Improving Text-mining PPIs by Learning to Filter Entities from the Output of a Knowledge Extractor

Michael Gabilondo

***Abstract*—We propose a rule-based system consisting of (1) the semantic interpreter, a system relying on syntactico-semantic rules specifying the argument structures of verb meanings, and (2) the knowledge extractor, a system relying on semantic extraction rules and syntactico-semantic filtering rules for extracting entities from arguments. We also propose a method to automatically learn to filter entities from the output of the existing rule-based system. The goal is to ease the burden in applying the system to a new domain, or to a similar domain using a gold standard with a differing annotation policy. We show the system with some of the manual rules replaced by the classifier as a post-processor produces comparable results to the original rule-based system, although with slightly more recall but less precision.**

***Keywords*-text-mining; protein-protein interactions; machine learning; rule-based systems**

I. Introduction

Protein-protein interactions (PPIs) are fundamental to almost all of the processes of living cells; as such, their study lies within the domain of several branches of biology [15]. Biologists conducting PPI experiments must first validate interactions from previous research [8]. There are several structured databases containing PPIs reported in the literature which biologists may search [18, 11, 7, 9, 4]. However, these database curation efforts have not kept up with the thousands of new research papers published every day and researchers must still search through the literature for interactions [16, 8].

As a result, growing attention has been directed at information extraction techniques for the automatic construction of PPI databases from the biomedical literature. Most of the PPIs can be found in publications indexed by MEDLINE, which contains about 15 million articles with over 2,000 articles added daily [8, 16]. In this paper, we study the problem of extracting PPIs from English text, which has become the most widely studied problem in the biomedical text-mining domain [10].

We formulate the PPI extraction problem as follows: given a plain-text input sentence, identify the unordered pairs of proteins in the sentence that are stated to chemically interact. Solving the PPI extraction problem involves both identifying protein names in the text and identifying the pairs of proteins that are stated to interact; both of these problems are difficult and have not been completely solved [8]. We assume the protein annotations have been provided by a gold standard or a NER system, and this paper only focuses on the problem of extracting PPIs given known proteins.

Most previous work has focused on optimizing F-score, an average of precision and recall [10]. However, for the application of enriching manually curated PPI databases with automatically extracted PPIs, obtaining high precision is more important than obtaining high recall. Hence, we focus on obtaining high precision while maintaining an acceptable level of recall.

II. Related Work

Previous approaches for solving the PPI extraction problem may be broadly categorized into rule-based extraction methods and machine learning (ML) classification methods.

A. Rule-based PPI extraction

The rule-based methods extract PPIs using rules defined in terms of lexical and syntactic information. Some systems use rules that rely only on words, part of speech (POS) tags, and a dictionary of interaction verbs [2, 14]. A dynamic programming algorithm for automatically discovering rules to extract PPIs has also been proposed [6]. Other approaches rely on parsers that can produce dependency or constituent trees, and rules are defined over such trees. An early approach similar to ours relied on manual mapping rules from argument structures (containing grammatical relations) to frame structures (containing slots/semantic roles) [19]. That approach differs from ours in that the SI attempts to overcome some PP attachment errors produced by the parser and that the KE allows mapping rules to be defined in terms of semantics rather than syntax. Another approach splits the parse tree into clauses and uses a general algorithm for extracting PPIs that relies on a list of interaction terms [1]; 45% of their misclassifications were caused by wrong named entity tags. Another system used a NER system with a dictionary, interaction terms, parsing and manual rules to identify relations [5]. A more recent system used a few simple rules on the output of the parser and achieved very high precision on AIMed (94-97%), although it achieves low recall (15-24%) [10]; the authors propose a more practical evaluation metric that requires, for each PPI, that only one PPI be recovered out of its possibly many identical copies in the text.

B. *Machine learning-based PPI extraction*

The machine-learning methods generally extract features from paths between protein pairs in the parse tree and use SVM to classify feature vectors of PPIs. Some approaches extracted features from the output of Enju, a deep parser that identifies arguments of verbs by linking them to syntactic predicate-argument structures [16] [17]; the latter system also uses an additional parser, and is the state of the art system for extracting PPIs from AIMed (P: 62.7%, R: 66.6%, F: 64.2%). Other systems also used multiple kernels and parsers [12] [13]. A hybrid approach extracted candidate PPIs from the parse tree using high-recall manual rules and then trained and evaluated a classifier that output to to produce a final classification [3].

III. System Description

The system consists of a preprocessor, a semantic interpreter (SI) and a knowledge extractor (KE). The preprocessor produces a list of clauses with their grammatical relations. The SI uses syntactic rules (defined in terms of restrictions on grammatical relations) to determine the arguments and adjuncts (labeled with semantic roles) of each verb as well as to generate relations of nouns. However, such a syntactic analysis of the clause to determine its arguments and adjuncts does not *fully* determine them *or* the meaning of the verb, since semantic rules are required to complete the semantic interpretation (the semantic rules are defined in terms on selectional restrictions on head noun concepts of arguments/adjuncts). The SI is capable of using such syntactico-semantic rules (i.e., restrictions on grammatical relations combined with restrictions on head nouns) to more fully determine the meaning of the verb, but, in our current pipeline, this step is postponed until the KE, where the desired semantic relations and their arguments are extracted from the output of the SI using semantic extraction rules. The KE also extracts individual noun entities from those relations using syntactico-semantic filtering rules and then creates new relations between the individual extracted entities of the arguments. The input to the system is a sentence which has optionally been annotated with named entities.

A. *Preprocessor*

The preprocessor performs the two main tasks of parsing the sentence (i.e., producing a parse tree consisting of phrases using Stanford parser) and producing a minimal clause reconstruction (MCR) of that parse tree. The MCR of the parse tree is performed by the module of the same name, and produces a list of verb clauses, each containing a list of indexes into the parse tree labeled with grammatical relations (e.g., candidate grammatical subjects and candidate NP and PP complements). The MCR also performs the additional tasks of (1) identifying the voice of the verb (i.e., passive or active), (2) identifying whether the clause is (potentially) a relative or not (i.e., a modifier clause that attaches to noun phrase, e.g., “*the proteins that interact* are known”) and (3) identifying NP appositions and NP coordinations by explicitly modifying node labels in the parse tree.

The preprocessor also post-processes the MCR output to (1) produce similar clauses for nominalizations of verbs, (2) produce clause representations which serve as input to the SI and (3) enumerate candidate antecedents of each anaphor. Such a clause produced by the preprocessor has phrases which preserves the hierarchical structure of the parse tree’s phrases, except that the noun phrases (NPs) do not contain attaching prepositional phrases (PPs) or embedded relative clauses. Instead, such PPs and embedded relative clauses (as well as PPs and clauses which attach to the verb syntactically, according to the parse tree) are represented as candidate grammatical relations (the information of the original syntactic attachment to the phrase is preserved). The SI is left to resolve this PP and clausal attachment ambiguity.

B. *Semantic Interpreter*

Let L denote the input to the SI, a list of clauses from the preprocessor for a given parsed sentence. The semantic interpreter (SI) produces an *SI tree*, a list of verb clauses (including nominalizations) generated from L and a list of noun relations generated from those formerly mentioned outputted verb clauses; we refer to these as the outputted clauses/relations of the SI. The predicate and noun ontologies as well as the noun relation rules are used to generate the SI tree. The structure of the SI tree preserves the hierarchical structure of L, except that (1) each clause has been linked to a predicate (possibly the empty, *nil* predicate), (2) grammatical relations of clauses have been replaced by (mapped to) arguments and/or adjuncts (labeled with semantic and/or grammatical roles), (3) head nouns of NPs have been linked to concepts in the noun ontology, (4) PP and clausal attachments to phrases (e.g., NPs) have been determined, and (5) candidate subjects/antecedents have been constricted. Once the SI tree is generated for each of the clauses in L, a list of noun relations are extracted from L using the list of noun relation rules; each generated noun relation is added to the SI tree before being finally outputted.

The noun ontology is a set of hierarchically structured noun concepts, each containing a list of nouns, an optional pattern, and an optional list of parent (i.e., “is-a”) concepts. A concept represents a meaning/sense of each of the nouns in its list of nouns. We say a concept, A, *subsumes* another concept, B, if A can be reached from B by recursively traversing the concepts in the list of parent concepts of each concept. We say a noun *has* a concept in the ontology if that noun is listed in that concept’s list of nouns , i.e., that noun can *mean* that concept. If some noun has no concept explicitly in the noun ontology, then that noun is still assigned to have the default, all-subsuming concept, *thing*. Concepts in the noun ontology also be defined to subcategorize for prepositions; if an NP argument/adjunct is headed by a concept that subcategorizes for some preposition, and a PP headed by such a preposition follows that NP, the SI will override syntactic (parser) attachment of that PP to preferentially attach it to that NP argument/adjunct in the PP attachment phase.

The pattern of a concept is intended to map (to that concept) from nouns that have been pre-tagged with a meaning/sense in a previous processing step, e.g., tagging nouns as named entities using a NER system or a gold standard. If the list of concepts of a noun is requested from the ontology, first the noun is matched against the pattern of each concept that has a pattern; if matches exist, the list of matching concepts is returned as the list of concepts of that noun. Otherwise, each concept that contains that noun in its list of nouns is returned as the list of concepts of that noun.

The predicate ontology is a set of hierarchically structured predicates, each of which represents a potential meaning/sense of a list of verbs. Each predicate contains that list of verbs, a list of of semantic role definitions (classified either as arguments or adjuncts), an optional priority integer, an optional list of required roles and an optional list of parent (i.e., "is-a") predicates. During the execution of the algorithm, each predicate has additional semantic roles, requirements and priorities generated from its parent predicates. There is a predicate ontology for verbs and a predicate ontology for nominalizations of verbs, which allows for different hierarchical structures for the two types of predicates.

Each semantic role definition is used to generate an argument or adjunct semantic role (tree) in the outputted clause linked to the predicate containing that semantic role definition. The arguments/adjuncts are generated from (mapped from) grammatical relations in the inputted clauses, L; a generated argument/adjunct preserves the hierarchical structure of its corresponding inputted grammatical relation, except that PPs and relatives have been attached to phrases. Each semantic role definition consists of a list of grammatical (i.e., syntactic) restrictions (GRs) and selectional (i.e., semantic) restrictions (SRs). The GRs restrict the grammatical relations which can generate the role based on their syntactic features, while the SRs restrict those grammatical relations based on their semantic features. The SRs restrict the head noun's concepts of the NP of the grammatical relation, while the GRs restrict the type of the grammatical relation (e.g., subject or PP), or in other cases, some GRs additionally restrict the types of phrases/tokens which are present in the semantic role (including its attached PPs) which would be generated for the grammatical relation.

If the semantic role definition contains a selectional restriction (SR) other than *thing*, a corresponding semantic role will only be generated if the head noun of the argument has a concept that is subsumed by a concept in the list of SRs (optionally, the list can contain "negated" SRs (prefixed with "-"), indicating the head noun *cannot* have such a concept). If a generated semantic role (i.e., argument/adjunct) was subject to an SR, the list of candidate concepts of its head noun will only contain concepts which are subsumed by the SR (or which are not subsumed, in the case of negated SRs). In any case, all generated noun phrases containing a head noun include a list of the head noun's candidate concepts in the noun ontology. If at least one GR and one SR of a semantic role definition is satisfied by a grammatical relation, a semantic role is generated for that grammatical relation.

1) SI Algorithm Overview: The SI determines the predicate of each clause and its arguments and adjuncts and generates a list of noun relations as a post-processing step. The SI links a clause to a predicate by choosing from the list of candidate predicates for that clause. A predicate is in the set of candidates if it, or any predicate which subsumes it, contains the main verb of the clause in its list of verbs. A candidate predicate contains a set of semantic role definitions, and so a choice of predicate is a choice of semantic roles, i.e., the choice of the predicate of the clause and its arguments are linked.

Let I be the list of candidate SI clauses containing at least one semantic role, such that each clause is generated from a different predicate in the list of candidate predicates. First, each candidate SI clause in I not satisfying the list of required semantic roles is removed from I. If all remaining candidate SI clauses have equal priority (as specified in their definition), then a candidate clause having the largest number of semantic roles (out of all candidate clauses in I) is chosen as the meaning of the verb. On the other hand, if the remaining candidate clauses do not all have the same priority, a candidate clause having the highest priority is chosen as the meaning of the verb. The SI algorithm performs the following main tasks.

a) Generate semantic roles with their attachments for all clauses in L in two passes: [1]

Semantic role definitions contain two kinds of GRs: ordinary GRs and CC GRs. CC GRs test for coordinated NPs or sub-tokens in NP coordinations or tokens of semantic roles and generate a list of arguments containing each coordinated NP or sub-token in the tested constituent. CC GRs have two key differences from ordinary GRs: (1) CC GRs test conditions on a semantic role *including its attached PPs* rather than on a grammatical relation of a clause in L, and (2) if the CC GR is satisfied, a list of semantic roles rather than a single semantic role is generated. These CC GRs require PP attachment to have taken place before they can be verified, but the decision of whether a PP/clause attaches to a phrase requires that each clause already be linked to a

[1]In the current implementation, grammatical roles are also generated within each of of these two passes, but this is not necessary and should be postponed until after generating semantic roles entirely.

predicate and contain semantic roles, since a PP/clause may not attach to any phrase if it is a semantic role. As such, semantic roles are be generated in two passes of the clauses in L.

In the first pass, the SI generates semantic roles with their attachments for all clauses in L; if a semantic role definition of a candidate predicate contains CC GR, that CC GR is not used directly, but is transformed into a less restrictive GR that does not test for conditions in PP attachments.[2] If the first pass has generated a clause containing a semantic role that was produced by being subject to such a reduced-restriction CC GR, then that semantic role and the predicate of the clause are not finally determined until the end of the second pass. In the beginning of the second pass, those generated semantic roles subject to such reduced-restriction CC GRs already contain their PP attachments, and only these kinds of clauses are processed to finally determine if its generated semantic roles satisfy their full-restriction CC GRs. If a semantic role does not satisfy the full-restriction CC GR, then a competing candidate predicate may linked to the clause instead. PP attachment is performed in both passes for the generated roles, but any candidate PP/clause grammatical relation that was used to generate a semantic role in the first or second pass may not attach to any phrase of a role generated in the second pass.

b) Generate semantic roles with their attachments for the clauses to be processed in two passes within either of the two passes described above: For each of the two passes described above, a semantic role for a clause is generated from some grammatical relation of that clause. With some restrictions, a semantic role can also be generated from a grammatical relation of its parent to recover from attachment errors made by the parser. Towards this end, the semantic roles and their attachments are generated also in two passes within either of the passes described above. The first pass generates semantic roles using only post-verbal grammatical relations that are syntactically attached by the parser to the verb to generate a list of grammatical relations mapping to such semantic roles and their attaching PPs and relatives. The list contains grammatical relations that are part of a semantic role generated using the attachment of the parser and that may not be used (in the second pass) by clauses not having that grammatical relation syntactically attached to generate semantic roles. That is, in the second pass, semantic roles are generated for all clauses in L such that a semantic role can also be generated from a grammatical relation that is not syntactically attached to the verb (but that is syntactically and post-verbally attached to its post-verbal parent) and that is not in the list of grammatical relations generated in the first pass.

[2]In the current implementation, for a given clause with candidate predicates containing CC GRs, the first pass only considers those candidate predicates without CC GRs and the second pass *always* reconsiders all of its candidate predicates. This is computationally inefficient, since the partial results from the first pass for these clauses are not used in the second pass.

In either the first pass or the second pass, the following steps are taken to generate semantic roles with their attachments. For each clause to be processed, a predicate (possibly the empty, *nil* predicate) is chosen and its corresponding *argument* semantic roles are generated. PPs and relative clauses are attached to phrases in two phrasal attachment phases: firstly, PPs are attached to NP semantic roles that have a head concept defining a preposition subcategorization (possibly overriding the parser's attachment), and secondly, PPs and relatives are attached to phrases using the syntactic attachment produced by the parser. Let P be a grammatical relation of the inputted clause that is a candidate PP complement or clausal complement. If P is not actually a complement, then it must instead attach to a phrase within a role, and the following steps are taken to attach such a phrase, P, in the second phrasal attachment phase. If a role was generated for P, then P cannot attach to any phrase within a role of that clause or any other clause (however, P may still be an argument of another clause). If a role was *not* generated for P, then the attachment information of P according to the parse tree is used, and P may attach either to the verb or to a phrase within a role of that clause. Following phrasal attachment, the *adjunct* semantic roles (including their phrasal attachments) of the chosen predicate are generated. Adjuncts are generated the same way as arguments, except they cannot be generated for any candidate grammatical relation that is an argument or is attached to a phrase in an argument.

c) Generate grammatical roles with their attachments for all clauses in L: For a given clause, a grammatical role is generated from any grammatical relation that attaches syntactically to the verb, but is not a semantic role and is not attached to any phrase within a semantic role. A grammatical role is generated by first generating a semantic role definition that depends on the type of grammatical relation, and then by generating a semantic role from that definition; phrasal attachment for a such a generated grammatical role is performed in the same way as for a semantic role.

d) Post-process the output to filter antecedents of anaphors generated by the preprocessor and generate new noun relations: Noun relations are generated from the list of noun relation rules of the SI, which, similarly to predicates, contain a list of semantic role definitions, although these definitions contain only GRs. These GRs place restrictions on which phrases of roles can be used to generate semantic roles. Like the CC GRs of predicates, these GRs verify properties of the SI output (in particular, phrases of roles) rather than the preprocessor output (i.e., grammatical relations); some of these GRs are also CC GRs that generate lists of semantic roles from a single phrase. A noun relation rule is defined for a concept in the noun ontology, and is fired on a noun phrase if its head noun has a concept that is subsumed (in the noun ontology) by the concept for which the noun relation rule is defined. Although the noun relation rules are also hierarchically structured, this hierarchy is used only by KE extraction rules to determine whether they can fire on a noun relation.

C. *Knowledge Extractor*

The KE extracts relations from the clauses/relations of the SI in three phases: (1) the KE-1 produces a list of the desired relations, *KE-1 relations*, extracted from the SI clauses/relations using a list of ordered *extraction rules*, (2) the KE-2 produces *KE-2 relations* by transforming its input so that arguments contain extracted noun entities using a list of ordered *filtering rules*, and (3) the KE-3 forms a list of n-tuples of extracted noun entities from each KE-2 relation, say, r, where such an n-tuple represents an instance of the relation, r, between its n noun entities.

Let S denote the input to the KE-1 from the SI, i.e., S is a list of clauses/relations that are linked to predicates/noun relation rules and that contain arguments and adjuncts labeled with semantic or grammatical roles. An extraction rule defines a relation in terms of semantic role definitions; while semantic role definitions of predicates are defined in terms of GRs (i.e., syntax), the semantic role definitions of extraction rules are defined in terms of semantic role labels of predicates and of noun relation rules (i.e., semantics). Semantic role definitions of extraction rules can also define optional SRs that are analogous to the SRs of semantic role definitions of predicates. A semantic role definition, say s, of an extraction rule, say r, also defines *extraction types*, which are are a list of noun concepts that restrict the concepts of the noun entities that the KE-2 can extract from the semantic role(s) generated by s in the KE-1 relation generated by r (multiple semantic roles can be generated from a single semantic role definition, s, if the semantic role of the SI being extracted from is actually a list of coordinated roles as produced by a CC GR). The extraction types of are also used by the KE-1 to filter roles, antecedents of anaphors, and subjects not containing a noun entity having a concept subsumed by an extraction type.

An extraction rule may also be defined for an empty, *nil*, predicate. The semantic role definitions of such a nil extraction rule are *not* defined in terms of semantic role labels of predicates and of noun relation rules. Instead, such a semantic role definition generates a semantic role by matching any role in the SI clause that satisfies the definition's SR (if one is defined; otherwise, any role will be chosen). Such nil extraction rules may be used in cases where the desired relation to be extracted is indicated by the head noun of an argument/adjunct rather than by a verb. An alternative to using a nil extraction rule is to define a noun relation rule in the SI. However, a noun relation rule may fire on the head noun of any noun phrase, not just noun phrases that are roles (i.e., arguments/adjuncts). For this reason, it is intended that a noun relation rule be used in cases when all of the arguments of the relation are within the scope of the NP and that a nil extraction rule be used when one of the arguments is external, i.e., one of its arguments is another argument/adjunct of its clause.

The primary task of the KE-2 is to extract noun entities having a concept subsumed by an extraction type (i.e., extraction entities) from the arguments of KE-1 relations. Rules that filter phrases and tokens within arguments are required by the entity extraction algorithm because even if the arguments of the KE-1 relation are correct, not every extraction entity in every argument may relate to every other extraction entity in every other argument. Although the inputted KE-1 relations should be correct, the KE-2 also contains rules to filter erroneous relations.

Rules to filter tokens, phrases and relations are defined by filter-token, filter-phrase and filter-relation types of filtering rules, respectively. These rules contain a list of conditions and pre-filtering actions to take upon firing the rule but before filtering the constituent. Each rule type defines its own set of primitive condition types which test syntactic properties (e.g., the phrase label) or semantic properties (e.g., a concept of the head noun) of constituents. A rule is constructed by specifying a list conditions and post-filtering actions with their parameters.

The conditions of each filter-phrase (filter-token) rule are tested on each phrase (token) of each KE-1 relation; if the conditions are satisfied, the rule fires, the pre-filtering actions are taken and the phrase (token) is filtered; a tree indicating which rule filtered the phrase (token) is inserted in the outputted KE-2 relation at the tree that corresponds to the tree in the KE-1 relation on which the rule fired (the hierarchical structure of the KE-1 relation is preserved in the KE-2 relation). The conditions of the filter-relation rules are similarly tested on each KE-1 relation to determine if the relation should be filtered.

IV. Learning to Filter Entities

We have developed a list of manually-crafted filtering rules for our gold standard development corpus, AIMed, termed the "manual ruleset". While those filtering rules work well for AIMed, they may not be suitable for extracting PPIs from other corpora with slightly differing annotation policies (e.g., LLL or BioInfer). Our goal is to learn a classifier to filter outputted entities of the system (i.e., present in an argument of a KE-3 relation) as a final post-processing step. To this end, some of the rules in the manual ruleset were replaced by *template rules* to create a new ruleset, the "template ruleset", which preserves the original ordering of the manual ruleset. The KE-2 uses the template rules to generate features for outputted entities. The classifier learns to label each outputted entity either as filter (1) or don't filter (0). An entity is classified as filter (1) if it is involved in a FP relation (e.g., PPI) with respect to the gold standard; it is classified as don't filter (0), otherwise.

A template rule is defined as a filtering rule containing a list of conditions such that the last condition has <T>as its last parameter (a condition instance of a template will only be generated if all previous conditions are satisfied). If the KE-2 tests whether the <T>condition of a template rule satisfies properties of a constituent, that constituent will not be filtered by the template rule, but a list of *condition instances* of that condition will be generated at the tree in the KE-2 relation corresponding to the tree in the KE-1 relation on which that condition was tested. Each condition instance contains the name of the template rule, the name of the condition containing <T>and a parameter for that condition which would cause the constituent to be filtered if such a rule existed as an actual filtering rule.

Any primitive condition (containing a <T>or not) being tested on a constituent first generates a list of all properties extracted from that constituent which could potentially satisfy it; the parameters of the condition are checked against this list of possibilities to determine if the constituent should be filtered. The properties that are extracted from the constituent depend on the condition type. For example, the primitive condition, h-concept, first generates a list of all concepts of the head noun and then checks if its list of parameters satisfies that generated list; a parameter of h-concept is a noun concept prefixed by "-" or not and can specify that a concept of a head noun either be subsumed by the parameter or not. The condition instances of a <T>condition only cause the first step of this process to be carried out, i.e., generating the list of properties extracted from the constituent, but not filtering; however, in addition, the properties in the generated list are outputted as condition instances.

The features of an entity are the condition instances of template filtering rules on the depth-first traversal from the root to that entity. Such a condition instance represents a potential filtering rule, i.e., an instantiation of the template rule with its corresponding condition instance which would cause that entity to be filtered. In addition, the features of an entity also include such condition instances of other entities involved in an outputted relation with that entity. We have defined eight template rules.

1) (filter-relation T1 (cond (ex-rule general-rule-interaction-property) (arg-head <T>)))
2) (filter-phrase T2 (cond (label np np-prn prn np-appos nom-np) (np-child-0 (h-concept <T>))))
3) (filter-phrase T3 (cond (label np np-prn prn np-appos nom-np) (np-child-0 (m-concept <T>))))
4) (filter-phrase T4 (cond (label np np-prn prn np-appos nom-np) (np-child-0 (mod-word <T>))))
5) (filter-phrase T5 (cond (m-concept <T>)))
6) (filter-phrase T6 (cond (mod-word <T>)))
7) (filter-phrase T7 (cond (h-concept <T>)))
8) (filter-phrase T8 (cond (alt-ex-head <T>)))

The condition instances vary depending on the condition, as described below. Let P (R) be the phrase (relation) on which the rule fired if the condition is part of a filter-phrase (filter-relation) rule.

- **arg-head** - All head noun concepts of all arguments of R.
- **m-concept** - All noun concepts of modifiers of the head noun of P.
- **h-concept** - All noun concepts of the head noun of P.
- **mod-word** - All modifier words of the head noun of P that have no noun concept.
- **alt-ex-head** - All "alternate" noun concepts of the head noun of P that has a noun concept with a pattern defined, indicating that head noun has a named entity tag; the "alternate" concepts of such a tagged entity are produced by first removing the tag using its pattern, and then looking up all noun concepts of the stripped word (looking up noun concepts of a named entity tagged word produces only concepts that have a pattern defined, if such a pattern exists to match the word). In addition, the head concept is restricted to the special type, "extraction-type".

V. Experiment

All classifiers were learned using 10-fold stratified cross-validation on AIMed, using nine parts for training and one part for testing in each of the ten folds. We used SVM with a linear kernel with default parameters from the scikit-learn package. Our human judge (a computational biologist) was presented with some sentences from AIMed and candidate PPIs, some of which were stated to interact in AIMed and others which were not. The judge either agreed with the candidate PPI or not. There were two cases for a candidate PPI with which the judge disagreed. If the candidate PPI was stated to interact in AIMed and judge stated it does not interact, then that PPI was added to the inter-annotator disagreement (IAD) file. If the candidate PPI is not stated to interact in AIMed and the judge stated it does interact, then that PPI was also added to the IAD file. To evaluate the impact of training the classifier with entities in the IAD file, we ran experiments with two types of training phases: "Train-IAD", training the classifier with entities in the IAD file, and "No-Train-IAD", not training the classifier with entities in the IAD file. We also ran two types of evaluation phases: "Test-IAD", testing outputted interactions present in the IAD file, and "No-Test-IAD", ignoring outputted interactions present in the IAD file.

Only entities outputted by the system with extracted features may be classified as filter (1). However, entities not outputted by the system, entities with no extracted features and (for "No-Train-IAD" experiments) entities involved in an interaction in the IAD file are all classified as don't filter (0) and have only a single special "nil" feature which no other entities have. There are a total of 692 features

Table I
EXPERIMENTAL RESULTS; I IN THE FIRST COLUMN CORRESPONDS TO "NO-TRAIN-IAD", AND I IN THE FIRST ROW CORRESPONDS TO "NO-TEST-IAD".

	P	R	F	P_I	R_I	F_I
RFECV	88.1	70.5	78.3	93.5	70.9	80.6
RFECV I	85.8	70.6	77.5	93.1	71	80.6
ALL I				92.6	71.1	80.5
ALL I c=.92				94.1	69.4	79.9
Manual	89	68.8	77.6	96.7	69.3	80.8
Template	80.5	70.8	75.3	86.8	71.2	78.2
RFECV+Man				97.9	69.3	81.1

including the "nil" feature. There are 111 entities labeled as filter (1) and 1072 entities labeled as don't filter (0) that have features (and there are an additional 289 entities labeled as don't filter (0) with only the "nil" feature).

Our two baseline systems are the manual ruleset and the template ruleset. The manual ruleset corresponds to the manually-crafted filtering rules for AIMed ("No-Test-IAD", P,R,F: 96.7%, 69.3%, 80.8% and "Test-IAD", P,R,F: 89%, 68.8%, 77.6%) which perform better than the state of the art (F: 64.2%) [17]. The template ruleset is a version of the manual ruleset containing the eight template filtering rules which replace some manual filtering rules, i.e., the rules that the classifier will replace have been removed (P,R,F: 86.8%, 71.2%, 78.2%). The recall of the template ruleset represents an upper bound on recall for the classifier.

We learned two types of SVM classifiers with linear kernels: those using all features, and those using recursive feature elimination with (stratified, 10-fold) cross-validation (RFECV). RFECV is a procedure whereby the least important features are recursively pruned from the entities' feature vectors; each featureset is evaluated using 10-fold cross-validation on AIMed and the featureset that minimizes the number of misclassifications is retained. In the first experiment, we evaluated the accuracy of the classifiers in labeling outputted entities of the system as filter (1) or don't filter (0). We trained two RFECV classifiers and two classifiers using all features; each of the two classifiers of either type is varied with "Train-IAD" and "No-Train-IAD". Surprisingly, all of these configurations yield a classification accuracy of 97%. For the second experiment, we evaluated the overall system output when a classifier was used as a post-processor; those results are shown in Table I. In the table, c=0.92 indicates that an entity is only classified as don't filter (0) if that class has probability 0.92 or greater.

VI. DISCUSSION

While the best RFECV classifier has F-score that is just 0.2% less than the manual ruleset, this classifier produces 3.6% less precision but 1.7% greater recall. RFECV produces only a slight increase in F-score (0.1%) compared to using all features with no probability cutoff. Training with entities in the IAD file or not produces similar results, although not identical. Ignoring outputted interactions in the IAD file produces better results in either case. The classifier only classifiers entities in interactions outputted by the system, i.e., TPs and FPs. Most of the misclassifications of the system come from FNs, so even though the classifier has 0.97% accuracy, the overall improvement in F-score compared with the template ruleset is only 2.4%. Manual rules combined with the RFECV classifier works best; this combination does not reduce recall from the manual ruleset but increases precision by 1.2%.

VII. CONCLUSION

We have proposed a method to learn to filter entities from the output of the Knowledge Extractor using a gold standard. The intention of this classifier is to reduce the workload in applying the system to a new domain. We have shown the classifier produces comparable results with our manual ruleset, but with less precision and greater recall.

REFERENCES

[1] Syed Toufeeq Ahmed, Deepthi Chidambaram, Hasan Davulcu, and Chitta Baral. Intex: A syntactic role driven protein-protein interaction extractor for bio-medical text. In *ISMB BIOLINK SPECIAL INTEREST GROUP ON TEXT DATA MINING AND THE ACL WORKSHOP ON LINKING BIOLOGICAL LITERATURE, ONTOLOGIES AND DATABASES: MINING BIOLOGICAL SEMANTICS (BIOLINK'2005)*, pages 54–61, 2005.

[2] Christian Blaschke, Miguel A. Andrade, Christos Ouzounis, and Alfonso Valencia. Automatic extraction of biological information from scientific text: Protein-protein interactions. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, pages 60–67. AAAI Press, 1999. ISBN 1-57735-083-9.

[3] Quoc-Chinh Bui, Sophia Katrenko, and Peter M. A. Sloot. A hybrid approach to extract proteinprotein interactions. *Bioinformatics*, 27(2):259–265, 2011. doi: 10.1093/bioinformatics/btq620.

[4] Gene Ontology Consortium. The gene ontology (go) database and informatics resource. *Nucleic Acids Research*, 32(suppl 1):D258–D261, 2004. doi: 10.1093/nar/gkh036.

[5] Katrin Fundel, Robert Kffner, Ralf Zimmer, and Satoru Miyano. Relexrelation extraction using dependency parse trees. *Bioinformatics*, 23, 2007.

[6] Minlie Huang, Xiaoyan Zhu, Yu Hao, Donald G. Payan, Kunbin Qu, and Ming Li. Discovering patterns to extract proteinprotein interactions from full texts. *Bioinformatics*, 20 (18):3604–3612, 2004. doi: 10.1093/bioinformatics/bth451.

[7] Ruth Isserlin, Rashad A. El-Badrawi, and Gary D. Bader. The biomolecular interaction network database in psi-mi 2.5. *Database*, 2011, 2011. doi: 10.1093/database/baq037.

[8] Hyunchul Jang, Jaesoo Lim, Joon ho Lim, Soo jun Park, and Kyu chul Lee. Finding the evidence for protein-protein interactions from pubmed abstracts. *Bioinformatics*, 22:2006, 2006.

[9] Minoru Kanehisa and Susumu Goto. Kegg: Kyoto encyclopedia of genes and genomes. *Nucleic Acids Research*, 28(1): 27–30, 2000. doi: 10.1093/nar/28.1.27.

[10] Junkyu Lee, Seongsoon Kim, Sunwon Lee, Kyubum Lee, and Jaewoo Kang. High precision rule based ppi extraction and per-pair basis performance evaluation. In *Proceedings of the*

ACM sixth international workshop on Data and text mining in biomedical informatics, DTMBIO '12, pages 69–76, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1716-0. doi: 10.1145/2390068.2390082.

[11] Michele Magrane and UniProt Consortium. Uniprot knowledgebase: a hub of integrated protein data. *Database*, 2011, 2011. doi: 10.1093/database/bar009.

[12] Makoto Miwa, Rune Saetre, Yusuke Miyao, and Jun'ichi Tsujii. A rich feature vector for protein-protein interaction extraction from multiple corpora. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1 - Volume 1*, EMNLP '09, pages 121–130, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics. ISBN 978-1-932432-59-6.

[13] Makoto Miwa, Rune Stre, Yusuke Miyao, and Junichi Tsujii. Proteinprotein interaction extraction by leveraging multiple kernels and parsers. *International Journal of Medical Informatics*, 78(12):e39 – e46, 2009. ISSN 1386-5056. doi: 10.1016/j.ijmedinf.2009.04.010. URL `http://www.sciencedirect.com/science/article/pii/S1386505609000768`. ¡ce:title¿Mining of Clinical and Biomedical Text and Data Special Issue¡/ce:title¿.

[14] Toshihide Ono, Haretsugu Hishigaki, Akira Tanigami, and Toshihisa Takagi. Automated extraction of information on proteinprotein interactions from the biological literature. *Bioinformatics*, 17(2):155–161, 2001. doi: 10.1093/bioinformatics/17.2.155.

[15] Eric M. Phizicky and Stanley Fields. Protein-protein interactions: Methods for detection and analysis, 1995.

[16] Rune Saetre and Kenji Sagae. Syntactic features for protein-protein interaction extraction. In *Proceedings of the 2nd International Symposium on Languages in Biology and Medicine*, Singapore, 2007.

[17] Rune Saetre, Kazuhiro Yoshida, Makoto Miwa, Takuya Matsuzaki, Yoshinobu Kano, and Jun'ichi Tsujii. Extracting protein interactions from text with the unified akanere event extraction system. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 7(3):442–453, July 2010. ISSN 1545-5963. doi: 10.1109/TCBB.2010.46. URL `http://dx.doi.org/10.1109/TCBB.2010.46`.

[18] Lukasz Salwinski, Christopher S. Miller, Adam J. Smith, Frank K. Pettit, James U. Bowie, and David Eisenberg. The database of interacting proteins: 2004 update. *Nucleic Acids Research*, 32(suppl 1):D449–D451, 2004. doi: 10.1093/nar/gkh086.

[19] Akane Yakushiji, Yuka Tateisi, Yusuke Miyao, and Jun ichi Tsujii. Event extraction from biomedical papers using a full parser. In *Pac. Symp. Biocomput*, pages 408–419, 2001.