Grant Agreement Number: 257528

KHRESMOI

www.khresmoi.eu

Report on Coupling Manual and Automatic Annotation

Deliverable number	*D1.4.2*
Dissemination level	*Public*
Delivery date	*31st May 2013*
Status	*Draft*
Author(s)	*Angus Roberts, Johann Petrak, Niraj Aswani,*

This project is supported by the European Commission under the Information and Communication Technologies (ICT) Theme of the 7th Framework Programme for Research and Technological Development.

Abstract

The Khresmoi project is building a multi-lingual search and access system for biomedical information and documents. The project is using automatic recognition of medical entities in text, such as Diseases and Drugs, to assist with that search. Automatic recognition of these entities is trained by manual correction of machine annotations. Manual correction proceeds iteratively. That is, an initial set of automatic annotations are corrected, these corrections are used to improve the automatic application, this improved application used to generate further annotations for correction, and so on. Improvements are generated in two ways. In the first, an evaluation between the automatic annotations and the corrections is used to drive an error analysis, and thence improvements to application dictionaries, heuristics and grammars. In the second, manual corrections are used to provide entity features for a machine learned statistical model of the entities. Results show the approach to lead to an initial measurable increase in performance, before plateauing.

Table of Contents

Index of Tables

List of figures

List of abbreviations

CUI	Concept Unique Identifier
F1	Harmonic mean of precision and recall, weighted equally for both precision and recall
GATE	General Architecture for Text Engineering
HON	Health on the Net
HTML	Hyper Text Markup Language
IAA	Inter Annotator Agreement
ML	Machine Learning
NLP	Natural Language Processing
P	Precision
PAUM	Perceptron Algorithm with Uneven Margins
POS	Part of Speech
PR	Processing Resource (GATE component)
R	Recall
RadLex	Radiology Lexicon
SPARQL	RDF Resource Query Language
SVM	Support Vector Machine
TUI	Term Unique Identifier (UML ID)
UMLS	Unified Medical Language System
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
USFD	University of Sheffield
UTS	UMLS Terminology Service

1 Executive summary

- The Khresmoi project is using automatic recognition of medical entities in text, such as Diseases and Drugs, to assist with search of biomedical documents.
- Automatic recognition is trained iteratively using manually correction of automatic annotations in text.
- An initial set of automatically created annotations is presented to manual annotators for correction.
- Corrections are used to improve the performance of the automatic application.
- This improved application is then used to generate further automatic annotations for correction, and so on, iteratively.
- This report examines how manual and automatic annotation have been coupled in the Khresmoi project to improve annotation performance.
- It describes how feedback from manual annotation is used to modify automatic annotation through:
  - error analysis and the development of dictionary and heuristic based entity recognition
  - the machine learning of models of the text to assist with automatic annotation.
- A description of the process and its results are presented.

2 Introduction

The Khresmoi project is building a multi-lingual search and access system for biomedical information and documents [17]. Several technologies are used to improve search, including machine recognition of medical entities within text, and the linking of these to the Khresmoi Knowledge Base. Automatic entity recognition is coupled iteratively with manual improvement, in order to drive up performance. In summary, the process for coupling automatic and manual entity recognition is as follows:

1. an initial machine annotated corpus is created;
2. this is corrected by human annotators;
3. the machine entity recognition model is updated with these corrections;
4. the steps are repeated.

This report describes the system used for the above process, iteration of the process, and the results achieved in those iterations.

2.1 Background

Manual annotations are corrected according to a set of guidelines and management protocols, described in Khresmoi deliverable D1.1 “Manual Annotation Guidelines and Management Protocol”[5], which were themselves based on the project requirements, described in Khresmoi Deliverable D8.2 “Use case definition including concrete data requirements” [7]. An early version of the system used to create automatic annotations was described in D1.2 “Initial prototype for semantic annotation of the Khresmoi literature” [2], and early results of the manual annotation described in D1.3 “Report on Results of the WP1 First Evaluation Phase” [1]. The final corpus of corrected annotations are collected into the Khresmoi Manually Annotated Reference Corpus, which is released as D1.4.1 with an accompanying report [6].

2.2 Summary of this report

This report starts with a description of the application used to generate automatic annotations, in the Section “Application description”. This is followed by a description of the process in which manual and automatic annotation are coupled, in the Section “Iterative development process”. Development iterations are then described in the remaining sections. “First iteration – improvement by error analysis” describes the initial iteration, and conclusions that were drawn and improvements made based on an error analysis of early manual corrections. “Second iteration – machine learned corrections” presents the first set of results from machine learned corrections. Finally, the Section “Ongoing iterations” describes the ongoing work to create further sets of automatic annotations for correction and feedback in to the development process.

3 Application description

The annotation software is written as a GATE application pipeline [8, 9]. The initial version of the software, as described in [6], was distributed and run on GATECloud.net [10] and equivalent systems. This initial version was developed into the version described here, in response to (a) analysis of dictionary lookup terms in the application and (b) analysis of initial manual corrections. The version described here may also be deployed via GATECloud.net, or within the Khresmoi crawling and indexing architecture. The application is still subject to development as improvements are suggested by the manual annotation. The final application will be reported by updates to this deliverable, and in future Khresmoi deliverables.

The application can be considered as a number of GATE pipelines, each of which consists of GATE "Processing Resources" (PRs), with each document being run through the pipelines and their component PRs in order. The PRs and their function are described in the table below, after which two sections detail key parts of the application, term lookup and machine learning of entities.

Pipeline	Processing resource	Description
1. Lexico-syntactic pre-processing	Tokeniser	Standard GATE tokeniser,
	Sentence Splitter	GATE regular expression based sentence splitter
	POS Tagger	GATE port of the Brill POS tagger
	Morphological analyser	FLEX based morphological analyser
	Stemmer	Porter stemmer
2. Content demarcation	Check document type	Checks document to see if following content PRs need to be run
	BoilerPipe	GATE wrapper for BoilerPipe HTML content detector
	Content grammars	Content heuristics
3. Stop-words	Gazetteer	Mark all stop-words from gazetteer
4. Term lookup	POS Tag selection grammar	Sets flags on tokens, based on their POS, specifying whether they may be at the start, end or middle of a term
	Gazetteer: abbreviations	Look up abbreviations
	Gazetteer: root forms	Look up root forms of words
	Gazetteer: stem forms	Look up stem form of words
	Gazetteer: string forms	Look up string form of words
	Merging grammar	Merges all looked up terms into a single annotation type, UmlsLookup

	Semantic type to name mapping grammar	Add human readable names for semantic types
5. Term disambiguation	Term selection grammars	Disambiguation heuristics: (1) retain only the longest UmlsLookup, (2) then among all the longest, first select all that come from UMLS preferred labels,(3) then choose among all of those the one with the highest CUI number (see reference [4])
6. Create annotations for correction	Gazetteer	Find all UmlsLookups with semantic types that match those used in the manual annotation guidelines
	Grammars	Post process UmlsLookup annotations into the schema used by Khresmoi
7. Machine Learning	Learning mode SVM, PAUM or other	Model learning only: learn model of Khresmoi entities from manual corrections to UmlsLookups, using UmlsLookups, other UMLS information and lexico-syntactic annotations as features.
	Application mode SVM, PAUM or other	Model application only: apply the above model, using UmlsLookups, other UMLS information and lexico-syntactic annotations as features, to predict Khresmoi entities.

Table 1: Application description

3.1 Term lookup: creating the gazetteers

The main data source used for the automatic semantic annotation of text within Khresmoi is the Unified Medical Language System (UMLS) [3]. Gazetteers in step 4 above were created from UMLS terms and concepts as represented in the Khresmoi knowledge base. For each concept in UMLS, the UMLS provides a unique concept identifier, or CUI. Each concept is also assigned to one of a small number of high level semantic types, from a semantic network. The identifier of this type is known as a TUI.

A preferred label is given to each concept by UMLS, and a number of alternative labels. We refer to these as "prefLabel" and "altLabel" respectively. These form the basis of the gazetteers. As the UMLS source vocabularies are not intended for natural language processing, however, there is much noise amongst these labels. We therefore carry out pre-processing to remove this noise, in line with the methods described by [11,16]. A full description of the gazetteer term processing follows:

1. A SPARQL query was used to retrieve, from the Khresmoi Knowledge Base:
    - all prefLabels and their associated CUI URIs (instance) and direct semantic type classes (TUI URIs) where the language tag of the label was "en". This resulted in 2 399 921 retrieved rows.
    - all altLabels in the same way: 2 447 977 rows
2. After filtering so that only the direct types corresponding to TUIs relevant to Khresmoi remain:
    - prefLabel: 454 127 rows
    - altLabel: 655 490 rows
3. The labels were then filtered as follows:
    - filter out labels that contain an at (@) sign
    - filter labels that contain "not otherwise specified", "unspecified" "[NOS]" and similar
    - filter labels that contain "NEC", "not elsewhere classified", "unclassified" and similar
    - filter very short labels
4. Also, labels were changed in the following ways:
    - remove angular brackets
    - remove multiple spaces
    - remove possessives
    - remove brackets at the end
    - remove parentheses at the end
    - invert labels that have a single comma: e.g. "pain, dorsal" → "dorsal pain"

5. Then a final stage of filtering:
    - remove labels with 6 or more tokens
6. If filtered labels match a pattern for being an abbreviation, the label gets added to an abbreviation list, otherwise to a processed label (standard) list. After this we have:
    - prefLabel standard: 317 619
    - prefLabel abbreviations: 1 763
    - altLabel standard: 561 603
    - altLabel abbreviations: 12 225
7. All labels that match a list of stop words are filtered out, after which we have:
    - prefLabel standard: 317 437
    - prefLabel abbreviations: 1 761
    - altLabel standard: 561 529
    - altLabel abbreviations: 12 172
8. After this, all labels that are not abbreviations are run through GATE, tokenised, POS tagged and stemmed and all word, number and symbol tokens are selected from the processed labels. For each original label, three new labels are created:
    - from all the original strings from the selected tokens
    - from all the roots from the selected tokens
    - from all the stems from the selected tokens
9. and from all of these, three gazetteer list files are created for each of the altLabel and prefLabel lists, giving a total of 8 gazetteer lists:
    - prefLabel, strings: 317 437
    - prefLabel, roots: 317 430
    - prefLabel, stems: 317 430
    - prefLabel, abbreviations: 1 761
    - altLabel, strings: 561 529
    - altLabel, roots: 561 512
    - altLabel, stems: 561 518
    - altLabel, abbreviations: 12 172

3.2 Machine learning of Khresmoi entities

The final stage of the Khresmoi application consists of a GATE machine learning processing resource. This operates in two modes.

In the first mode, training, the machine learning PR uses manual annotations to learn a classifier for the Khresmoi entities. The features for learning instances are provided by the prior steps of the application. In the second mode, application, the machine learning PR applies the classifier to unseen documents, constructing instances for classification from the same features used for learning the classifier.

This is shown in the following diagram, adapted from [12]:

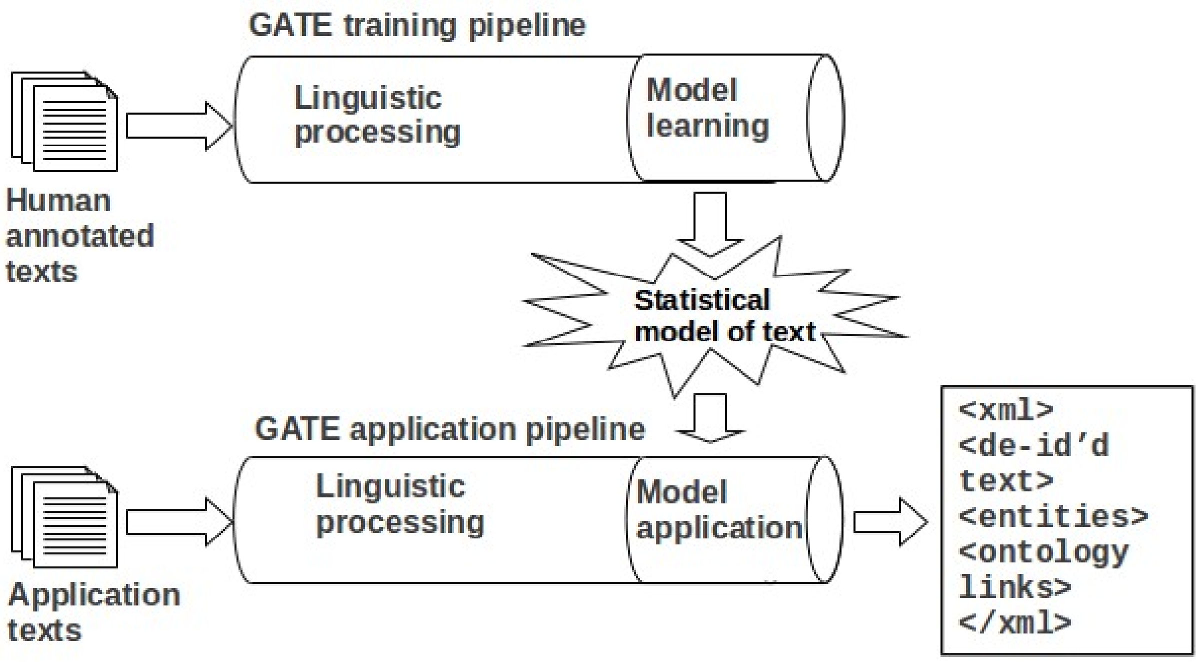

Figure 1: Machine learning application

The GATE machine learning PR provides an abstraction layer over multiple machine learning algorithms. The PR defines how instances should be constructed from text, and what should be used to define features for those instance. It also deals with mapping multi-class problems (such as the one faced with the Khresmoi entities) into a series of binary classification problems. The PR is configured by providing an XML description of the class and of instances, and a definition of the machine learning library to be used.

The machine learning algorithms used for experiments are the Support Vector Machine (SVM) and Perceptron implementations shipped with GATE. The precise algorithm used will be described with each set of results. Both of these algorithms may suffer from imbalanced training data, i.e. the case where tokens comprising entities are less frequent than tokens comprising non-entities. The GATE implementations allow this unevenness to be taken into account, by adjusting the margins between classes in the classifier hyperspace [13].

The features used for constructing classification instances are taken from those described in the following table, precise features will be given with each set of results.

Annotation providing classification instance feature	**Windowing**	**Attribute of annotation used**
Token	From -n to n tokens on each side of the entity, n reported with each set of results	String
		Part of Speech
		Root
		Orthography
UMLS gazetteer lookup	From -n to n tokens on each side of the entity, n reported with each set of results	TUI (semantic type identifier)
		CUI (concept identifier)

Table 2: Outline of machine learning features

4 Iterative development process

Manual and automatic annotation are coupled through an iterative process. This is illustrated in the diagram below, with each step described in the text following the diagram.

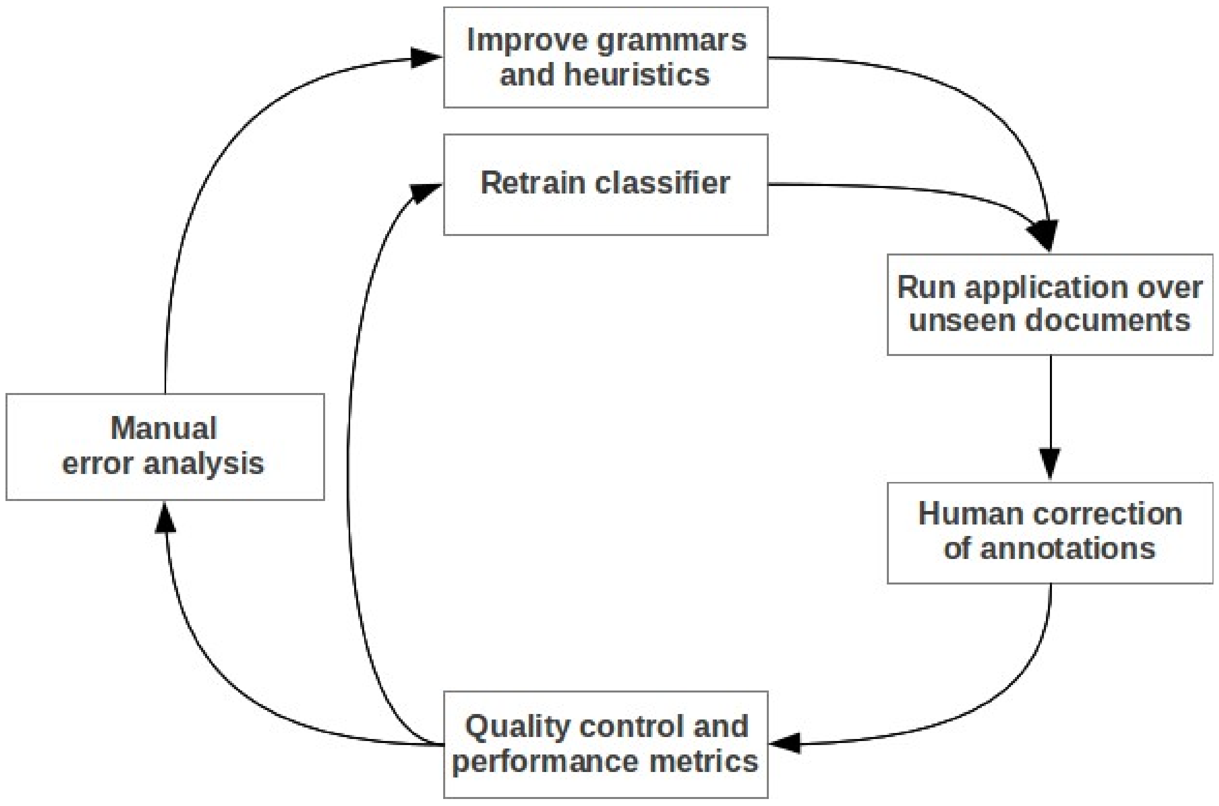
Improve grammars and heuristics
Retrain classifier
Run application over unseen documents
Manual error analysis
Human correction of annotations
Quality control and performance metrics

Figure 2: Iterative development process

Run application over unseen documents

An iteration starts by running the full application over unseen documents. Documents are selected to:

- Be representative of the full Khresmoi document set.
- Be within parameters that make them straightforward for human annotators (e.g. within certain length limits, and having contentful text).
- Reflect problems that need addressing, given the outcome of previous iterations.

Human correction of annotations

Automatically annotated documents are uploaded up to GATE Teamware [14], an annotation workflow web server. GATE Teamware allows complex annotation workflows to be executed online. These workflows may include distributed manual annotation steps, with human annotators being assigned a GATE Teamware account. In the case of Khresmoi, a workflow has three criteria:

- Which documents need annotating
- How many manual annotators should annotate each document – allowing double, treble, or greater levels of annotation.
- Which human annotators should be used for this workflow
- Whether an addition "consensus" step is required.

As annotators login, Teamware assigns documents to them, in order to meet these criteria. If a consensus step is required, then the results from individual annotators will be passed to a further annotator for reconciliation of differences.

Quality control and performance metrics

Once a batch of documents has been completed, an initial analysis is carried out to ascertain the quality of the batch. This includes measurement of inter annotator agreement, and measurement of automatic / manual agreement (i.e. precision and recall). It should be noted that this is not a formal evaluation against a gold standard, but a comparison of annotations to manual corrections. At this stage, some documents or annotators may be rejected as outliers. If no consensus set of annotations has been created in the previous step, annotations may be combined by majority voting – i.e. an annotation will be accepted if it has been accepted by the majority of annotators.

Manual error analysis

The quality controlled batch is then analysed for differences between the automatic and manual sets, using tools from the GATE quality control suite. These include a variety of quality control measurements, together with difference viewers.

Improve grammars and heuristics

The error analysis is used to inform manual improvements to the application's grammars and heuristics. For example, quality control may show that a particular kind of UMLS term is leading to false positives. The heuristics for gazetteer filtering could be amended in response to this. As another example, quality control may show that a only the head word of particular kind of term is annotated automatically, and so grammar rules could be added to improve recognition of the whole term.

Retrain classifier

In addition to the above grammar improvement step, the corrected documents are used to retrain the entity classifier.

5 Summary of iterations

In total, eight iterations of the above process were carried out. These are summarised in the following table:

Iter-ation	Description	Corpus	Input application	Output application
1	Initial experiments using UMLS; no machine learning	Described in deliverable D1.3	Described in deliverable D1.2	First Prototype, described Section 3
2	Initial experiments using machine learning	Initial Gene Home Reference Corpus	First Prototype, described in Section 3	Second Prototype, described in Section 3 (with ML model) V01
3	Knowledge based improvements	C0504	First Prototype, described in Section 3 (V01)	V02
4	Knowledge based improvements	C0512	V02	V03
5	Knowledge based improvements and first ML model for use in future iterations	C0524	V03	• V05 • ML model trained from C0524 corrections
6	Second ML model	C0701	V05 + ML model from previous iteration	V06 + ML model trained from C0701
7	Third ML model	C0830	V06 + ML model from previous iteration	ML model trained from C0830 corrections
8	Fourth ML Model	C0925	V06 + ML model from previous iteration	No further changes

Table 3: Summary of iterations

Note that after the prototype GATE applications, new applications were given version numbers, starting at V02. V04 was a minor version, and was not used in experiments. Iterations are described further in the following sections. Iterations have been grouped in to sections as follows:

First iteration: improvement by error analysis (Section 6)

This iteration took the application and corpus created in the first phase of Khresmoi, used the application to create annotations on the corpus, and examined manual corrections of those annotations to provide an error report. The error report was used to create the first prototype described in Section 3 of this report. The corpus used is described in Khresmoi Deliverable D1.3.

Second iteration – experiments in manual and machine learned corrections (Section 7)

This iteration was used to perform experiments in the machine learning approach, as described in Section 7. The first prototype, as created by the first iteration, was used to annotate the Gene Home Reference corpus described in Khresmoi Deliverable D1.4.1. Manual corrections of this were used to train and evaluate a machine learning model, and to carry out experiments on machine learning feature sets. The results of this evaluation and the experiments are described in Section 7. The corpus used is described in Khresmoi Deliverable D1.4.1.

Iterations 3 to 5: knowledge engineered improvements (Section 8)

The prototype application created in iteration 1 was further improved in iterations 3 to 5, on analysis of errors highlighted by the manual corrections from three corpora. This led to several new versions of the prototype, named V02 to V06. The changes made can be characterised as knowledge engineered changes: all required manual edits to rules, heuristics, and gazetteers. The corpus used is described in Khresmoi Deliverable D1.4.1.

Iterations 6 to 8: machine learned models (Section 9)

Iteration 5 was also used to create a machine learned model of the manual corrections. This was subsequently used with application V06 in iteration 6, to again create annotations for manual correction and generation of a new machine learned model. This process was repeated for iterations 7 and 8. The corpus used is described in Khresmoi Deliverable D1.4.1.

6 First iteration – improvement by error analysis

6.1 Description

The first iteration involved no machine learning of corrected annotations. While the application used was analogous to the non-machine learning parts of the application described above, the definition of annotations created differed, as described in D1.4.1 [6].

During this iteration, issues concerning the use of UMLS arose from an error analysis of manually corrected documents. Although UMLS is a rich source of biomedical terms, its source vocabularies were never created with natural language processing in mind. The use of UMLS for NLP is therefore problematic, and within the first iteration, this was characterised by three overlapping issues:

- High degree of term ambiguity – many terms in UMLS correspond to multiple concepts.
- Low signal to noise ratio – many terms in UMLS can only be understood in the context of their source vocabulary ("heart" for example, may refer to the concept for "mouse heart"). Many others are also ambiguous with words in general language.
- Large numbers of "non-content" terms, such as HTML menus, disclaimers etc., were annotated leading to a large number of irrelevant and false positive annotations.

These issues led to problems with early batches used for manual correction:

1. The more annotations there are, the harder it is for manual annotators to correct the annotations.
2. The number of resulting annotation types makes it difficult to construct meaningful semantic search queries.
3. Text containing meaningful health information content was sometimes swamped by non-content, making it hard for annotators to deal with.

As discussed in [1] the approach we adopted to deal with the first two problems during manual annotation is based upon work reported in [4]. Essentially whenever more than one UMLS annotation is created for the same document span only the annotation with the lowest CUI is retained. The reasoning behind this approach is that the lower the CUI, the more general the concept. Keeping only the most general concept should result in fewer annotations which still encode the same information.

The third problem was dealt with by demarcating content sections of text. In the application described in previous sections, this task is carried out by a GATE BoilerPipe processing resource.

6.2 Results

Corpora and results for this iteration are given in Khresmoi Deliverable D1.3 [1].

7 Second iteration – experiments in manual and machine learned corrections

7.1 Description

The application used for the second iteration resulted from changes made in response to the error analysis of the first iteration, as described in the previous section, and from changes to the gazetteers, after an analysis of gazetteer term lists and of annotated documents from earlier iterations. The application also incorporated machine learning of entities, as a way to take further corrections and feed them forward to further application improvements. These two changes resulted in the application described in the earlier Section "Application description" .

The corpus used for this evaluation consisted of the Initial Corpus described in the accompanying deliverable, D1.4.1 "Manually Annotated Reference Corpus", comprising 625 documents. As there was some change in the definitions of semantic types between this Initial Corpus, and those created by the application, the initial corpus was amended to map the manual annotations to the annotations created by the application, as follows:

1. "Content" annotations were created from "Section" annotations: the Section annotation was used from within the consensus set but if none is found there, one was used from the manual annotation sets;
2. A new annotation type UmlsLookup was created wherever there was a Problem (i.e. Diseases) or Anatomy annotation. This was carried out for manual annotation sets, consensus sets and for the original automatic pre-annotation set.

The UmlsLookup annotations (combined Problem, i.e. Diseases, and Anatomy) could then be compared to the results from the original automatic annotation, and to results from the latest automatic annotation.

For comparisons, the application was also amended to only create semantic annotations of the same UMLS type as in the manual corpus.

7.2 Results

Results given in this and future section use standard definitions of Precision (P), Recall (R), and F1 measure. Two variants of each are given:

- Strict, where for two annotations to match, they must cover exactly the same document span;
- Lenient, where for two annotations to match, they must have overlapping document spans.

The Lenient measure allows us to determine how much error is due not to a failure to find an entity, but to a failure to find the correct extent of the entity (e.g. an application might miss adjectives that are commonly part of the entity label).

The first set of results compares the manual annotations from the Initial Corpus to two non-machine learning applications:

- Initial Prototype: the initial prototype application, from which manual corrections were made
- Second Prototype: the application reported in this document

The results of these evaluations are given in the following table:

Experiment	**Key annotations**	**Response annotations**	**Strict**			**Lenient**		
			P	**R**	**F1**	**P**	**R**	**F1**
First prototype	Manual corrections from first prototype	Automatic annotations from first prototype	0.56	0.60	0.58	0.67	0.72	0.70
Second Prototype	Manual corrections from first prototype	Automatic annotations from second prototype	0.58	0.64	0.61	0.71	0.78	0.74

Table 4: Results of second iteration

The second prototype shows small gains in both precision and recall, for the UMLS semantic types present in the initial corpus. This is presumably due to the improvements in term lookup, through improved gazetteers.

The second prototype was augmented with machine learning as described in the section "Machine Learning of Khresmoi Entities" above, using the GATE Perceptron with Uneven margins algorithm (PAUM). This application was used to predict chunks where a UMLS term should be found, again training and comparing against manual corrections where all annotation types had been merged into a single type. As an initial test of the effectiveness of manual annotations in improving annotation quality, the application was evaluated over two folds with a 75% holdout. Several feature sets were evaluated, as described in the following table:

	Feature set				
	1 TOK	**2 +TUI**	**3 +CUI**	**4 +THR**	**5 -SUR**
Algorithm and options (see GATE manual for option descriptions)	PAUM -p 1 -n 10 -optB 0.3	PAUM -p 1 -n 10 -optB 0.3	PAUM -p 1 -n 10 -optB 0.3	PAUM -p 1 -n 10 -optB 0.3	PAUM -p 1 -n 10 -optB 0.3
POS, window -3 to +3	Yes	Yes	Yes	Yes	Yes
Root, window -3 to +3	Yes	Yes	Yes	Yes	Yes
Kind, -3 to +3	Yes	Yes	Yes	Yes	Yes
TUI, -8 to +8	No	Yes	No	Yes	No
TUI -3 to +3	No	No	Yes	No	Yes
CUI -3 to +3	No	No	Yes	No	Yes
Thresholds	Default	Default	Default	Changed	Default
Surround mode	True	True	True	True	False

Table 5: Second iteration features

The results of evaluations of these feature sets are given below:

Experiment	**Key annotations**	**Response annotations**	**Strict**			**Lenient**		
			P	R	F1	P	R	F1
1 TOK	Manual corrections from first prototype	Automatic annotations from second prototype with PAUM	0.89	0.59	0.71	0.98	0.65	0.78
2 +TUI			0.90	0.65	0.76	0.98	0.70	0.82
3 +CUI			0.91	0.65	0.75	0.98	0.70	0.81
4 +THR			0.91	0.63	0.75	0.98	0.68	0.80
5 -SUR			0.30	0.45	0.36	0.60	0.91	0.72

Table 6: Second iteration feature experiment results

Token level features alone give greater precision and worse recall than in the non-PAUM gazetteer applications reported in the previous table. This is to be expected: the PAUM is able to generalise the context of terms, giving precision in detecting them, without the noise present in a dictionary. It does not, however, have the recall possible with a dictionary. These results are broadly similar to those reported in [15], and the literature referenced there.

Additional UMLS lookup features (experiments 2 and 3) make use of the term dictionaries in UMLS, and show some improvement in recall over token features alone, without loss in precision. The approach in these experiments has successfully combined the precision of the PAUM, with at least some of the recall of a dictionary. In general, performance is greater when combining the PAUM with dictionary lookup. Changing thresholds and algorithm surround mode makes little difference.

8 Iterations 3 to 5: knowledge engineered improvements

8.1 Description

Iterations 3 to 5 followed the same pattern as iteration 1, updating the application gazetteers, pattern matching rules and heuristics in response to an error analysis of manual corrections. These were therefore purely knowledge engineering changes, with no machine learning component.

A different corpus was used for each iteration. The corpora consisted of a mixture of Wikipedia and other Web documents, better reflecting the actual target of the Khresmoi application than the Gene Home Reference corpus used in earlier iterations. Manual corrections were made by five annotators, and a consensus set (majority vote) selected from these. Full details of the corpus are given in Khresmoi Deliverable D1.4.1.

8.2 Results

The table below gives the performance of the applications created at each of these three iterations. In each case, the key annotations are the consensus of annotations from the five independent annotators. Evaluation metrics are true micro averages, i.e. the mean of all measures for each annotation type in each document. Definitions of strict and lenient metrics are as before.

Iter-ation	**Corpus**	**Key annotations**	**Response annotations**	**Strict**			**Lenient**		
				P	**R**	**F1**	**P**	**R**	**F1**
3	C0504	Manual corrections	Annotations from V01 pipeline	0.59	0.86	0.70	0.63	0.91	0.74
4	C0512	Manual corrections	Annotations from V02 pipeline	0.67	0.90	0.77	0.69	0.92	0.79
5	C0524	Manual corrections	Annotations from V03 pipeline	0.81	0.94	0.87	0.82	0.94	0.87

Table 7: Iterations 3 to 5, results

The evaluations show an improvement in Precision with no loss in the already high Recall. Knowledge engineered changes were focussed on correcting false positives. Additionally, it might be expected that manual corrections are more likely to spot false positives than false negatives: it is easier for someone to scan annotations and deleted them where wrong, than to read a full document and add in things that have been missed. This would impact evaluations, potentially giving a falsely raised recall.

9 Iterations 5 to 8: machine learned models

9.1 Description

Iterations 5 to 8 built on the output of iteration 4 by creating machine learned models from manual corrections, in a process analogous to that of iteration 2.

9.2 Results

Following the approach to machine learning from manually corrected annotations in iteration 2, optimal features for machine learning were first selected. Features for testing were grouped in to sets, and models built and evaluated for each of these sets.

The following algorithm configuration options were kept constant (see the GATE user guide for an explanation of each):

• **Algorithm:**	PAUM
• **PAUM parameters:**	p 1 -n 10 -optB 0.3
• **Thresholds:**	default
• **Surround mode:**	true

The PAUM implementation in GATE classifies each token as being either the beginning of an entity, the end of an entity, or neither. Machine learning instances are tokens, machine learning features are defined relative to tokens. Features may be **windowed,** for example, we could consider the part of speech of each of the tokens 3 to the left and 3 to the right of a token to be features of that central token. In the description that follows, windowed features are suffixed thus: [-n to +n], meaning the feature is 2n+1 features, at the token position and n tokens to each side. Where no window is give the feature is constructed at just the token position.

Features were constructed to make use of information added by the knowledge engineered application, such as semantic types and information from the UMLS. It was intended that the information guide machine learning model building. The knowledge engineered application can add multiple entities at a given position, where it is unable to assign an unambiguous meaning. Several features make use of the machine learning framework's default behaviour of choosing one of these entities at random as a feature, whereas other features have been constructed to make a best guess, or to combine unigrams of all possible guesses at this point. These and other features are listed below.

- **POS:** Part of speech of the token at a position
- **Root:** Morphological root of the token at a position
- **Kind:** Orthographic kind of the token at a position (number, word etc.)
- **TUI:** UMLS Type identifier for a term found at underlying the given position by the knowledge engineered application. If several terms exist at the position, then the algorithm chooses one randomly.
- **CUI:** UMLS Concept identifier for a term found at underlying the given position by the knowledge engineered application. If several terms exist at the position, then the algorithm chooses one randomly.
- **TUI / CUI unigrams:** All unigrams of any UMLS terms found to be underlying the given position by the knowledge engineered application.
- **Khresmoi type:** The Khresmoi semantic type (Disease, Investigation, Anatomy, Drug) for a term at a position. If several terms exist at the position, then the algorithm chooses one randomly.
- **Best TUI / CUI / Khresmoi type:** The TUI, CUI, or Khresmoi type of the knowledge engineered application's best guess at which UMLS term underlying the position is correct.
- **Umls_Txxx:** For each UMLS TUI, a binary feature is created that indicates if at this token position, there is an overlapping UMLS term with the TUI.

Feature sets constructed from these features are given in the following table:

	Feature set										
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
POS, window [-3 to +3]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Root, window [-3 to +3]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Kind, [-3 to +3]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
TUI [-3 to +3]	Yes	No	No	No	No	No	No	No	No	Yes	No
CUI [-3 to +3]	Yes	No	No	No	No	No	No	No	No	No	No
TUI and CUI unigrams contained within the term	No	Yes	No	No	No	No	No	No	No	No	No
TUI and CUI unigrams [-3 to +3]	No	No	Yes	No	No	No	No	No	No	No	No
Khresmoi types [-3 to +3]	No	No	No	No	Yes	No	No	No	Yes	Yes	No
Best Khresmoi type [-3 to +3]	No	No	No	No	No	Yes	No	No	Yes	Yes	Yes
Umls_Txxx [-3 to 3]	No	No	No	No	No	No	Yes	No	No	No	No
Best TUI [-3 to +3]	No	No	No	No	No	No	No	Yes	No	No	Yes

Table 8: Iterations 5 to 8, feature sets

Feature set selection was carried out with corpus C0524. For evaluation of feature sets, machine learning was trained twice, on two random splits of 75% of the corpus, and tested on the remaining 25%. The results of the two evaluations were micro averaged, and are given below.

Feature set	**Strict**			**Lenient**		
	P	**R**	**F1**	**P**	**R**	**F1**
1	0.8616	0.6519	0.7420	0.9008	0.6814	0.7757
2	0.4458	0.5088	0.4752	0.6454	0.7367	0.6880
3	0.4458	0.5088	0.4752	0.6454	0.7368	0.6880
4	0.7610	0.1463	0.2454	0.9240	0.1776	0.2979
5	0.8566	0.6307	0.7259	0.9026	0.6641	0.7646
6	0.8519	0.6580	0.7418	0.8955	0.6911	0.7794
7	0.8044	0.3776	0.5139	0.9015	0.4230	0.5757
8	0.8553	0.6480	0.7373	0.8992	0.6811	0.7751
9	0.8583	0.6789	0.7579	0.9006	0.7121	0.7951
10	0.8509	0.6730	0.7514	0.8919	0.7052	0.7874
11	0.8499	0.6725	0.7508	0.8922	0.7057	0.7880

Table 9: Iterations 5 to 8, feature experiment results

Feature set 9 gave the best F measure, as well as balanced P and R measures, and was therefore selected for use in all future machine learning iterations. Three iterations were carried out using these features. As before, the key annotations are the consensus of annotations from five independent annotators. Evaluation metrics are true micro averages, i.e. the mean of all measures for each annotation type in each document.

Iter-ation	Corpus	Key annotations	Response annotations	Strict			Lenient		
				P	R	F1	P	R	F1
6	C0701	Manual corrections	Annotations fromV06 pipeline, model trained on C0524	0.86	0.83	0.84	0.87	0.87	0.86
7	C0830	Manual corrections	Annotations fromV06 pipeline, model trained on C0701	0.98	0.77	0.87	0.99	0.78	0.87
8	C0925	Manual corrections	Annotations fromV06 pipeline, model trained on C0830	0.98	0.57	0.72	0.98	0.57	0.72

Table 10: Iterations 6 to 8: results

Iteration 6 shows a large increase in precision over iteration 5, with a 4% loss in recall. There is an overall gain in F1. Iteration 7 shows a further large increase in precision to 0.99, with a 9% loss in recall. There is a small increase in F1. Iteration 8 shows a further large degradation of recall.

Adding machine learning to the application allows us to improve precision. As is often the case, this is traded against recall.

Considering the machine learning model as a vetoing model provides a possible explanation for the results. The machine learning algorithm is provided with a number of candidate annotations as features, from gazetteer lookup and from the knowledge engineered rules. The model must select one or none of these candidates as positive, thus vetoing the others, and disambiguating the annotations. If the model creation encodes, as is likely, that the candidate annotations provide a high level of information about the correct result, then it will weight these features very highly. It will therefore be unlikely to classify tokens as positive where they do not lie within a candidate annotation, and be unable to create new positives. It will become fitted to those annotations considered positive by the knowledge engineered portion of the application, and be unable to find entities that have never been seen before. Recall will therefore suffer when run over new data in a second or third corpora. As iterations proceed, this is reinforced, and the model becomes over-fitted to the true positives accepted by the manual annotators.

10 Conclusion

This report has described methods for coupling manual and automatic annotation of Khresmoi entities, in an iterative process, in an attempt to drive improvement in the automatic annotation. Improvements are made in two ways: manual changes to grammars and heuristics; and re-training of a classifier.

Both approaches have shown some success. Manual changes to grammars and heuristics resulted in a good improvements in both precision and recall when measured against corrected annotations. Four iterations of classifier re-engineering against maual corrections has shown that precision can be improved with no loss in recall, again when tested against the manually corrected annotations.

Manually corrected annotations have also been used to select sets of features form which to build machine learning classifiers. These classifiers have been used to generate further data for correction, and this iteratively fed back in to classifier training. This did lead to improvements in precision, when measured against manual corrections. Recall, however, suffered, and it is likely that continued iterations are over-fitting the models to the training data. Optimal results were achieved after a single machine learned iteration.

11 References

[1] Niraj Aswani, Liadh Kelly, Mark Greenwood, Angus Roberts, Matthias Samwald, Natalia Pletneva, Gareth Jones, Lorraine Goeuriot. Report on Results of the WP1 First Evaluation Phase, Khresmoi project deliverable D1.3 August 2012.

[2] Mark A. Greenwood, Angus Roberts, Niraj Aswani, Phil Gooch. Initial prototype for semantic annotation of the Khresmoi literature, Khresmoi project deliverable D1.2 May 2012.

[3] Betsy L. Humphreys and Donald A.B. Lindberg and Harold M. Schoolman and G. Octo Barnett. The Unified Medical Language System: an informatics research collaboration. J Am Med Inform Assoc. 1998, 5:1

[4] King, B., Wang, L., et al. (2011). Cenagage Learning at TREC 2011 Medical Track. The Twentieth Text Retrieval Conference Proceedings (TREC 2011), Gaithersburg, MD. National Institute for Standards and Technology.

[5] Angus Roberts, Niraj Aswani, Natalia Pletneva, Celia Boyer, Thomas Heitz, Kalina Bontcheva, Mark A. Greenwood. Manual Annotation Guidelines and Management Protocol, Khresmoi project deliverable D1.1 , February 2012.

[6] Mark A. Greenwood, Angus Roberts, Niraj Aswani, Johann Petrak. Manually Annotated Reference Corpus, Khresmoi project deliverable D1.4.1, May 2013.

[7] Use case definition including concrete data requirements. Khresmoi project deliverable D8.2

[8] H. Cunningham, et al. Text Processing with GATE (Version 6). University of Sheffield Department of Computer Science. 15 April 2011. ISBN 0956599311

[9] H. Cunningham, V. Tablan, A. Roberts, K. Bontcheva (2013) Getting More Out of Biomedical Documents with GATE's Full Lifecycle Open Source Text Analytics. PLoS Comput Biol 9(2): e1002854. doi:10.1371/journal.pcbi.1002854

[10] V. Tablan, I. Roberts, H. Cunningham, and K. Bontcheva. GATECloud.net: a Platform for Large-Scale, Open-Source Text Processing on the Cloud. Philosophical Transactions of the Royal Society A, 371(1983), 2013 doi:10.1098/rsta.2012.0071.

[11] A. McCray, O. Bodenreider, J. Malley, and A. Browne. Evaluating UMLS Strings for Natural Language Processing. In Proceedings of the 2001 American Medical Informatics

Association Annual Symposium, pages 448–452, Portland, OR, USA, 2001.

[12] A. Roberts, R. Gaizauskas, M. Hepple, G.Demetriou, Y. Guo, I. Roberts, and A. Setzer. Building a semantically annotated corpus of clinical texts. Journal of Biomedical Informatics, 42(5):950–66, October 2009

[13] Y. Li, K. Bontcheva and H. Cunningham. Adapting SVM for Data Sparseness and Imbalance: A Case Study on Information Extraction. Natural Language Engineering, 15(02), 241-271, 2009

[14] K. Bontcheva, H. Cunningham, I. Roberts, A. Roberts, V. Tablan, N. Aswani, G. Gorrell. Teamware: A Web-based, Collaborative Text Annotation Framework. Language Resources and Evaluation. In Press, preprint: http://gate.ac.uk/sale/teamware-lre2012/teamware.pdf

[15] A. Roberts, R. Gaizauskas, M. Hepple, and Y. Guo. Combining terminology resources and statistical methods for entity recognition: an evaluation. In Proceedings of the Sixth International Conference on Language Resources and Evaluation, LREC 2008, Marrakech, Morocco, May 2008

[16] A. Aronson. Filtering the UMLS metathesaurus for MetaMap. Technical report, U.S

National Library of Medicine, Lister Hill National Center for Biomedical Communications, Cognitive Science Branch, 2005.

[17] A. Hanbury, C. Boyer, M. Gschwandtner, H. Müller. KHRESMOI: towards a multi-lingual search and access system for biomedical infromation. Med-e-Tel, Luxembourg, 2011.