorem; proving multiple theorems from a largely common set of axioms; and generating proofs objects found in very large search spaces.

The focus of this project is extracting a small subset of axioms sufficient for proving that a given conjecture is a theorem. Traditional ATP search strategies are swamped by very large numbers of axioms, and an ability to extract a small sufficient subset of axioms in advance enables existing ATP techniques to be used more successfully.
Optimally, a minimal sufficient subset of axioms would be extracted, but theoretical considerations show that this is not possible in general (or if so, $P = NP$). Thus the goal of this project is to develop a system for extracting a small, but not necessarily minimal, subset of axioms sufficient for proving that a given conjecture is a theorem. This will be achieved in the context of multiple conjectures that are to be proved from a common set of axioms.

2 Project Overview

Goal: Given a large theory with multiple conjectures to be proved from a common set of axioms, an ATP system, a CPU time limit on each proof attempt, and proofs of some of the conjectures (i.e., some of the conjectures have already been proved to be theorems, but the others have not due to the large number of axioms swamping the ATP system’s search space, making it unable to find a proof within the CPU time limit), iteratively use machine learning (ML) to learn from existing proofs which axioms are more likely to be used in proofs of further theorems. The ML is used to suggest small sets of axioms that are likely (more likely than a random selection) to be sufficient for proving that a given conjecture is a theorem, so that the ATP system’s search space is reduced, and the system is thus able to find a proof within the CPU time limit. The ATP system and CPU time limit may be specified statically in advance, or may be determined dynamically by the process. The details of implementing this process are given in Section 4.

Two approaches are used for learning and selecting the small sufficient sets of axioms. The first approach uses a measure of similarity between the given conjecture and previously proved theorems. Given such a measure, small sets of axioms are formed by combining the sets of axioms used in the proofs of the previously proved theorems, with preference for those axiom sets used in proving theorems that are more similar to the given conjecture. The ML is thus “nearest neighbor” learning. The details of implementing this approach are given in Section 5. The second approach learns a measure of relevance of axioms to conjecture features. Given such a measure, small sets of axioms are formed by selecting axioms that are more relevant to the given conjecture. The measure of similarity is machine learned using supervised learning on a training set formed from previously proved theorems. Positive training examples have the features of a proved theorem as input and the axioms used in its proof as expected output. Negative training examples have the features of a proved theorem as input and axioms not used in its proof as expected output. The details of implementing this approach are given in Section 6.

3 Project Software

The implementation of this project requires a large theory with multiple conjectures to be proved from a common set of axioms, use of an ATP system, use of a ML system, and a harness to run the various components.

3.1 The TPTPWorld

The TPTPWorld is a package containing the TPTP problem library, ATP systems, and tools for processing ATP problems. The TPTPWorld runs in a UNIX environment, and
has been tested on Linux variants, FreeBSD variants, and Mac OS X. The TPTPWorld is available from http://www.tptp.org/TPTPWorld.tgz. Uses of the various tools in the TPTPWorld for this project are described in Sections 4 to 6. The selection of suitable problems and a suitable ATP system are described here.

The TPTP problem library in the TPTPWorld (in the TPTP directory) is the source for the LT problems. For this project, first-order form (FOF) problems are required, which are identified by a + sign in the file name. The problems are written in a standard format using the TPTP language. An introduction to the TPTP problem format, with examples, is available at http://www.tptp.org/QuickGuide. Each problem has a header that includes statistics on the problem, such as the number of formulae, numbers of predicate and function symbols, etc. Problems in the TPTP (in the Problems directory) may use an include directive to include a common set of axioms (in the Axioms directory). This project can use any set of problems that have many predicate and function symbols, and all include that same large set of axioms, of which typically only a few are necessary for a proof. An example set of suitable LT problems in the TPTP is the set of 422 software creation problems SWC001+1.p to SWC422+1.p in the SWC domain (in the Problems/SWC directory). These problems all include the axioms in the SWC001+0.ax axiom file. The problems have 95 axioms, 20 predicate symbols, and 5 function symbols. State-of-the-art ATP systems are able to prove around two thirds of them with a 600s time limit on a 2.80GHz Intel CPU, e.g., EP 0.999 [Sch02] proves 297, Vampire 9.0 [RV02] proves 267, and SPASS 3.0 [WSH+07] proves 250. There are thus sufficient proofs to learn from, and sufficient unproved conjectures to attempt. The proofs use only a few of the axioms, e.g., EP’s 297 proofs use from none to 27 axioms. The unproved conjectures are expected to also have proofs from a small subset of the axioms.

The ATP system used in this project must be able to produce a proof in the standard TPTP proof format. This allows use of TPTPWorld tools to extract the information required for the ML. An introduction to the TPTP proof format, with examples, is available at http://www.tptp.org/QuickGuide. ATP systems in the TPTPWorld that produce complete TPTP format proofs are EP [Sch02] and Metis [Hur03].

3.2 The SNoW Machine Learning System

The SNoW (Sparse Network of Winnows) learning system is a multi-class classifier that is specifically tailored for large scale learning tasks and for domains in which the potential number of features taking part in decisions is very large, but may be unknown a priori. It learns a sparse network of linear functions in which the targets concepts (class labels) are represented as linear functions over a common feature space. The SNoW system is available from http://l2r.cs.uiuc.edu/~danr/snow.html, and documentation is online at http://l2r.cs.uiuc.edu/~cogcomp/software/snow-userguide/. Use of the ML capabilities of SNoW for this project are described in Section 6.

3.3 Programming Tools

Implementation of this project requires working with formulae, ATP problems, and sets of formulae (the axioms used in a given proof). Additionally it is necessary to execute programs, e.g., the ATP and ML systems, and capture their standard output. It is recom-
mended that the project be implemented in the scripting language perl, which has strong string handling, list handling, and process control capabilities.

4 Controlling the Process

The overall process is formalized as follows: Let ConjectureSet = C₁...Cₙ be the set of conjectures, and AxiomSet = Ax₁...Axₙ be the common set of axioms from which the Cᵢ are to be proved (if there are any axioms specific to a conjecture, they are added to the corresponding ATP problem). At any point in the process, let TheoremSet = T₁...Tᵢ be the Cᵢ that have been proved. Let AxSetTᵢ ⊆ AxiomSet be the axioms used in the proof of Tᵢ. The task is then to learn from the information captured in the ⟨Tᵢ, AxSetTᵢ⟩ pairs, to select a small set of axioms sufficient for proving that some Cₙₑ𝐰 ∈ ConjectureSet, /∈ TheoremSet, is a theorem. If this is achieved then Cₙₑ𝐰 is added to TheoremSet, and the process iterates until no further Cₙₑ𝐰 can be proved. Figure 1 provides the algorithm for implementing this.

```plaintext
1 Let ConjectureSet = C₁...Cₙ be the set of conjectures;
2 Let AxiomSet = Ax₁...Axₙ be the common set of axioms;
3 Attempt to prove each Cᵢ ∈ ConjectureSet from AxiomSet, with MaxCPULimit;
4 Let TheoremSet = T₁...Tᵢ be the Cᵢ that are proved;
5 Run ML on the positive examples ⟨Tᵢ, AxSetTᵢ⟩;
6 Optionally, run ML on the negative examples ⟨Tᵢ, AxiomSet \ AxSetTᵢ⟩;
7 Let CPULimit = MinCPULimit;
8 repeat
9    Mark all Cᵢ ∈ ConjectureSet \ TheoremSet as unattempted;
10   Clear records of axiom sets used for all Cᵢ;
11   while There are unattempted Cᵢs do
12      Let Cₙₑ𝐰 be an unattempted Cᵢ;
13      Mark Cₙₑ𝐰 as attempted;
14         while Cₙₑ𝐰 is not proved and there are unused sets of axioms for Cₙₑ𝐰 do
15             Select unused set of axioms for Cₙₑ𝐰;
16             Record axiom set as used for Cₙₑ𝐰;
17             Attempt to prove Cₙₑ𝐰 from this set of axioms;
18             if Cₙₑ𝐰 is proved then
19                 Add Cₙₑ𝐰 to TheoremSet;
20                 Run ML on the positive examples;
21                 Optionally, run ML on the negative examples;
22                 Mark all Cᵢ ∈ ConjectureSet \ TheoremSet as unattempted;
23            end
24        end
25    end
26   Increase CPULimit;
27 until CPULimit > MaxCPULimit or TheoremSet = ConjectureSet ;
```

Figure 1: Control Algorithm
4.1 Useful Tools

The implementation of the algorithm requires manipulating formulae as items of data – to select axioms from the common axiom set, combine axioms and a conjecture to form a problem, etc. The formulae are supplied in the TPTP files in annotated formulae, using the TPTP language, and formatted so as to be human-readable. Each (annotated) formula has a unique name that can be used as a handle for manipulating the formula. To extract a formula with a given name from a file of formulae, the tptp4X tool (in the ServiceTools directory) is used to reformat each formula onto one line, so that the UNIX grep tool can be used to select the named formula. After that tptp4X can be used again to reformat the selected formula into a human-readable form. tptp4X is used with the command line

```
tptp4X -u machine FormulaFilename
```
to reformat the formula onto one line, and with the command line

```
tptp4X -u human FormulaFilename
```
to reformat into a human-readable form. The FormulaFilename can be -- to apply tptp4X to the standard input stream. Thus, to extract the formula named i_want from the file all_formulae, the command line is

```
tptp4X -u machine all_formulae | grep i_want | tptp4X -u human --
```

Once the required formulae have been manipulated into an ATP problem file, the problem has to be submitted to the chosen ATP system with the given CPU time limit. The SystemOnTPTP tool in the TPTPWorld (in the SystemExecution directory) is used for this. SystemOnTPTP deals with the steps of checking the problem file syntax, adding axioms of equality if necessary, converting from the TPTP syntax to the syntax of the ATP system, running the ATP system with the CPU time limit, extracting the result and output from the system, and converting the output to the TPTP syntax. The command line format is

```
SystemOnTPTP [-qN] SystemName---Version TimeLimit [-S] ProblemFilename
```
e.g., SystemOnTPTP -q4 EP---0.999 30 -S MyProblem.p. The optional -S flag causes the system output to be converted to a TPTP format proof, if possible. The optional -qN flag controls the debug level, with -q0 being most verbose and -q4 least. For this project -q4 is used so that SystemOnTPTP outputs only the TPTP format proof. The problem in ProblemFilename must be in TPTP format.

5 Selecting Sets of Axioms

For each proof attempt on a $C_{new}$, a small set of axioms has to be selected. The process for selecting sets of axioms for a given $C_{new}$ is formalized as follows: Let $TC_{new,1} \ldots TC_{new,p}$ be a reordering of the $T_i \in TheoremSet$ based on their similarity to $C_{new}$, such that $TC_{new,1}$ is most similar to $C_{new}$ down to $TC_{new,p}$ being least similar to $C_{new}$. Let $AxSetTC_{new,1} \ldots AxSetTC_{new,p}$ be the sets of axioms used in the proofs of $TC_{new,1} \ldots TC_{new,p}$. The small sets of axioms used for attempting to prove $C_{new}$ are $AxSetTC_{new,1} \cup \ldots \cup AxSetTC_{new,l}$, for $l = min \ldots max$, for some $min > 0$ and $max < p$, thus providing $max - min + 1$ sets of selected axioms. Figure 2 provides the algorithm for implementing this.

The limits $min$ and $max$ may be specified statically in advance, or may be determined dynamically by the process. One possibility is to place limits on the minimal and maximal
Compute similarity of each \( T_i \in \text{TheoremSet} \) to \( C_{\text{new}} \):

1. Sort \( \text{TheoremSet} \) in decreasing order of similarity to \( C_{\text{new}} \) to produce \( TC_{\text{new},1} \ldots TC_{\text{new},p} \);

2. Set \( \text{SelectedAxioms}_{\text{min}} = AxSetTC_{\text{new},1} \cup \ldots \cup AxSetTC_{\text{new},\text{min}} \);

3. for \( l = \text{min} + 1 \ldots \text{max} \) do
   
   4. Set \( \text{SelectedAxioms}_l = \text{SelectedAxioms}_{l-1} \cup TC_{\text{new},l} \);

end

Figure 2: Selecting Sets of Axioms Algorithm

numbers of axioms that can be selected. The selection algorithm is quite fine grained, in that at each iteration it adds (by union) the axioms used in the proof of only one more theorem. The algorithm can be coarsened to produce fewer sets of selected axioms, by amalgamating the steps for the \( TC_{\text{new},i} \) that have very close (e.g., equal) similarity to \( C_{\text{new}} \).

5.1 Similarity Measures

The similarity of the already proved theorems to the new conjecture can be computed in many ways. The similarity measure suggested for this project is that computed by the \texttt{prophet} tool (in the \texttt{ServiceTools} directory). \texttt{Prophet} is based on simple information retrieval techniques (see [Sut07] for a brief description). The input file for \texttt{prophet} must contain the proven theorems with their role changed from \texttt{conjecture} to \texttt{axiom}, and the new conjecture. (Use the UNIX stream editor \texttt{sed} to change the roles of a file of formulae.) \texttt{prophet} adds a \texttt{relevance}/1 term to the useful information list of each “axiom”, indicating the similarity (“relevance” in other applications) of that formula to the new conjecture formulae. A value of 0.0 means no similarity, and 1.0 means very high similarity (the conjecture itself is also given a \texttt{relevance} term, whose value is naturally 1.0). The command line format is

\texttt{prophet -p FormulaFilename}

The \texttt{FormulaFilename} can be -- to apply \texttt{prophet} to the standard input stream. A file of formulae with \texttt{relevance} terms can be sorted using the \texttt{SortByUsefulInfoField} tool (in the \texttt{ServiceTools/JJUsers} directory). The command line format is

\texttt{SortByUsefulInfoField -f FormulaFilename}

The \texttt{FormulaFilename} can be -- to apply \texttt{SortByUsefulInfoField} to the standard input stream.

5.2 Useful Tools

The axioms used in the proof of a theorem can be extracted from the proof using the \texttt{ProofSummary} tool (in the \texttt{ServiceTools/JJUsers} directory) with the command line

\texttt{ProofSummary -parents ProofFilename}

The \texttt{ProofFilename} can be -- to apply \texttt{ProofSummary} to the standard input stream.
6 Selecting Individual Axioms

For each proof attempt on a $C_{new}$, a small set of axioms has to be selected. The process for selecting individual axioms is formalized as follows: Let $features(F)$ be a vector of features of the formula $F$. For each $T_i \in TheoremSet$, for each $Ax_{T_i,j} \in AxSetT_i$, supply $\langle features(T_i), Ax_{T_i,j} \rangle$ as a positive training example to a ML system. Optionally, for each $T_i \in TheoremSet$, $Ax_{T_i,k} \in AxiomSet \setminus AxSetT_i$, supply $\langle features(T_i), Ax_{T_i,k} \rangle$ as a negative training example to the machine learning system. For a $C_{new}$, supply $features(C_{new})$ to the trained ML system, whose output is then $AxiomSet$ with each $Ax_i$ annotated with a measure of relevance to (the features of) $C_{new}$. Let $Ax_{C_{new},1}, \ldots, Ax_{C_{new},n}$ be a reordering of $AxiomSet$ based on their relevance to $C_{new}$, such that $Ax_{C_{new},1}$ is most relevant to $C_{new}$ down to $Ax_{C_{new},n}$ being least relevant to $C_{new}$. The small sets of axioms used for attempting to prove $C_{new}$ are $\{Ax_{C_{new},1}, \ldots, Ax_{C_{new},l}\}$, for $l = min \ldots max$, for some $min > 0$ and $max < n$, thus providing $max - min + 1$ sets of selected axioms. Figure 3 provides the algorithm for implementing this.

1. Compute relevance of each $Ax_i \in AxiomSet$ to $C_{new}$;
2. Sort $AxiomSet$ in decreasing order of relevance to $C_{new}$ to produce $Ax_{C_{new},1}, \ldots, Ax_{C_{new},n}$;
3. Set $SelectedAxioms_{min} = \{Ax_{C_{new},1}, \ldots, Ax_{C_{new},min}\}$;
4. for $l = min + 1 \ldots max$ do
5. Set $SelectedAxioms_l = SelectedAxioms_{l-1} \cup \{Ax_{C_{new},l}\}$;
6. end

Figure 3: Selecting Individual Axioms Algorithm

The limits $min$ and $max$ may be specified statically in advance, or may be determined dynamically by the process. One possibility is to place limits on the minimal and maximal numbers of axioms that can be selected. The selection algorithm is quite fine grained, in that at each iteration it adds only one more axiom. The algorithm can be coarsened to produce fewer sets of selected axioms, by amalgamating the steps for the $Ax_{C_{new},i}$ that have very close (e.g., equal) relevance to $C_{new}$.

6.1 Formula Features

There are many possible feature vectors for a formulae. The feature vector suggested for this project counts the occurrences of predicate and function symbols that occur in the conjectures. For example, the conjecture of $SWC005+1.p$ has one occur of the function symbol $tl$, three of $app$, five of $nil$, five occurrences of the predicate symbol $neq$, eight of $ssList$, and six of the equality predicate.

The symbols that occur in a formula can be extracted using the GetSymbols tool (in the ServiceTools/JJUsers directory) with the command line

GetSymbols Filename

The Filename can be -- to apply GetSymbols to the standard input stream. The output is a term for each formula in the file, listing the formula name, its function symbols, and its predicate symbols. Each symbol is listed with its arity and number of occurrences (note that same named symbols with different arities are distinct). For example, the output
from running GetSymbols on a file containing just the first axiom and the conjecture of SWC005+1.p is

\[
\text{symbols(ax1, [], \{equal/2/1, neq/2/1, ssItem/1/2\}).}
\]
\[
\text{symbols(co1, [tl/1/1, app/2/3, nil/0/5], \{neq/2/5, equal/2/6, ssList/1/8\}).}
\]

The function symbols in the named formula are listed in the first \[\text{\textbf{boxed}}\text{\textbf{ed}}\text{\textbf{ed}}\text{\textbf{ed list of the term,}}\]
and the predicate symbols are listed in the second \[\text{\textbf{boxed}}\text{\textbf{ed}}\text{\textbf{ed list. A summary of all the symbols}}\]
that occur in a file of formulae can be extracted with the command line

\text{GetSymbols -all Filename}

For example, the output from running GetSymbols -all on a file containing just the first axiom and the conjecture of SWC005+1.p is

\[
\text{symbols(all, [nil/0/5, app/2/3, tl/1/1], [ssList/1/8, ssItem/1/2, neq/2/6, equal/2/7]).}
\]

This usage can be used to extract all the predicate and function symbols that occur in the conjectures, by running it on a file containing all the conjectures.

### 6.2 Using Machine Learning

Machine learning the relevance of the axioms to the features of the theorems can be done in many ways. The SNoW ML system uses a sparse network of sparse linear functions, for which one of several update rules may be used: classical winnow and perceptron, regularized winnow and perceptron, regression algorithms based on gradient descent, and the naive Bayes algorithm. Use of the naive Bayes algorithm is described here, although others might also be suitable for this project.

The input to SNoW is a plain text file of examples, with one example per line. Each example is a comma separated, colon terminated, list of non-negative integer labels. Each integer label is optionally followed by a bracketed real strength value. For example

\[
7(1.5), 5, 10(0.6), 13(-3.2):
\]

If no strength value is given the default strength of 1 is assumed. Each integer label corresponds to a feature of the input or output space. In this project the input features are the function and predicate symbols of the conjectures, and the output features are the axioms. It is therefore necessary to map the predicate and function symbols, and the axioms to integers for use in SNoW - see Section 6.3.

The training examples for this project are built from the \[\text{T}_i \in \text{TheoremSet} - \text{one example per theorem}.\] Each training example lists the symbols in the theorem, with the symbols' strengths being their numbers of occurrences, and all the axioms used in the proof of the theorem, with their strengths being left at the default value of 1. (In this manner all the \[AxT_{used} \text{ referred to at the start of Section 6 are collated into a single training example.}) For example, if the symbols in the conjecture and the axioms used in EP’s proof of SWC005+1.p (see Section 6.3) are mapped to integers as follows, the training example is as shown.

\[
tl/1/1 \rightarrow 47, \text{app/2/3} \rightarrow 8, \text{nil/0/5} \rightarrow 28, \text{neq/2/6} \rightarrow 27, \text{equal/2/1} \rightarrow 11, \text{ssList/1/8} \rightarrow 53,
\]
\[
ax22 \rightarrow 522, ax24 \rightarrow 524, ax78 \rightarrow 578, ax4 \rightarrow 504, ax21 \rightarrow 521, ax82 \rightarrow 582, ax16 \rightarrow 516,
\]
\[
ax84 \rightarrow 584, ax27 \rightarrow 527, ax26 \rightarrow 526, ax81 \rightarrow 581, ax17 \rightarrow 517, ax28 \rightarrow 528, ax15 \rightarrow 515
\]

\[
47(1), 8(3), 28(5), 27(5), 11(6), 53(8), 522, 524, 578, 504, 521, 582, 584, 516, 527, 526, 581, 517, 528, 515:
\]
The SNoW system is trained with the command line
```bash
esnow -train -I TrainingFile -F NetFile -B:AxStart-AxEnd
```
where the `TrainingFile` contains the training examples, `NetFile` is an arbitrary name for a file that stores the trained system parameters, and `AxStart` and `AxEnd` are the integer values corresponding to the first and last axiom labels (the dependent feature labels). This instructs SNoW to learn how the axiom labels depend on the symbol labels, i.e., the relevance of the axioms to the theorem’s symbols.

The query example for a \( C_{\text{new}} \) lists the symbols in the conjecture, with the symbols’ strengths being their numbers of occurrences. For example, using the mapping above, the query example for `SWC005+1.p` is
```plaintext
47(1),8(3),28(5),9(5),11(6),53(8):
```

The SNoW system is queried with the command line
```bash
snow -test -I QueryFile -F NetFile -o allboth
```
where the `QueryFile` contains the query example, `NetFile` is the name of the file containing the trained system parameters, and `-o allboth` specifies the type of output. The output (to `stdout` - it can be redirected to a file using a `-R` flag) is of the form

```
Algorithm information:
Naive Bayes: (0.1) Targets: 0-3
Example 1 Not labeled.
527: 1 0.375 0.375 0.28218
547: 0 0.25 0.25 0.24903
553: 0 0.25 0.25 0.24903
...
```

where the numbers before the colons refer to the axioms in decreasing order of relevance to the conjecture’s symbols, thus providing the required ordering of the axioms. The real numbers in the third to fifth column measures the prediction confidence, raw activation, and softmax normalized activation (see the SNoW documentation for details), which reflect levels of relevance of the corresponding axiom to the symbols.

The ML described so far uses only positive training examples, and rates the relevance of all axioms on that basis. The approach can be extended to include negative examples, and also rate the irrelevance of axioms. This is done by giving each axiom two labels, one for use and one for non-use. In the training examples, the first label is listed if the axiom is used in the proof of the theorem, and the second label is listed if the axiom is not used. In the output the line corresponding to the “use” label measures the relevance of the axiom to the conjecture’s symbols, and the line corresponding to the “non-use” label measures the irrelevance of the axiom to the conjecture’s symbols. If the “use” line occurs before the “non-use” line, then the axiom has been gauged to be more relevant than irrelevant, and vice versa. The differences in the levels of relevance provide combined indicators of relevance. For example, if 27 is the “use” label for an axiom, and 28 is its “non-use” label, this output suggests that the axiom is irrelevant to the conjecture and should not be selected.
Algorithm information:
Naive Bayes: (0.1) Targets: 0-3
Example 1 Not labeled.
28: 1 0.375 0.375 0.28218
42: 0 0.25 0.25 0.24903
36: 0 0.25 0.25 0.24903
27: 0 0.125 0.125 0.24767
...

6.3 Useful Tools

The mapping of axioms to integers for use in SNoW can be done simply by mapping the axioms’ names to integers, using tptp4X and grep as described in Section 4 to extract the axioms from their file. The ProofSummary tool, whose use for extraction of the axioms used in a proof is described in Section 5.2, can be used to extract just the names of the axioms used in a proof. This is done with the command line

```
ProofSummary -prolog ProofFilename
```

For example, the output from running `ProofSummary -prolog` on EP’s proof for `SWC005+1.p` is

```
proved(co1,[ax22,ax24,ax78,ax4,ax21,ax82,ax84,ax16,ax27,ax26,ax81,ax17,ax28,ax15]).
```

The names of the axioms used in the proof are in the `[]`ed list.

7 Deliverables

The deliverables from this project are:

- A perl program that controls the overall process, the selection of sets of axioms, and the selection of individual axioms. The following should be provided:
  - The source code.
  - User documentation.
  - Technical documentation.
  - A document discussing the implementation decisions made, changes that would improve the implementation, and suggestions for extensions to the implementation.

- Results from running the system on several sets of LT problems. For each set the results should provide:
  - The configuration of the system - see the next point for the configuration variable that need to be documented.
  - Information about the axioms and the conjectures - numbers of function and predicate symbols, number of conjectures, number of theorems provable from all the axioms, number of theorems proved from a selected set of axioms.
  - Data showing the progression of new proofs with respect to previous proofs, e.g., numbers of examples learned from, accumulated CPU time for learning and proving, etc.
  - Data regarding the numbers of axioms used in the proofs.
• A report analyzing the results, indicating how choices made in the implementation affect the results. Variation of some of the following aspects should be analyzed:
  – The ATP system.
  – The CPU time limit.
  – Axiom selection - sets vs. individuals.
  – Axiom selection coarseness - degrees of amalgamation of selection steps.
  – Alternative similarity measures - prophet vs. others.
  – Alternative formula features - symbols vs. others.
  – Use of SNoW - Bayes vs. other other update rules; positive vs. positive and negative learning.

References


