

Text Annotation Tools Development In Natural Language Processing

A Project Report

Submitted by,

Anshit Kumar Sharma

120103005

Arundhuti Dasgupta

130203022

Atanu Roy

130203026

Deepesh Maskara

130203037

Kajal Giri

130203048

In partial fulfillment for the award of the degree

Of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING



JIS College of Engineering

(An Autonomous Institute)

Block "A" Phase III, Kalyani Nadia-741235

May, 2017

DECLARATION

We hereby declare that the project entitled “Text Annotations Tools Development in Natural Language Processing” submitted for the B. Tech. (CSE) degree is our original work and the project has not formed the basis for the award of any other degree, diploma, fellowship or any other similar titles.

Signature of the Student(s)

Place:

Date:



JIS College of Engineering

Block 'A', Phase-III, Kalyani, Nadia, 741235
Phone: +91 33 2582 2137, Telefax: +91 33 2582 2138
Website: www.jiscollege.ac.in, Email: info@jiscollege.ac.in

CERTIFICATE

This is to certify that **Anshit Kumar Sharma(120103005), Arundhuti Dasgupta(130203022), Atanu Roy(130203026), Deepesh Maskara(130203037) and Kajal Giri(130203048)** have completed their project entitled “**Text Annotation Tools Development in Natural Language Processing**”, under the guidance of **Mr. Apurba Paul** in partial fulfillment of the requirements for the award of the **Bachelor of Technology in Computer Science and Engineering** from JIS college of Engineering (An Autonomous Institute) is an authentic record of their own work carried out during the academic year 2016-17 and to the best of our knowledge, this work has not been submitted elsewhere as part of the process of obtaining a degree, diploma, fellowship or any other similar title.

Signature of the Principal

Signature of the HOD

Signature of the Supervisor

Signature of the External Expert

Place:

Date:

ACKNOWLEDGEMENT

The analysis of the project work wishes to express our gratitude to **Mr. Apurba Paul** for allowing the degree attitude and providing effective guidance in development of this project work. His conscription of the topic and all the helpful hints, he provided, contributed greatly to successful development of this work, without being pedagogic and overbearing influence.

We also express our sincere gratitude to **Mr. Amrut Ranjan Jena**, HOD of the Department of Computer Science and Engineering of JIS College of Engineering and all the respected faculty members of Department of CSE for giving the scope of successfully carrying out the project work.

Finally, we take this opportunity to thank to **Dr. Malay R Dave**, Principal of JIS College of Engineering and **Dr. Somsubhra Gupta**, Dean Academic Affairs, JISCE for giving us the scope of carrying out the project work.

Date:

.....
Anshit Kumar Sharma
B. TECH in Computer Science and Engineering
4th YEAR/8th SEMESTER
Univ Roll--120103005

.....
Arundhuti Dasgupta
B. TECH in Computer Science and Engineering
4th YEAR/8th SEMESTER
Univ Roll—130203022

.....
Atanu Roy
B.TECH in Computer Science and Engineering
4th YEAR/8th SEMESTER
Univ Roll—130203026

.....
Deepesh Maskara
B.TECH in Computer Science and Engineering
4th YEAR/8th SEMESTER
Univ Roll--130203037

.....
Kajal Giri
B.TECH in Computer Science and Engineering
4th YEAR/8th SEMESTER
Univ Roll—130203048

Text Annotation Tools Development In Natural Language Processing

Abstract

Manual annotation is a part which is gaining more and more popularity among the Natural Language Processing community but the recent advancement in this section creates more and more place for errors rather than having a full and absolute solution. This project focuses on having a solution for creating an absolute and correct form of annotation.

The product developed in this project will not only add meaning to the annotation but help in visualization of these annotations. We are using Brat rapid annotation tool to create an extension for the annotations done from this tool.

One of the major advancement of our project will be that each tool will be independent in its processing and platform.

NAME	UNIVERSITY ROLL NUMBER
Anshit Kumar Sharma	120103005
Arundhuti Dasgupta	130203022
Atanu Roy	130203026
Deepesh Maskara	130203037
Kajal Giri	130203048

Table of Contents

	Title page	i
	Declaration of the Student	ii
	Certificate of the guide	iii
	Abstract	iv
1.	Introduction	7
1.1	Natural Language Processing	7
	1.1.1. Importance of Natural Language Processing	7
	1.1.2. Application of Natural Language Processing	7-8
	1.1.3. Difficulties of Natural Language Processing	8
1.2	What is Annotation?	9
	1.2.1. Types of Annotation	9-10
	1.2.2. The Annotation Process	10-11
1.3.	Text Annotation	11
	1.3.1 Applications	11-12
	1.3.2 Web-Based text Annotation	12
2.	Semantic Network	13-14
2.1	Basics of Semantic Networks	13
2.2.	Types of Semantic Networks	14
2.3	Limitations of Semantic Network	14
3.	Resource Definition Framework	15-16
3.1	RDF Triples	15
3.2	RDF Syntax	15-16
4.	Stanford CoreNLP	17
4.1	Why Stanford CoreNLP?	17
5	Brat Rapid Annotation Tool	18-32
5.1	Brat and its features	18-25
5.2	Installation of brat tool in Linux environment	26-27
5.3	Configuring project on brat tool	27-30
5.4	Accessing your project in brat tool	30
5.5	Screenshots of Brat tool	30-32
6	Graphviz	32-33
6.1	Types of files in Graphviz	33
7	Explanation of the Developed Classes	33
7.1	Class : TripleExtractionBrat	33-34
7.2	Class : DotCreation	35-36
7.3	Class : PatterMatcher	36-37
7.4	Class : TregexInterface	37-39
8	Future Scope	40
9	Conclusion	41
10	References	42

1. Introduction

1.1. Natural Language Processing

Natural language processing is a field of computer science, artificial intelligence, and computational linguistics concerned with the interactions between computers and human (natural) languages. As such, NLP is related to the area of human–computer interaction. Many challenges in NLP involve: natural language understanding, enabling computers to derive meaning from human or natural language input; and others involve natural language generation.

Modern NLP algorithms are based on machine learning, especially statistical machine learning. The paradigm of machine learning is different from that of most prior attempts at language processing. Prior implementations of language-processing tasks typically involved the direct hand coding of large sets of rules. The machine-learning paradigm calls instead for using general learning algorithms — often, although not always, grounded in statistical inference — to automatically learn such rules through the analysis of large corpora of typical real-world examples.

Many different classes of machine learning algorithms have been applied to NLP tasks. These algorithms take as input a large set of "features" that are generated from the input data. Some of the earliest-used algorithms, such as decision trees, produced systems of hard if-then rules similar to the systems of hand-written rules that were then common. Increasingly, however, research has focused on statistical models, which make soft, probabilistic decisions based on attaching real-valued weights to each input feature. Such models have the advantage that they can express the relative certainty of many different possible answers rather than only one, producing more reliable results when such a model is included as a component of a larger system.

1.1.1. Importance of Natural Language Processing

We interact with technology every single day of our lives, but humans and computers communicate in fundamentally different ways. Whereas we tell stories, take notes, and relay information through a narrative, computers rely on objective data and commands. Bridging this gap is the root of natural language processing.

Natural language processing is considered important because natural language is important. Consider a relatively early entrance of machines into a domain: ATMs. In order to supply money through a machine, one had to build a model of bank accounts, enable mechanical handling of banknotes, build a filling mechanism, handle ATM use cases, security and many things I am unaware of.

1.1.2. Applications of Natural Language Processing

Natural language processing has tremendous applications in a lot of fields which has made our life easier, some of them are as follows-

1.1.2.1. Machine Translation

As the world's information is online, the task of making that data accessible becomes increasingly important. The challenge of making the world's information accessible to everyone, across language barriers, has simply outgrown the capacity for human translation. Innovative companies like Duolingo are looking to recruit large amounts of people to contribute, by coinciding translation efforts with

learning a new language. But machine translation offers an even more scalable alternative to harmonizing the world's information. Google is a company at the forefront of machine translation, using a proprietary statistical engine for its Google translate service.

1.1.2.2. Fighting Spam

Spam filters have become important as the first line of defense against the ever-increasing problem of unwanted email. But almost everyone that uses email extensively has experienced agony over unwanted emails that are still received, or important emails that have been accidentally caught in the filter. The false-positive and false-negative issues of spam filters are at the heart of NLP technology, again boiling down to the challenge of extracting meaning from strings of text.

1.1.2.3. Information Extraction

Many important decisions in financial markets are increasingly moving away from human oversight and control. Algorithmic trading is becoming more popular, a form of financial investing that is entirely controlled by technology. But many of these financial decisions are impacted by news, by journalism which is still presented predominantly in English. A major task, then, of NLP has become taking these plain text announcements, and extracting the pertinent info in a format that can be factored into algorithmic trading decisions.

1.1.2.4. Summarization

Information overload is a real phenomenon in our digital age, and already our access to knowledge and information far exceeds our capacity to understand it. This is a trend that shows no sign of slowing down, and so an ability to summarize the meaning of documents and information is becoming increasingly important. This is important not just in allowing us the ability to recognize and absorb the pertinent information from vast amounts of data.

1.1.2.5. Question Answering

Search engines put the world's wealth of information at our fingertips, but are still generally quite primitive when it comes to actually answering specific questions posed by humans. Google has seen the frustration this has caused in users, who often need to try a number of different search results to find the answer they are looking for. A big focus of Google's efforts in NLP has been to recognize natural language questions, extract the meaning, and provide the answer, and the evolution of Google's results page has shown this focus.

1.1.3. Difficulties in Natural Language Processing

There are a lot of difficulties and challenges that we need to overcome in the field of natural language processing, some of them are listed as follows-

1.1.3.1. Ambiguity in language

English language is very ambiguous, lot of words in English language have multiple meanings and it is sometimes difficult to understand which meaning is being conveyed at the time of being used. For example, duck may refer to the animal as well as crouching, another example is light, it may refer to something being light or the source of luminosity.

1.1.3.2. Meaning is context sensitive

In a sentence, each word may convey different meaning but for the sentence itself the meaning is implicated by collectively analyzing the sentence. For example, the boy caught a frog with a cap; this sentence may either imply that the boy caught the frog in a cap or the boy caught a frog which has a cap.

1.1.3.3. It is not accurate

The processing may not be accurate, the results obtained may differ from the expectations as there are lot of parameters to check and also the ambiguity.

1.1.3.4. Metaphor or sarcasm detection

The algorithm created may not be capable to detect metaphors or sarcasm and even if it could then it would be difficult to be accurate.

1.2. What is Annotation?

Annotation is a methodology for adding information to a document at some level, a word or phrase, paragraph or section or the entire document. This information is “metadata,” that is, data about other data. The difference between annotation and other forms of meta-data is that an annotation is grounded to a specific point in a document. For example, one might consider a folder name on a computer as meta data for the files in that folder. So, a folder labeled “holiday 2008” might hold files of photographs taken on holiday. The folder name is a form of metadata. But, when an image file is taken out of the folder, it becomes separated from that meta data and thus loses some valuable context. Many desktop image managers compensate by including meta-data tags that can be added to the image file. Thus, the data stays with the image, but you need a specialized picture viewer to see that meta-data and use it to sort, categorize or find pictures. Some programs allow tags to be applied to specific regions of an image. Facebook has such an application, but such tags are useful only in Facebook and are not really a part of the image file. The Facebook tags are more like folder names, as they are not really anchored to the underlying data. When we annotate text files for use in linguistic analysis, semantics, business intelligence, enterprise search and other applications, we apply the concept of annotations to specific ranges of text within a document.

1.2.1.Types of Annotation Projects

There are many types of annotation projects, some of which do not fit neatly into categories. Below are four examples that describe the most common types of projects. The general guidelines cover all of these projects, but the specific guideline processes may be quite different.

1.2.1.1. Gold Standard

The goal of the Gold Standard is to create a top threshold for measuring computer performance through manual annotations applied to the same text by more than one person. This is often used at the outset of a project to see how often two or more annotators agree on a particular annotation or type of annotation. When the independent annotators reach some level of agreement, maybe 85 of the time, that becomes the target for a computer-generated annotation.

1.2.1.2. Quality Assurance

A quality assurance project often looks for errors in previously-processed documents. The errors may either be corrected, or simply collected and analyzed, depending on the need. A quality assurance process may also create its own gold standard on a subset of all documents—a mini corpus—to target specific problems. Quality Assurance processes often seek 100% accuracy, unlike other processes, but on small, manageable sets of documents.

1.2.1.3. Processing

A processing project may seek to add annotations that would be used in a processing step and then could be removed. For example, a processing project might highlight errors in a document and provide corrective labels which could be used to make automatic corrections in the original and then stripped from the text itself. Or, a processing annotation could be used in automatic routing of a document in an information management system.

1.2.1.4. Social

Social annotation (or “social tagging”) is becoming increasingly popular both on the Web and within organizations. Often social tagging is an attempt to arrive at a new nomenclature or to accommodate bottom-up terminology (AKA “folksonomy”) with a more formal system of names and labels. Social annotation might uncover new popular trends, or help with navigation, or generate civil discussion of various topics. In general, there is no attempt to constrain the annotations or how they are applied, but to harvest the various inputs to see if/when patterns emerge.

1.2.2. The Annotation Process

It would be possible to read a document from start to end, marking all annotations in order, as they are found. This has not, however, been found to give the most accurate results. In order for annotations to be consistent, a more methodical approach has to be applied, according to which all annotators should perform these actions in this particular order:

1.2.2.1. Read the whole document

Read the document through in its entirety, marking no annotations, to get an understanding.

1.2.2.2. Mark the entities

Read the document a second time, adding annotations for the mentions (including pronouns) of these basic entities, (in parallel if it is easier). April 2010 Introduction to Manual Annotation.

1.2.2.3. Look again

Review your work to be certain that you have not missed anything and especially that the annotation types and features are correct. Certain entities may suggest that other also exist, so also look for these if appropriate.

1.2.2.4. Record any additional information

As you are annotating the document, record any comments that you feel are important. You may have to do this in some text file/wiki, or perhaps in the annotation tool itself.

For example, record:

1. Whether an annotation decision was hard to make.
2. Any questions or uncertainty you may have about the annotations and guidelines.
3. Anything unclear or ambiguous in the guideline.
4. Things that you consider important, which were not covered by the annotations allowed.
5. Annotation tool bugs and/or issues.

1.3. Text Annotation

Text Annotation is the practice and the result of adding a note or gloss to a text, which may include highlights or underlining, comments, footnotes, tags, and links. Text annotations can include notes written for a reader's private purposes, as well as shared annotations written for the purposes of collaborative writing and editing, commentary, or social reading and sharing. In some fields, text annotation is comparable to metadata insofar as it is added post hoc and provides information about a text without fundamentally altering that original text. Text annotations are sometimes referred to as marginalia, though some reserve this term specifically for handwritten notes made in the margins of books or manuscripts. Annotations are extremely useful and help to develop knowledge of English literature.

A well-annotated text will accomplish all of the following:

- clearly identify where in the text important ideas and information are located
- express the main ideas of a text
- trace the development of ideas/arguments throughout a text
- introduce a few of the reader's thoughts and reactions

1.3.1. Applications

Text annotations can serve a variety of functions for both private and public reading and communication practices. Readers annotate texts depends on the purpose, motivation, and context of reading.

1.3.1.1. Educational applications

Educational research in text annotation has examined the role that both private and shared text annotations can play in supporting learning goals and communication. Much educational research examines how students' private annotation of texts supports comprehension and memory; for example, research indicates that annotating texts causes more in-depth processing of information, which results in greater recall of information.

1.3.1.2. Social Reading

Although reading itself is often a solitary experience, taken more broadly, reading can nevertheless be viewed as a very social activity. From individuals sharing their perspectives in a book club to students discussing a particular passage in class, reading often involves a significant social component. Furthermore, even if readers do not engage in such explicit social behavior, they will often carry out a conversation with the text itself, such as questioning an author's argument or agreeing with a particular section. In one sense, annotations may be considered the currency of such interactions.

1.3.1.3. Writing and text-centered collaboration

Text annotations have long been used in writing and revision processes as a way for reviewers to suggest changes and communicate about a text. In book publishing, for example, the collaboration of authors and editors to develop and revise a manuscript frequently involves exchanges of both in-line revisions or notes as well as marginal annotations. Similarly, copyeditors often make marginal annotations or notes that explain or suggest revisions or are directed at the author as questions or suggestions (commonly called "queries").

1.3.2. Web based text annotation

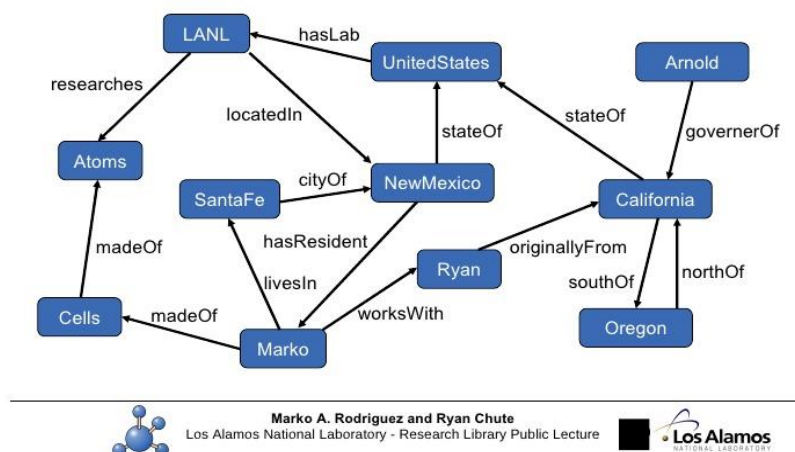
Tim Berners-Lee had already implemented the concept of directly editing web documents in 1990 in World Wide Web, the first web browser, but later ported versions removed this collaborative ability. An early version of NCSA Mosaic in 1993 also included a collaborative annotation capability, though it was quickly removed. Web Distributed Authoring and Versioning, WebDAV, was then reintroduced as an extension.

A different approach to distributed authoring consists in first gathering many annotations from a wide public, and then integrate them all in order to produce a further version of a document. This approach was pioneered by Stet, the system put in place to gather comments on drafts of version 3 of the GNU General Public License. This system arose after a specific requirement, which it served egregiously, but was not so easily configurable as to be convenient for annotating any other document on the web. The co-ment system uses annotation interface concepts similar to Stet's, but it is based on an entirely new implementation, using Django/Python on the server side and various AJAX libraries such as JQuery on the client side. Brat annotation tool is also a web based annotation tool.

2. Semantic Network

Semantic networks are knowledge representation schemes involving nodes and links (arcs or arrows) between nodes. The nodes represent objects or concepts and the links represent relations between nodes. The links are directed and labeled; thus, a semantic network is a directed graph. In print, the nodes are usually represented by circles or boxes and the links are drawn as arrows between the circles as in Figure 1. This represents the simplest form of a semantic network, a collection of undifferentiated objects and arrows. The structure of the network defines its meaning. The meanings are merely which node has a pointer to which another node. The network defines a set of binary relations on a set of nodes.

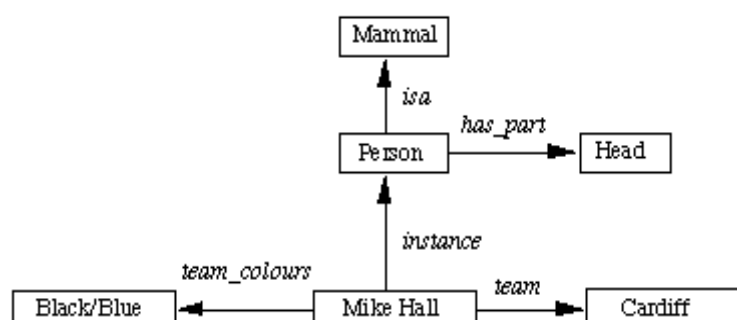
Example semantic network.



2.1. Basics of semantic networks

A semantic network is used when one has knowledge that is best understood as a set of concepts that are related to one another. What is common to all semantic networks is a declarative graphic representation that can be used to represent knowledge and support automated systems for reasoning about the knowledge. In semantic network information is likely to be stored in terms of concepts or categories which are linked together in terms of their meaning.

The physical attributes of a person can be represented as



These values can also be represented in logic as: `isa(person, mammal)`, `instance(Mike-Hall, person)` `team(Mike-Hall, Cardiff)`

2.2. Types of Semantic Network

2.2.1. Definitional networks emphasize the subtype or is-a relation between a concept type and a newly defined subtype. The resulting network, also called a generalization or subsumption hierarchy, supports the rule of inheritance for copying properties defined for a supertype to all of its subtypes.

2.2.2. Assertional networks are designed to assert propositions. Unlike definitional networks, the information in an assertional network is assumed to be contingently true, unless it is explicitly marked with a modal operator. Some assertional networks have been proposed as models of the conceptual structures underlying natural language semantics.

2.2.3. Implicational networks use implication as the primary relation for connecting nodes. They may be used to represent patterns of beliefs, causality, or inferences.

2.2.4. Executable networks include some mechanism, such as marker passing or attached procedures, which can perform inferences, pass messages, or search for patterns and associations.

2.2.5. Learning networks build or extend their representations by acquiring knowledge from examples. The new knowledge may change the old network by adding and deleting nodes and arcs or by modifying numerical values, called weights, associated with the nodes and arcs.

2.2.6. Hybrid networks combine two or more of the previous techniques, either in a single network or in separate, but closely interacting networks.

2.3. Limitations of Semantic Networks

Semantic networks as a representation of knowledge have been in use in artificial intelligence (AI) research in a number of different areas. The other major area in which the use of semantic networks is prevalent, is in models based on linguistics.

- There are many formalisms under the name semantic networks with different expressive capabilities but always with a formal reasoning model
- Different levels of abstraction are mixed in the representation
 - Concepts/instances/values
 - Relations/properties
- A more structured formalism is necessary
- A formal semantic model for reasoning is necessary

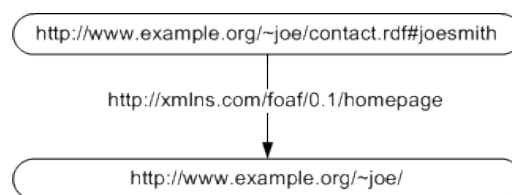
3. Resource Description Framework (RDF):

Resource Description Framework (RDF) is a framework for representing information about resources in a graph form. Since it was primarily intended for representing metadata about WWW resources, it is built around resources with URI.

3.1. RDF Triples:

An RDF triple contains three components: the subject, which is an RDF URI reference or a blank node, the predicate, which is an RDF URI reference, the object, which is an RDF URI reference, a literal or a blank node.

Information is represented by triples *subject-predicate-object* in RDF. An example of a triple is shown in the figure below. It says that "Joe Smith has homepage <http://www.example.org/~joe/>". All elements of this triple are resources defined by URI. The first resource <http://www.example.org/~joe/contact.rdf#joesmith> (*subject*) is intended to identify Joe Smith. Note that it precisely defines how to get to a RDF document as well as how to get the joe smith RDF node in it. The second resource <http://xmlns.com/foaf/0.1/homepage> (*predicate*) is the predicate homepage from a FOAF (Friend-of-a-friend) vocabulary. The last resource (*object*) is Joe's homepage <http://www.example.org/~joe/>.



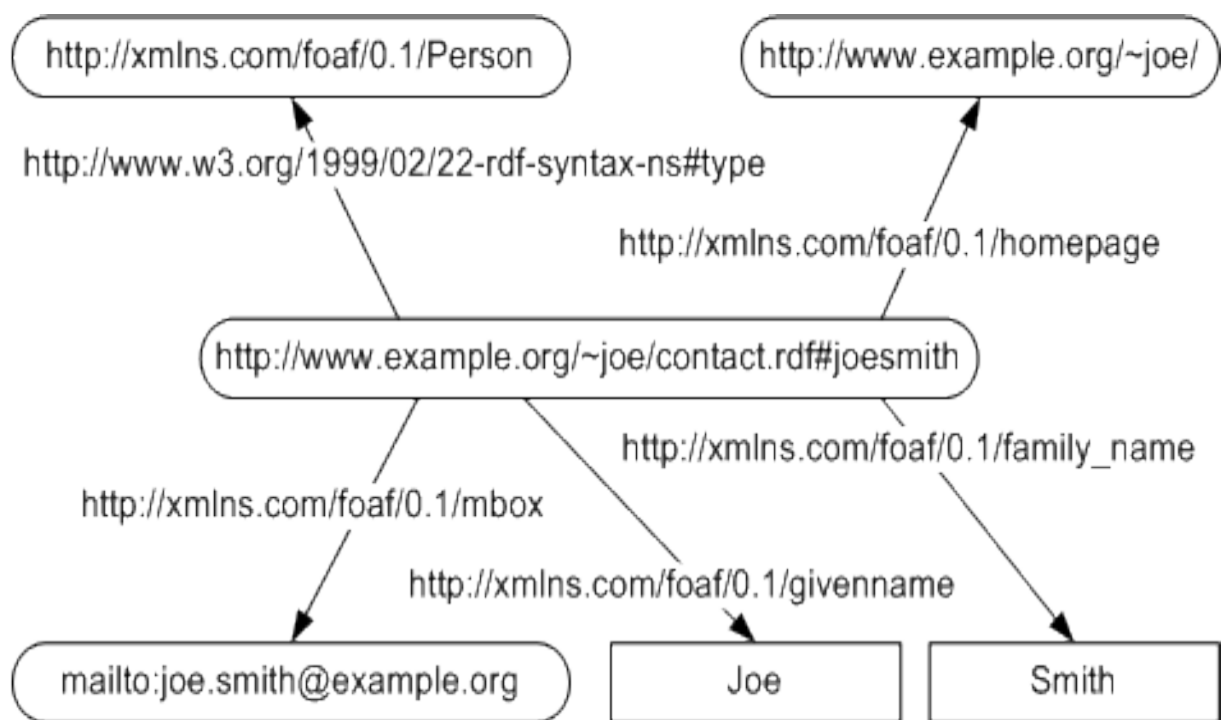
RDF triple (in graph representation) describing Joe Smith - "Joe has homepage identified by URI <http://www.example.org/~joe/>"

3.2. RDF Syntax:

A normative syntax for serializing RDF is RDF/XML. The RDF graph from the figure below is written in RDF/XML as follows. Note that it uses XML namespaces with prefixes defined in the beginning of the XML document.

Code:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns="http://www.example.org/~joe/contact.rdf#">
  <foaf:Person rdf:about="http://www.example.org/~joe/contact.rdf#joesmith">
    <foaf:mbox rdf:resource="mailto:joe.smith@example.org"/>
    <foaf:homepage rdf:resource="http://www.example.org/~joe/">
    <foaf:family_name>Smith</foaf:family_name>
    <foaf:givenname>Joe</foaf:givenname>
  </foaf:Person>
</rdf:RDF>
```



RDF graph describing Joe Smith

4. Stanford CoreNLP

Stanford CoreNLP provides a set of natural language analysis tools. It can give the base forms of words, their parts of speech, whether they are names of companies, people, etc., normalize dates, times, and numeric quantities, mark up the structure of sentences in terms of phrases and word dependencies, indicate which noun phrases refer to the same entities, indicate sentiment, extract particular or open-class relations between mentions, etc.

Stanford CoreNLP's goal is to make it very easy to apply a bunch of linguistic analysis tools to a piece of text. A tool pipeline can be run on a piece of plain text with just two lines of code. CoreNLP is designed to be highly flexible and extensible. With a single option, you can change which tools should be enabled and which should be disabled. Stanford CoreNLP integrates many of Stanford's NLP tools, including the part-of-speech (POS) tagger, the named entity recognizer (NER), the parser, the coreference resolution system, sentiment analysis, bootstrapped pattern learning, and the open information extraction tools. Moreover, an annotator pipeline can include additional custom or third-party annotators. CoreNLP's analyses provide the foundational building blocks for higher-level and domain-specific text understanding applications.

4.1. Why Stanford CoreNLP?

Stanford CoreNLP's goal is to make it very easy to apply a bunch of linguistic analysis tools to a piece of text. A tool pipeline can be run on a piece of plain text with just two lines of code. CoreNLP is designed to be highly flexible and extensible. With a single option, you can change which tools should be enabled and which should be disabled. Stanford CoreNLP integrates many of Stanford's NLP tools, including the part-of-speech (POS) tagger, the named entity recognizer (NER), the parser, the coreference resolution system, sentiment analysis, bootstrapped pattern learning, and the open information extraction tools. Moreover, an annotator pipeline can include additional custom or third-party annotators. CoreNLP's analyses provide the foundational building blocks for higher-level and domain-specific text understanding applications.

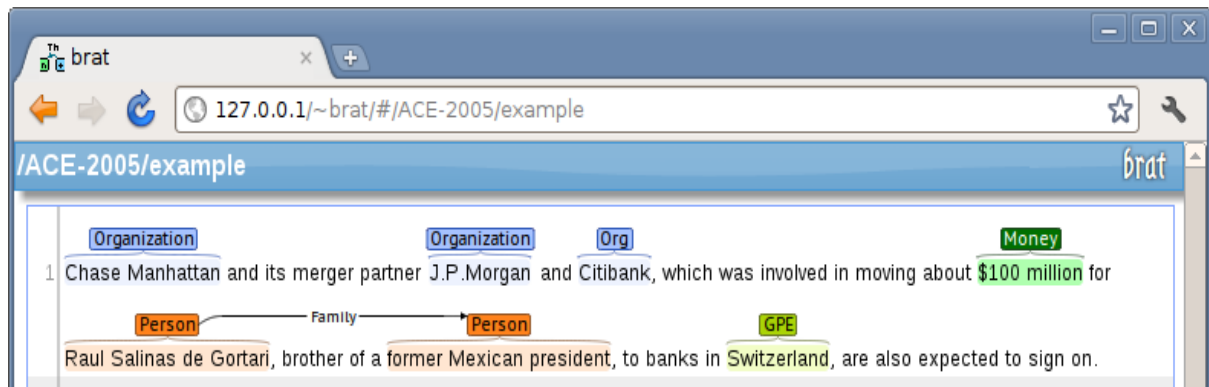
Major features of CoreNLP package are:

- 2.1.1. An integrated toolkit with a good range of grammatical analysis tools.
- 2.1.2. Fast, reliable analysis of arbitrary texts.
- 2.1.3. The overall highest quality text analytics.
- 2.1.4. Support for a number of major (human) languages.
- 2.1.5. Interfaces available for various major modern programming languages.
- 2.1.6. Ability to run as a simple web service

5. Brat Rapid Annotation Tool

Brat is a web-based tool for text annotation; that is, for adding notes to existing text documents. Brat is designed in particular for structured annotation, where the notes are not free form text but have a fixed form that can be automatically processed and interpreted by a computer.

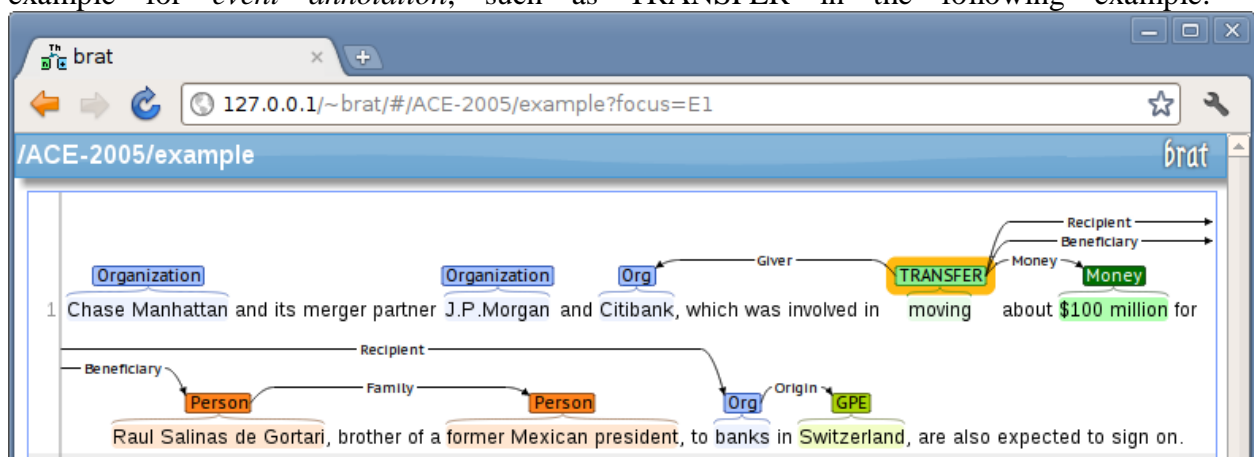
The following screenshot shows a simple example where a sentence has been annotated to identify mentions of some real-world entities (things) and their types, and a relation between two.



This above example illustrates two basic categories of annotation:

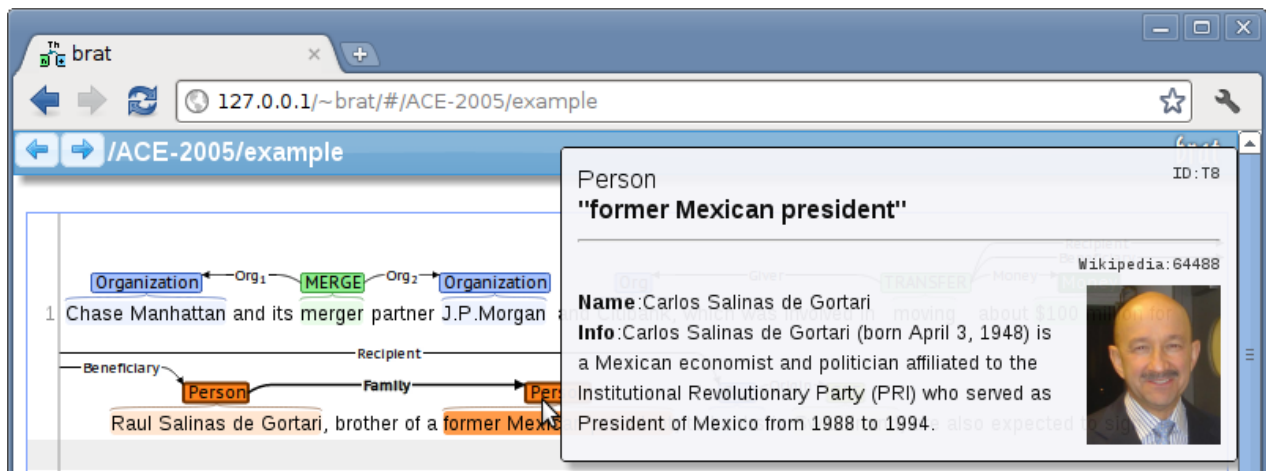
- **text span** annotations, such as those marked with the Organization and Person types in the example
- **Relation** annotations, such as the Family relation in the example.

Brat also supports the annotation of ***n*-ary associations** that can link together any number of other annotations participating in specific roles. This category of annotation can be used for example for *event annotation*, such as TRANSFER in the following example:



The detailed types and properties of other annotations can be further specified through the use of **attributes** that can be set on annotations, for example marking an event as being factual or speculative, or marking an entity mention as referring to a group or an individual.

To allow the unique identification of the real-world entities referred to by specific text expressions, Brat supports also **normalization** annotations (Brat v1.3 (Crunchy Frog) and newer) that associate other annotations with entries in resources such as Wikipedia:



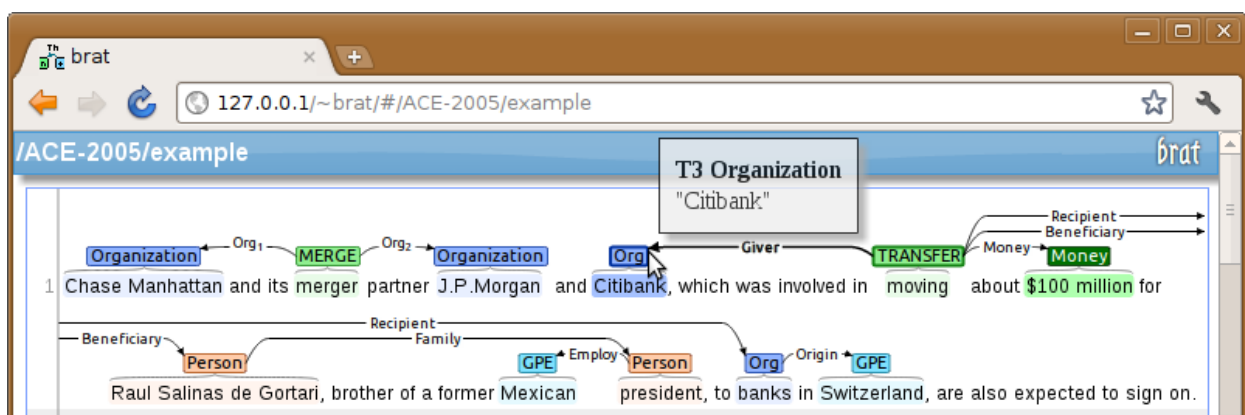
Finally, although not a primary focus of the tool, Brat does also allow free form text "**notes**" to be added to an annotation.

The applied categories of annotations, their types, and the **constraints** regarding their use (for example, that a Family relation must always connect annotations of the Person type) are all fully configurable, allowing Brat to be applied to nearly any text annotation task. Brat also implements a number of features relying on natural language processing techniques to support human annotation efforts.

5.1. Brat and its features

5.1.1. Comprehensive visualization

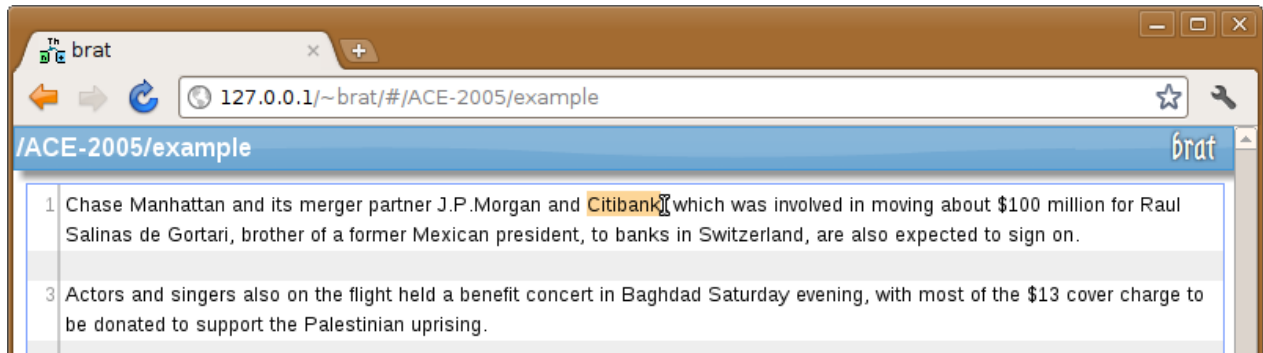
The Brat annotation visualization is based on the concept of "what you see is what you get": all aspects of the underlying annotation are visually represented in an intuitive way.



Annotation visualization

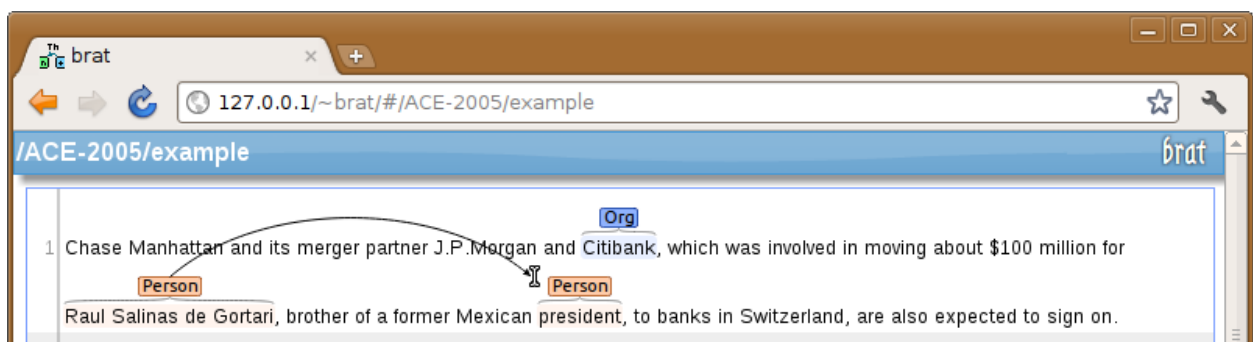
5.1.2. Intuitive editing

Annotation editing is mouse-based and uses intuitive "gestures" familiar from text editors, presentation software, and many other tools. To mark a span of text, simply select it with the mouse by "dragging" or by double-clicking on a word.



Selecting text for annotation

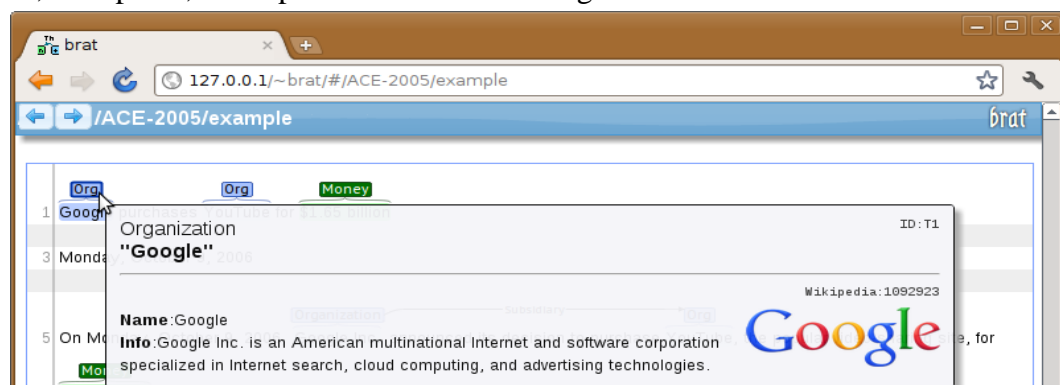
Connecting annotations, for example adding a relation between two annotations, is equally simple: click with the mouse on one annotation and drag a connection to the other.



Connecting annotations

5.1.3. Integration with external resources

As of v1.3 (Crunchy Frog), Brat includes support for normalization and various features for associating annotations with data in external database, lexical and ontological resources such as Freebase, Wikipedia, and Open Biomedical Ontologies.



Showing information from Wikipedia

5.1.4. Zero setup

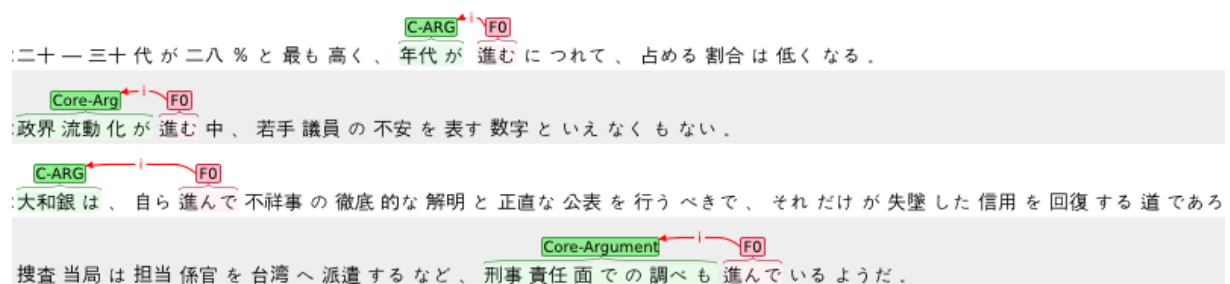
Brat is built entirely on standard web technologies, and it is not necessary to install any local software or browser plugins to use it.

An annotator can "set up" and start using Brat simply by entering the address of the Brat installation into the address bar of a browser.

(Setting up an entirely new Brat server does require some action, but can be done in just five minutes on any system running a web server.)

5.1.5. Annotation of texts in any language

Both the Brat server and client implement full Unicode support, thus supporting nearly 100 different scripts.

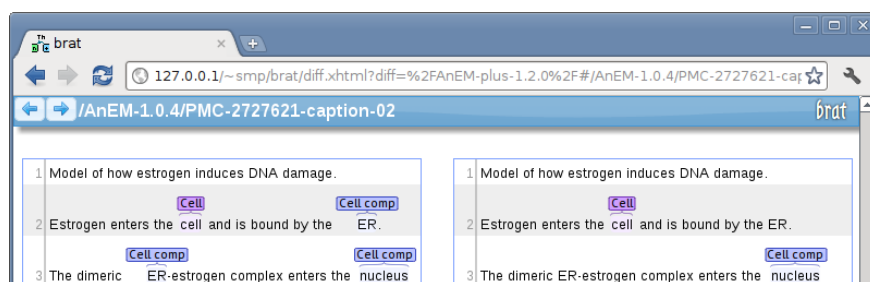


Annotation of text in kanji

Text documents in any language can be converted into [UTF-8 encoded](#) Unicode, which can be annotated in Brat identically to texts in [ASCII](#) format.

5.1.6. Integrated annotation comparison

As of version 1.3, Brat includes a number of features for comparing multiple sets of annotations for the same documents, including automatic comparison for identifying and marking differences and side-by-side visualization.

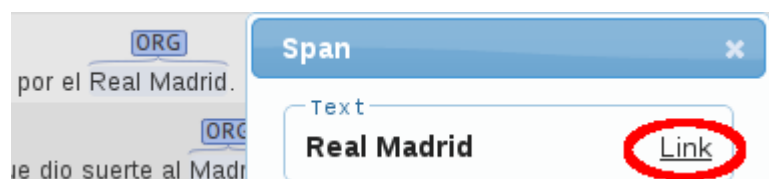
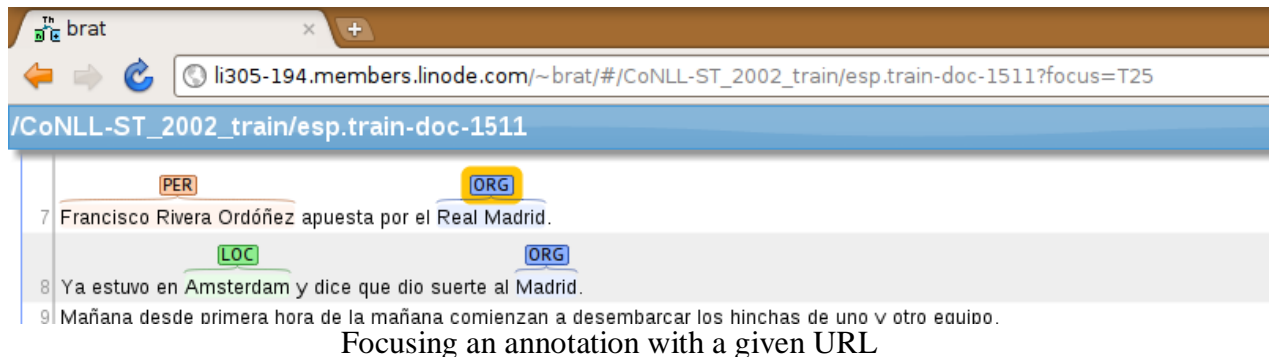


Side-by-side comparison of annotations.

Such comparisons can be used to evaluate automatic systems or the agreement between human annotators, and visualization of differences can help quickly identify common sources of error.

5.1.7. An address for each annotation

Every Brat annotation can be uniquely addressed within the Brat server. Together with the URL of the server, this form of addressing provides a globally unique address for every Brat annotation.

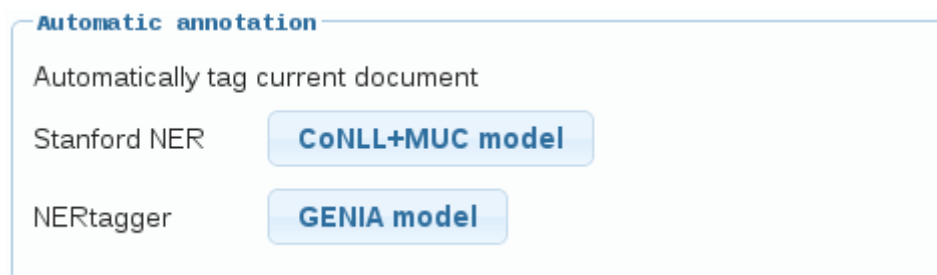


Entering such an address into a browser will not only show the relevant document, but Brat will further highlight and center the specific annotation. These addresses can thus be used in email and online documentation and discussions to simply and unambiguously refer to any annotation in Brat.

The address for each annotation can be easily accessed from a dialog shown by double-clicking on an annotation.

5.1.8. Integration with automatic annotation tools

Brat implements a simple interface for integrating the output of automatic text annotation tools accessible as web services into annotation efforts.

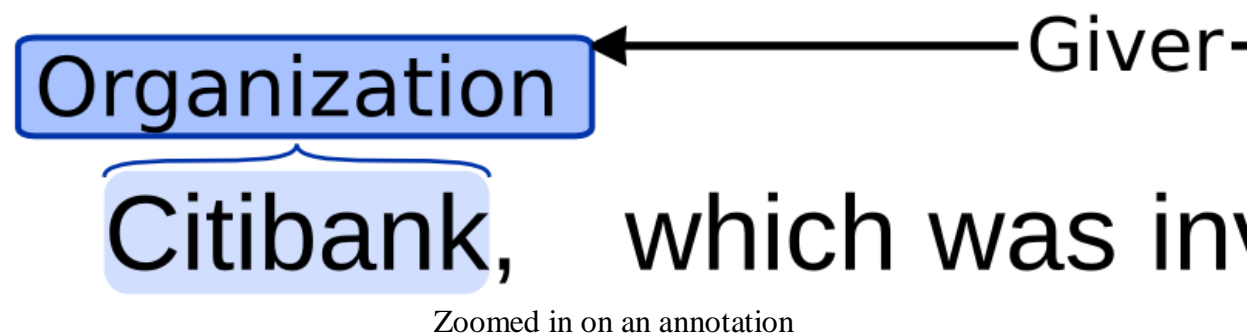


Invoking automatic annotation tools as web services with one click

Brat also features transparent integration with state-of-the-art methods for basic annotation support such as sentence splitting (English and Japanese) and tokenization (Japanese).

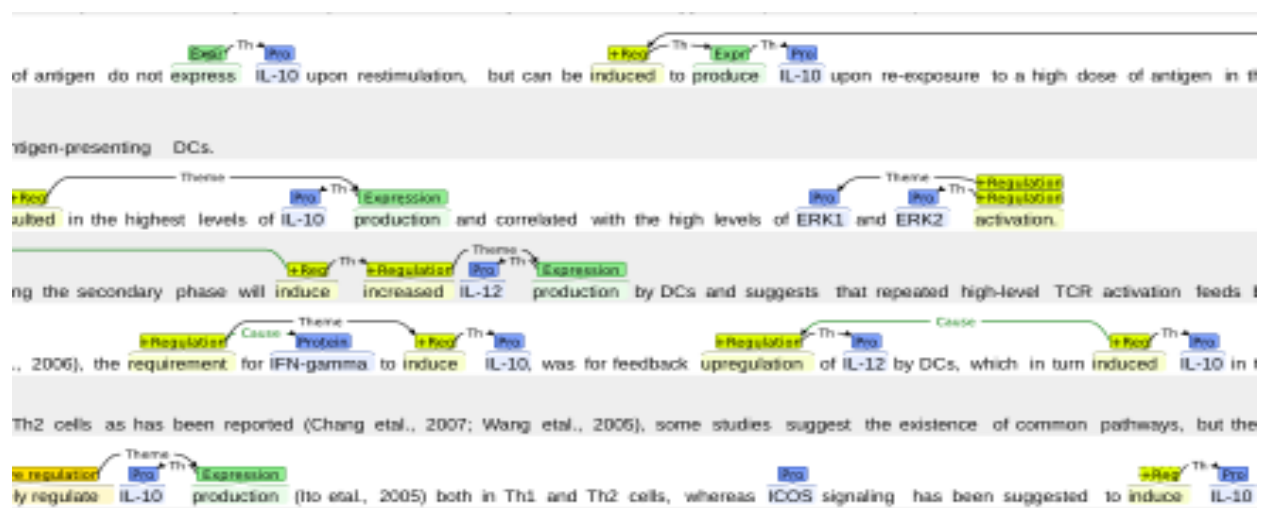
5.1.9. High-quality visualization at any scale

The visualization of Brat is based on Scalable Vector Graphics (SVG), which can be rendered in arbitrary detail and precision.



Brat annotation visualizations are thus inherently print quality and can be used as figures in publications to illustrate annotations.

SVG allows the built-in zoom functions of the browser to be used for closeups or for a high-level view of the annotations of a document.



Zoomed out for an overview

5.1.10. Easy export in multiple formats

Annotations created in Brat can be exported with a few clicks from the interface in a simple standoff format that can be easily analyzed, processed, and converted into other formats.

Visualizations can be similarly being exported in their native SVG format, rendered as a bitmap (PNG format), or converted into other vector formats for embedding into documents (PDF or EPS).

5.1.11. Always saved, always up to date

Brat eliminates any risk of losing annotation efforts to tool crashes, forgetting to save work, or even complete failure of an annotator computer by transparently communicating all edit operations by annotators to the Brat server as they are done. Similarly, in maintaining a single

authoritative version of data shared by all annotators working on a project, the Brat server removes the possibility of conflicting versions of annotations arising or out-of-date data being used as well as the need for annotators to use a separate version control system for coordinating their work.

5.1.12. Real-time collaboration

The Brat client-server architecture and design allow multiple annotators to work simultaneously on the same collection of document, or even on the same document, seeing each other's edits nearly as they are made (some delay is inherent in the communication). All edit operations are coordinated by the server to assure that the annotation remains coherent even if multiple users attempt to simultaneously modify a single annotation.

5.1.13. Detailed annotation process measurement

Brat can optionally be configured to record the precise time that an annotator opens a document, each edit action, and even the time spent selecting the type to assign to an annotation after selecting where to place it.

5.1.14. Rich set of annotation primitives

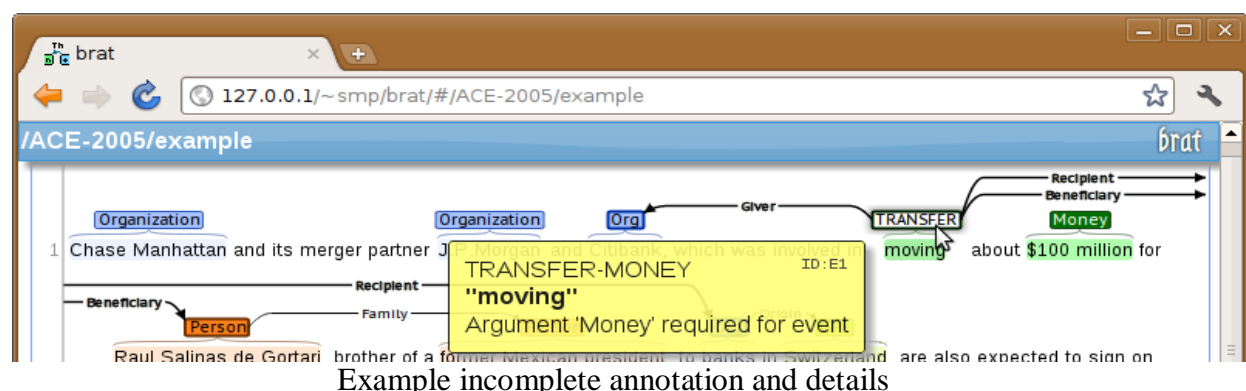
Brat provides a rich set of basic categories of annotation: marking for text spans (for e.g. entity annotation), binary relations, equivalence classes, n -ary associations (for e.g. event annotation) and attributes can be applied together in any combination for defining specific annotation tasks. Some of the many annotation tasks to which Brat can be applied are presented on the examples page.

5.1.15. Fully configurable

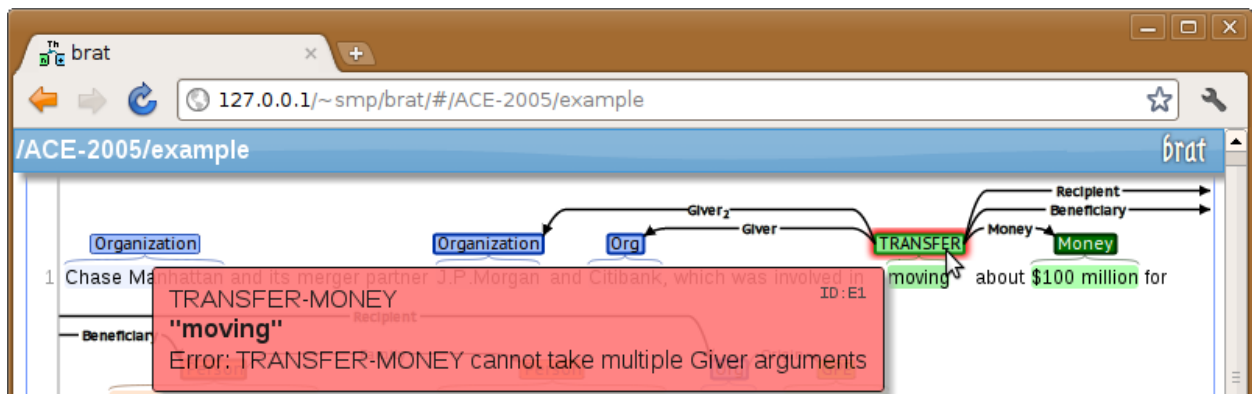
All aspects of annotation are configured using a simple declarative configuration language. Each document collection has its own configuration, allowing a single Brat server to host many projects with different annotation targets. Additionally, most aspects of the visualization such as font, annotation "box" and "arc" colors as well as arrow head and arc drawing style can be controlled in detail using well-documented and widely known HTML/CSS style specifications. For more details on how to configure Brat, see the *CONFIGURATION* page

5.1.16. Always validated

Brat incorporates annotation validation capable of checking all the constraints that can be defined in its expressive configuration. The validation of annotations created in Brat is not isolated into a separate process but closely integrated into the annotation process: the validity of annotations is checked after every edit option, giving annotators immediate feedback with simple visual hints.



Annotations that lack some mandatory part are not given a colored fill and given a gray highlight. Placing the mouse over such an annotation provides the details of the issues detected by the annotation verifier.

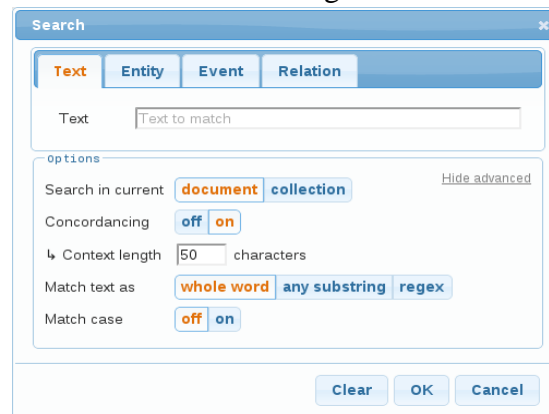


Example annotation with error and details

Annotations that have extra or erroneous parts are given a red "halo" that indicates a problem with the annotation. Again, placing the mouse over such an annotation details the detected issues.

5.1.17. Search

Brat implements a full set of functions for searching documents or document collections for annotation of any type with a detailed set of configurable constraints.



5.1.18. Concordance

Brat supports basic keyword-in-context (KWIC) style concordance of search results.

Document	Annotation	Left context	Text	Right context
esp.train-doc-216	2654-2660	scuelas de Europa. Añadió que	España	tuvo un problema parecido al
esp.train-doc-600	1416-1422	os, ya al menos diez casos en	España	, "sienten inclinación por reb
esp.train-doc-600	3493-3499	la única entidad deportiva de	España	que pasó de sociedad anónima
esp.train-doc-46	2129-2135	e Barcelona, en el noreste de	España	, había cubierto entre otros l
esp.train-doc-46	4559-4565	o del equipo de la capital de	España	en la final de la liga de cam
esp.train-doc-423	680-686	entrenador más carismático de	España	y es el estandarte. Todo el m
esp.train-doc-423	2204-2210	na de las mejores canteras de	España	. Hay dos o tres jugadores del
esp.train-doc-896	44-50	may (EFE). - Los gobiernos de	España	y Francia se comprometieron h
esp.train-doc-896	1331-1337	ilidades de cooperación entre	España	y Francia, para conseguir no
esp.train-doc-896	1569-1575	será "cada día más fuerte" y "	España	siempre tendrá todo nuestro a
esp.train-doc-896	2108-2114	sidente francés reconoció que	España	ha presentado solicitudes par

Search results with concordance

5.2. Installation of Brat tool on Linux environment

To install Brat Rapid Annotation tool locally on your Linux operating system, few steps are to be followed for smooth and error free installation of the tool. The steps are as follows.

5.2.1. Pre-Setup

Prerequisites for Brat Rapid Annotation Tool installation:

(The commands are processed in 'Home' directory of the Linux distro)

1. *apt-get update*
2. *apt-get install apache2-bin*
3. *apt-get install apache2*

Above Installs apache group and bin to your Linux repositories apt-get update (to update Linux repositories)

5.2.2. Installing BRAT Locally

1. Extract the downloaded file to Brat directory click to get into the directory.
2. Create data work inside the directory
mkdir -p data work
3. Check your system's Apache group (this is necessary if you have multi user Linux system)

./apache-group.sh

- 3.1. You get the name of your Apache group that was created at the point of installation.

- 3.2. Inside Brat directory accessible to Apache group by modifying the access points for your Linux distros.

Modification is done something like this:

chmod -R \${Name} g+rx data work

If the system minutes multi user system then you can use another command which is:

chmod 777 data work

5.2.3. Setting up dependencies

Check the python version of your system if it is not greater than 2.5 then you need to install the latest python libraries. To check python version just type on terminal:

python --version

5.2.4. Installing UltraJSON

1. *(cd server/lib/ && unzip ujson-1.18.zip && cd ujson-1.18 && python setup.py build && mv build/lib.*../ujson)*

If the installation of UltraJSON fails, then install SimpleJSON all included in the directory of server, a simple python installation is what is required.

Install it after extracting it all in the Brat directory, directly.

5.2.5. Final Step

After all the above steps are successfully over go to Brat directory and start installation process through the command

```
./install.sh
```

Installation should ask for username, password and email

After installation complete you should have an error less page of terminal commands that load up after installation.

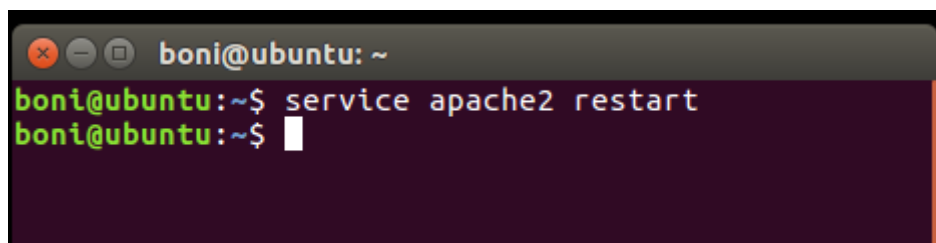
Few steps are necessary to get the Brat Tool running smoothly

1. Perform an *apt-get update* in your /home to get all repos updated.
2. Perform a *service apache2 restart* command.
3. Now start standalone python program through *python standalone.py* or *./standalone.py*

5.3. Configuring project on Brat-tool

To start the configuration of Brat annotation tool we have to do the following steps:

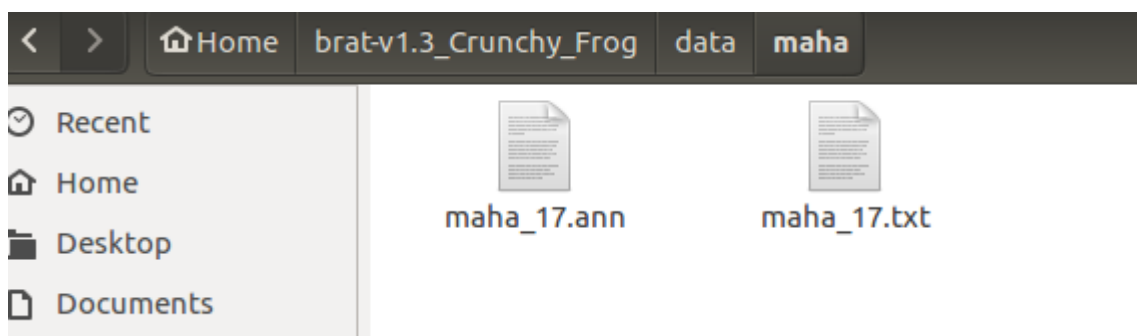
5.3.1. How to start the server



```
boni@ubuntu: ~  
boni@ubuntu:~$ service apache2 restart  
boni@ubuntu:~$
```

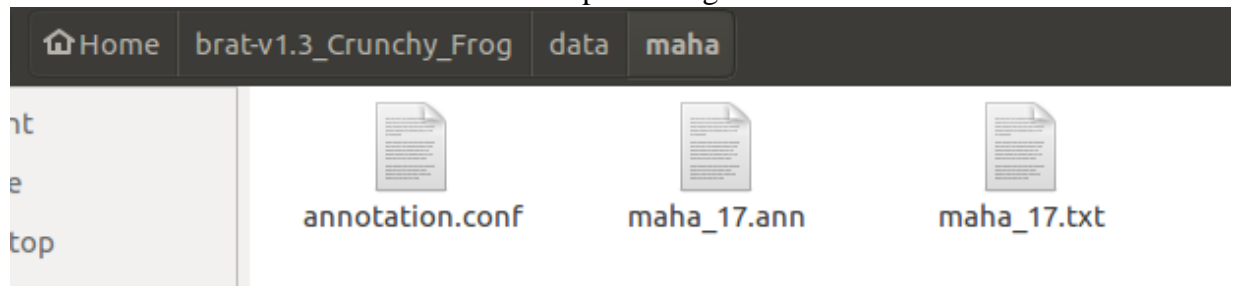
5.3.2. Creating your own directory

- Go to the **Brat-v1.3_Crunchy_Frog** folder in your home directory of linux. In the **data** directory create your own project folder.
- Create a new **.txt** file in that folder which will contain the sample text required for annotation.



5.3.3. Creating .ann file

- In the data folder create a new **.ann** file which contains the same name as the **.txt** file, also create annotation.conf file for future processing.



5.3.4. Programming in annotation.conf

The annotation.conf file consists of 4 major parts, they are:

1. entities
2. events
3. relations
4. attributes

5.3.5. [Entities]section

```
[entities]
Character
Organization
Non-Living
Community
GPE
```

The above syntax specifying a hierarchy of entity organization which consists simply of TAB characters at the beginning of each line.

To include your project specific entities keep them under the tag of [entities] within box brackets and they will show up in your project.

Furthermore if you have to segregate few entities like under Human you want to segregate it as Male, Female, Transgender, you can simply do this like the following:

```
[entities]
!Human
    Male
    Female
    Transgender
```

Simply by adding “!” we can tell the tool not to annotate under that particular word but indent gives the user insight to know it well which to tag.

5.3.6. [Relations]section

```
[relations]
Located                Arg1:Character, Arg2:GPE
Geographical_part      Arg1:GPE,   Arg2:GPE
Family                 Arg1:Character, Arg2:Character
Son_of                 Arg1:Character, Arg2:Character
Daughter_of            Arg1:Character, Arg2:Character
Father_of              Arg1:Character, Arg2:Character
Mother_of              Arg1:Character, Arg2:Character
Brother_of             Arg1:Character, Arg2:Character
Sister_of              Arg1:Character, Arg2:Character
Siblings_of            Arg1:Character, Arg2:Character
Uncle_of               Arg1:Character, Arg2:Character
Aunt_of                Arg1:Character, Arg2:Character
Wife_of                Arg1:Character, Arg2:Character
Ruler_of               Arg1:Character, Arg2:Community
King_of                Arg1:Character, Arg2:GPE
Member_of              Arg1:Character, Arg2:Community
Disciple_of            Arg1:Character, Arg2:Character
Enemy                  Arg1:Character, Arg2:Character
Enemy_of_States        Arg1:Character, Arg2:GPE
Employment             Arg1:Character, Arg2:GPE
Ownership              Arg1:Character, Arg2:Organization
Origin                 Arg1:Organization, Arg2:GPE
```

This can be defined in the configuration by listing all the possible types separated by the pipe character "|".

As for entities, a hierarchy of relation types can be defined simply by placing TAB characters at the beginning of each line. The Arg1 and Arg2 must be the name of the entities defined in the annotation file.

for example, let us take a snippet and understand the syntax:

```
[relations]
    Location      Arg1:Character, Arg2:GPE  (Geo-Political Entity)
```

Explanation:

name_of_the_relation Argument1:(is a)Character_type, Argument2: (is a) Geopolitical Entity

5.3.7. [Events]section

[events]

!Life

```
Be-born    Person-Arg:Character, Place-Arg?:GPE,  
Be-B0rnExample    Person-Arg:Character, Place-Arg?:Organization,  
Marry      Person-Arg{2}:Person, Place-Arg?:GPE  
Divorce    Person-Arg{2}:Person, Place-Arg?:GPE  
Die        Person-Arg:Person, Agent-Arg?:<POG>, Place-Arg?:GPE
```

The above example shows the events section where each line defined one event type with the event arguments specified with a ROLE: TYPE syntax and separated by commas. The ROLE: TYPE must be different from each other.

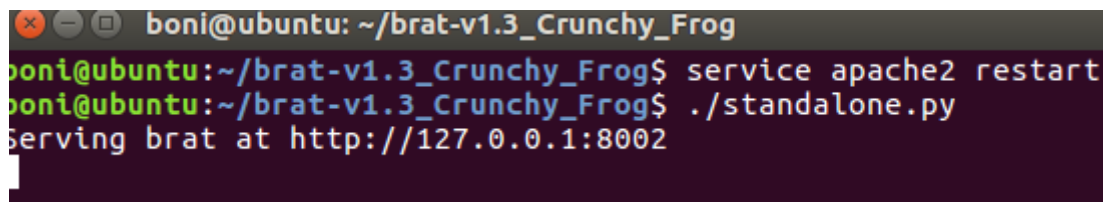
5.3.8. [Attributes]section

[attributes]

```
Individual    Arg:<ENTITY>  
Mention       Arg:<ENTITY>, Value:Name|Nominal|Other  
  
Negation      Arg:<EVENT>  
Confidence    Arg:<EVENT>, Value:High|Neutral|Low|
```

The values a multi-valued attribute can take are defined using the syntax Value: VAL1|VAL2|VAL3 [...], where "Value" is a literal string and VAL1, VAL2 etc. are the possible values.

5.4. Accessing your project in Brat tool

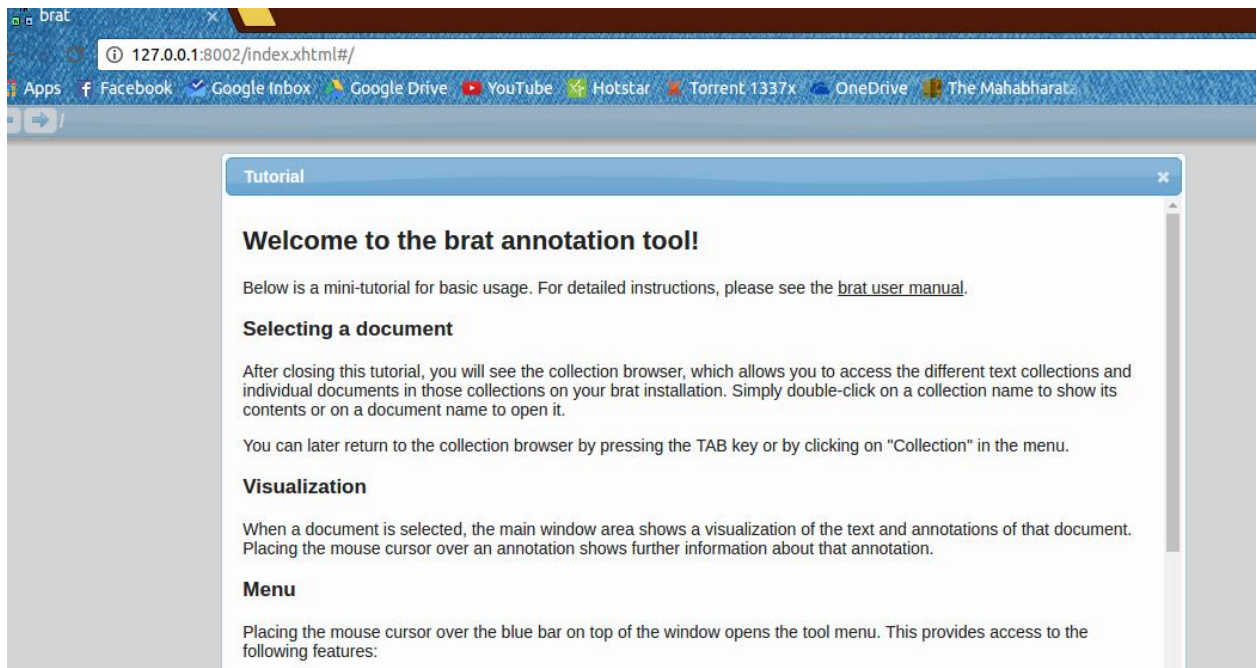


```
boni@ubuntu: ~/brat-v1.3_Crunchy_Frog  
boni@ubuntu:~/brat-v1.3_Crunchy_Frog$ service apache2 restart  
boni@ubuntu:~/brat-v1.3_Crunchy_Frog$ ./standalone.py  
Serving brat at http://127.0.0.1:8002
```

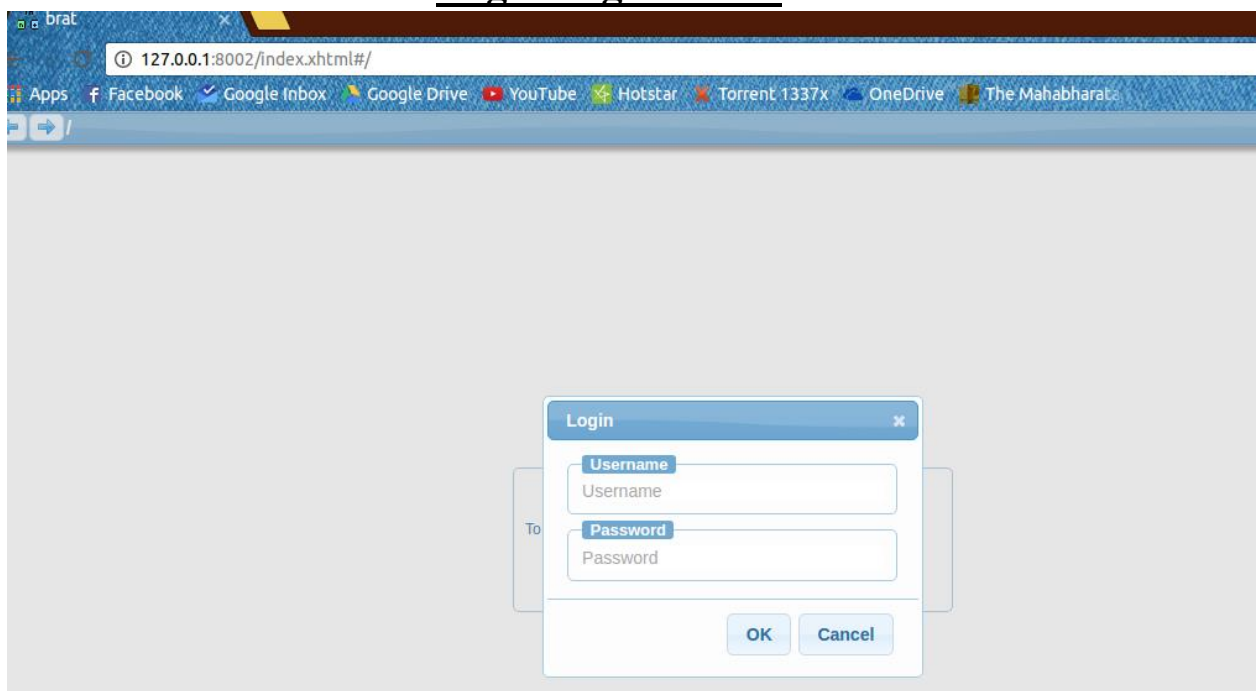
the command “./standalone.py” starts the server on your local server.

5.5. Screenshots of Brat tool

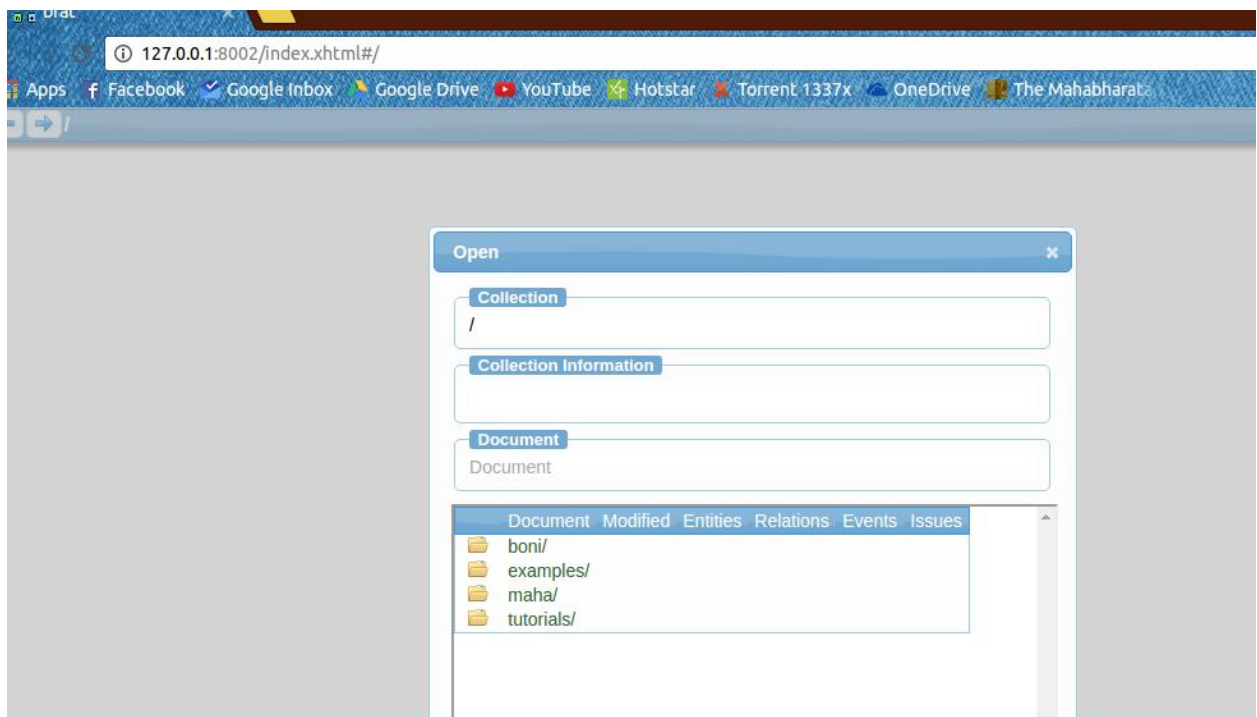
Welcome windows of Brat



Login Page of Brat



5.5.1. Selecting the document in the browser:



When starting, Brat will show the collection browser. To open a document, simply double-click on one of the documents (📁) in the document selector.

6. Graphviz

Graphviz is open source graph visualization software. Graph visualization is a way of representing structural information as diagrams of abstract graphs and networks.

It has important applications in networking, bioinformatics, software engineering, database and web design, machine learning, and in visual interfaces for other technical domains.

The Graphviz layout programs take descriptions of graphs in a simple text language, and make diagrams in useful formats, such as images and SVG for web pages; PDF or Postscript for inclusion in other documents; or display in an interactive graph browser.

Graphviz has many useful features for concrete diagrams, such as options for colors, fonts, tabular node layouts, line styles, hyperlinks, and custom shapes.

6.1. Types of files in Graphviz

dot - "hierarchical" or layered drawings of directed graphs. This is the default tool to use if edges have directionality.

neato - "spring model" layouts. This is the default tool to use if the graph is not too large (about 100 nodes) and you don't know anything else about it. Neato attempts to minimize a global energy function, which is equivalent to statistical multidimensional scaling.

fdp - "spring model" layouts similar to those of neato, but does this by reducing forces rather than working with energy.

sfdp - multiscale version of fdp for the layout of large graphs.

twopi - radial layouts, after Graham Wills 97. Nodes are placed on concentric circles depending their distance from a given root node.

circo - circular layout, after Six and Tollis 99, Kaufmann and Wiese 02. This is suitable for certain diagrams of multiple cyclic structures, such as certain telecommunications networks.

7. Explanation of the Developed Classes

7.1. Class : TripleExtractionBrat

The class TripleExtractionBrat is written in Java. We have used ArrayList, Iterator, List, StringTokenizer classes from util package and io package for the creation of this code.

TripleExtractionBrat extracts the triples (subject, object and predicate) from the ANN file we received by annotating the given document in Brat annotation tool. The triples extracted are stored in a file in table format to be further analyzed and for the creation of the graph.

TripleExtractionBrat class takes input the location of the ANN file and then finds for relations in each line. When a relation is found then the subject, object and predicate are extracted and stored in separate lists. Then the triples are written in a file extracted for each sentence.

Code Snippet

```
TripleExtractionBrat.java
25 fr = new FileReader("C:\\Users\\dupsm\\Documents\\Experiment\\mana_17.ann");
26 br = new BufferedReader(fr);
27 PrintWriter out = new PrintWriter ( new BufferedWriter ( new FileWriter ( "C:\\Users\\dupsm\\Documents\\Experiment\\Output.txt",
28 String sCurrentLine="";
29 String allLines= "";
30 while((sCurrentLine=br.readLine())!= null){
31     allLines+=sCurrentLine+"\n";
32 }
33 out.format("%15s%15s%15s", "Subject", "Predicate", "Object");
34 out.println();
35 t = new StringTokenizer(allLines);
36 String word=" ";
37 while(t.hasMoreTokens())
38 {
39     word=t.nextToken();
40     myList.add(word);
41 }
42 //Converting tokens to elements of list
43 Iterator<String> iterate=myList.iterator();
44 int i=0;
45
46 while(iterate.hasNext()){
47
48     String wording=iterate.next().toString();
49     /*Checking for relation and finding R(Relation) in the form of ann and extracting the sentence related to
50     * R which is the relation annotation shortcut in brat tool.
51     */
52     if(wording.length()==2 && wording.charAt(i)=='R' && (int)wording.charAt(i+1)>(int)0)
53     {
54         Display=RelationExtraction(myList.indexOf(wording));
55     }
56 }
```

Input

mana_17.ann - WordPad

File Home View

Clipboard Font Paragraph Insert Editing

Find Replace Select all

Picture drawing Paint Date and time Insert object

T1 Character 507 513 Arjuna
T3 GPE 1444 1455 Hastinapura
T4 Alias 1502 1515 Shakraprastha
E2 Alias:T4 Place:T3
T6 Character 1482 1487 Vajra
T5 Alias 1463 1480 the Yadava prince
E3 Alias:T5 Named:T6
T7 Character 1391 1396 Vajra
T8 Alias 1364 1389 The survivor of the Yadus
E4 Alias:T8 Named:T7
T10 Character 1962 1968 Narada
T11 Character 1617 1643 king Yudhishtira the just
T9 Character 1935 1956 the Island-born Vyasa
T12 Character 1974 2016 Markandeya possessed of wealth of penances
T13 Character 2022 2053 Vajnavalkya of Bharadwaja race
T14 Ee-born 371 377 Having
E5 Ee-born:T14
T15 Organization 413 422 slaughter
T16 Ee-BOmExample 388 359 particulars
E6 Ee-BOmExample:T16 Place-Arg:T15
T17 Character 28 36 Narayana
T18 Character 45 49 Nara
R1 Family Arg1:T17 Arg2:T18
T19 Character 95 104 Sarasvati
R2 Family Arg1:T17 Arg2:T19
R3 Enemy Arg1:T18 Arg2:T19
R4 Family Arg1:T19 Arg2:T17
T2 Character 772 777 Runti
R5 Family Arg1:T1 Arg2:T2

Output



Subject	Predicate	Object
Narayana	Family	Nara
Narayana	Family	Sarasvati
Nara	Enemy	Sarasvati
Sarasvati	Family	Narayana
Arjuna	Family	Kunti
Nara	Family	Arjuna
Nara	IS	Nara
Narayana	IS	Narayana

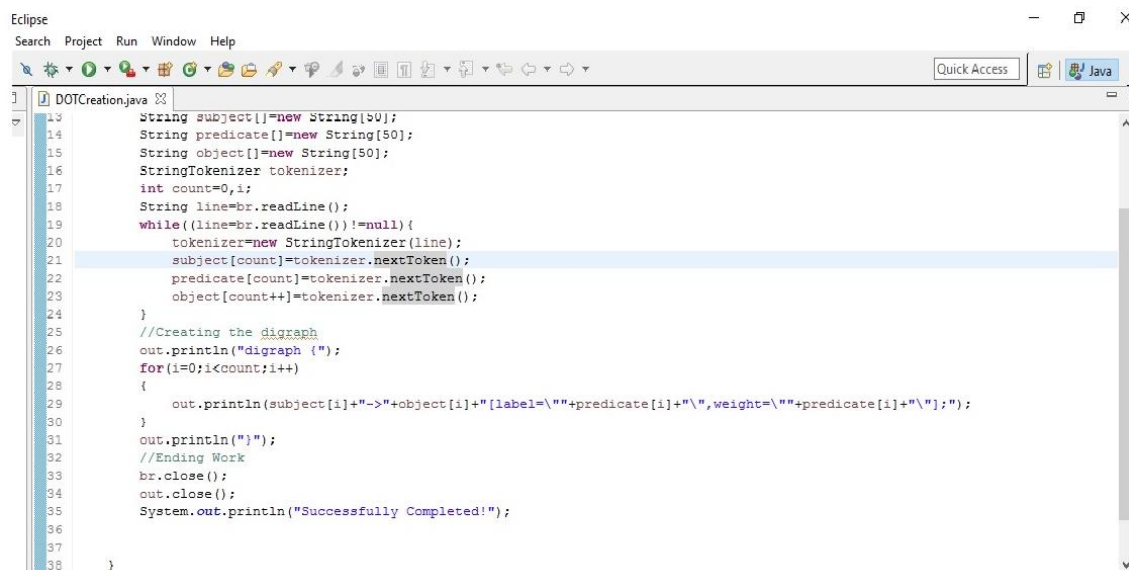
7.2. Class : DOTCreation

The class DOTCreation is written in java. We have used BufferedReader, BufferedWriter, FileReader, FileWriter, IOException, PrintWriter classes from io package and StringTokenizer class from util package for the creation of this code.

DOTCreation creates a DOT file which is used in Graphviz to create the graph. It takes input the text file created by TripleExtractionBrat and gives as output the DOT file.

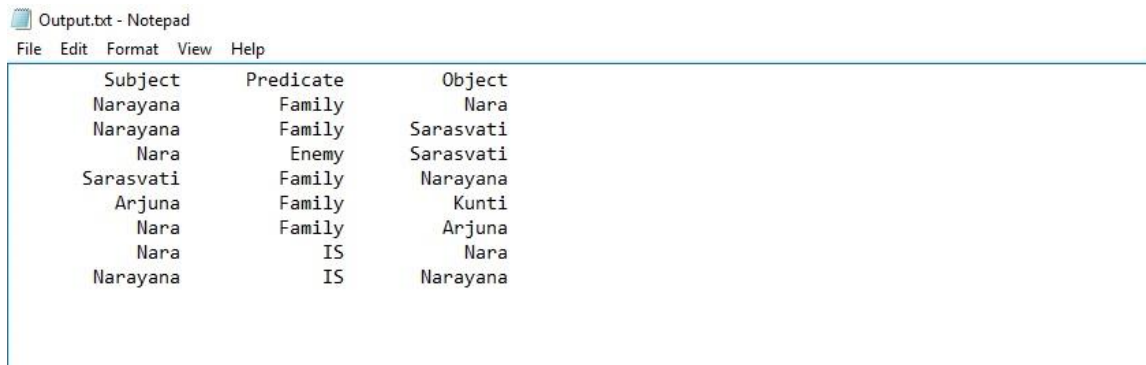
DOTFile takes input the location of the text document and then formats the data in DOT format. The formatted data is then written in the DOT file.

Code Snippet



```
13 String subject[]=new String[50];
14 String predicate[]=new String[50];
15 String object[]=new String[50];
16 StringTokenizer tokenizer;
17 int count=0,i;
18 String line=br.readLine();
19 while((line=br.readLine())!=null){
20     tokenizer=new StringTokenizer(line);
21     subject[count]=tokenizer.nextToken();
22     predicate[count]=tokenizer.nextToken();
23     object[count++]=tokenizer.nextToken();
24 }
25 //Creating the digraph
26 out.println("digraph {");
27 for(i=0;i<count;i++){
28     {
29         out.println(subject[i]+"->"+"object[i]+"[label=\""+predicate[i]+"\",weight=\""+predicate[i]+"\"]");
30     }
31 out.println("}");
32 //Ending Work
33 br.close();
34 out.close();
35 System.out.println("Successfully Completed!");
36
37
38 }
```

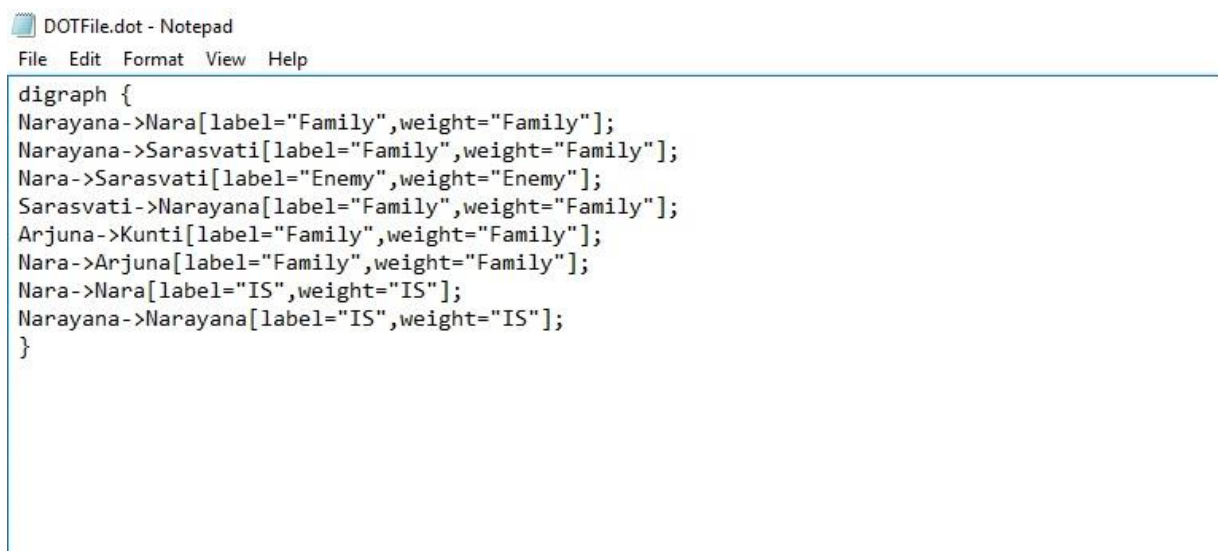
Input



Output.txt - Notepad

Subject	Predicate	Object
Narayana	Family	Nara
Narayana	Family	Sarasvati
Nara	Enemy	Sarasvati
Sarasvati	Family	Narayana
Arjuna	Family	Kunti
Nara	Family	Arjuna
Nara	IS	Nara
Narayana	IS	Narayana

Output



DOTFile.dot - Notepad

```
File Edit Format View Help

digraph {
Narayana->Nara[label="Family",weight="Family"];
Narayana->Sarasvati[label="Family",weight="Family"];
Nara->Sarasvati[label="Enemy",weight="Enemy"];
Sarasvati->Narayana[label="Family",weight="Family"];
Arjuna->Kunti[label="Family",weight="Family"];
Nara->Arjuna[label="Family",weight="Family"];
Nara->Nara[label="IS",weight="IS"];
Narayana->Narayana[label="IS",weight="IS"];
}
```

7.3. Class : PatternMatcher

The class PatternMatcher is written in Java. We have used BufferedReader, FileReader, IOException, StringReader classes from io package, List class from util package, CoreLabel, LexicalizedParser, CoreLabelTokenFactory, PTBTokenizer, Tokenizer, TokenizerFactory, Tree, TregexMatcher, TregexPattern classes from CoreNLP package by Stanford University for the creation of this code.

PatternMatcher is used to create a summary of the given document based on the given pattern and the formed summary is stored in a text document.

PatternMatcher takes input the file path and pattern upon which the task is to be performed. The given pattern is parsed and a parse object is created which is compared to the given text for matches, if a match is found then the sentence is stored in the output file for the summary.

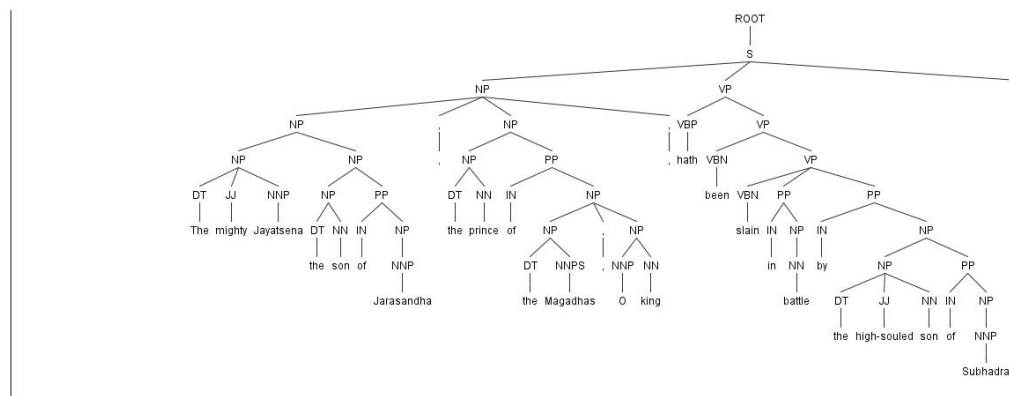
```

1 PatternMatcher.java
29     tree = parser.apply(tokens);
30     return tree;
31 }
32
33 private List<CoreLabel> tokenize(String str) {
34     int an=657583,dp=6877,ax=6568,at=6582,ka=7571;
35     Tokenizer<CoreLabel> tokenizer =
36         tokenizerFactory.getTokenizer(
37             new StringReader(str));
38     return tokenizer.tokenize();
39 }
40 public String PatternMatch(String PATH, String Pattern)throws IOException {
41     //ChooseFile cf = new ChooseFile();
42     int an=657583,dp=6877,ax=6568,at=6582,ka=7571;
43     String sentence;
44     PatternMatcher matcher=new PatternMatcher();
45     Tree tree=null;
46     TregexPattern p=TregexPattern.compile(Pattern);
47     TregexMatcher m=null;
48     // ===== PATH =====
49
50     System.out.println("THIS IS IN PATTERN MATCHER \n\n"+PATH);
51
52     BufferedReader br=new BufferedReader(new FileReader(PATH));
53     //an an=657583,dp=6877,ax=6568,at=6582,ka=7571;
54     while ( (sentence=br.readLine()) !=null) {
55         tree=matcher.parse(sentence);
56         m=p.matcher(tree);
57         while (m.find()) {
58             output+="\n"+m.getMatch().toString();
59         }
60     }
61 }

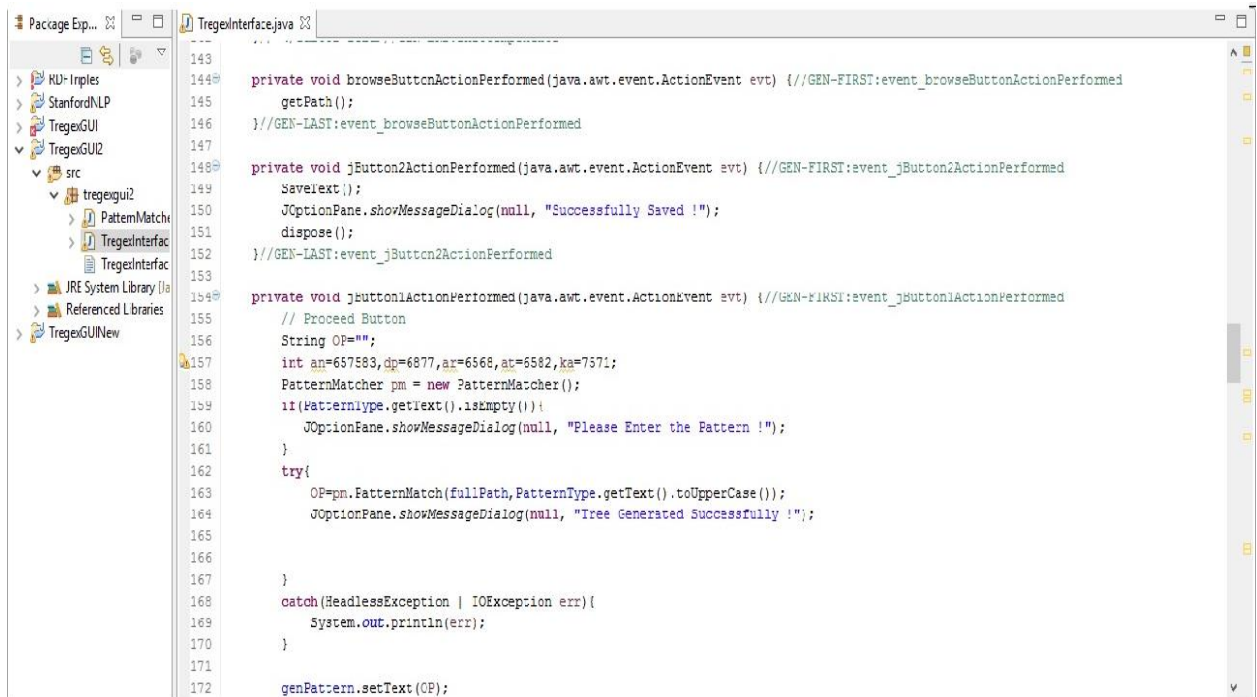
```

The TregexInterface is written in Java. We have used FileNotFoundException, IOException, BufferedReader, FileOutputStream, FileReader, PrintStream classes in io package, HeadlessException in AWT package and JOptionPane in Swing package for the creation of this code.

TregexInterface provides a graphical interface which allows to select the file to be summarized and a textbox to enter the pattern. When the user clicks the proceed button the file and pattern are sent to the PatternMatcher class which then creates the summary and returns it to TregexInterface which gives a preview of the summary in a textbox and also gives the user an option to save the summary in a text file.

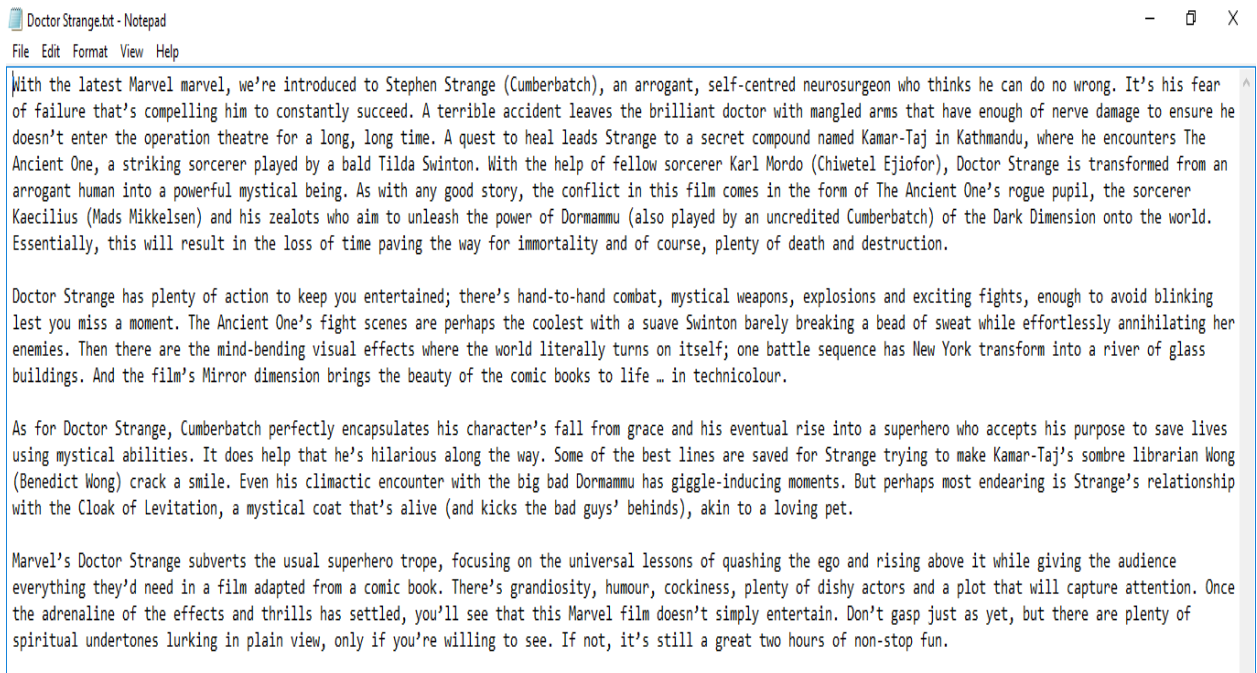


Code Snippet



```
143
144 private void browseButtonActionPerformed(java.awt.event.ActionEvent evt) {GEN-FIRST:event_browseButtonActionPerformed
145     getPath();
146 } //GEN-LAST:event_browseButtonActionPerformed
147
148 private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {GEN-FIRST:event_jButton2ActionPerformed
149     SaveText();
150     JOptionPane.showMessageDialog(null, "Successfully Saved !");
151     dispose();
152 } //GEN-LAST:event_jButton2ActionPerformed
153
154 private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {GEN-FIRST:event_jButton1ActionPerformed
155     // Proceed Button
156     String OP="";
157     int an=657583,dp=6877,ar=6568,at=6582,ka=7371;
158     PatternMatcher pm = new PatternMatcher();
159     if(PatternType.getText().isEmpty()){
160         JOptionPane.showMessageDialog(null, "Please Enter the Pattern !");
161     }
162     try{
163         OP=pm.PatternMatch(fullPath,PatternType.getText().toUpperCase());
164         JOptionPane.showMessageDialog(null, "Tree Generated Successfully !");
165     }
166 }
167
168 catch (HeadlessException | IOException err){
169     System.out.println(err);
170 }
171
172 genPattern.setText(OP);
```

Input



Doctor Strange.txt - Notepad

File Edit Format View Help

With the latest Marvel marvel, we're introduced to Stephen Strange (Cumberbatch), an arrogant, self-centred neurosurgeon who thinks he can do no wrong. It's his fear of failure that's compelling him to constantly succeed. A terrible accident leaves the brilliant doctor with mangled arms that have enough of nerve damage to ensure he doesn't enter the operation theatre for a long, long time. A quest to heal leads Strange to a secret compound named Kamar-Taj in Kathmandu, where he encounters The Ancient One, a striking sorcerer played by a bald Tilda Swinton. With the help of fellow sorcerer Karl Mordo (Chiwetel Ejiofor), Doctor Strange is transformed from an arrogant human into a powerful mystical being. As with any good story, the conflict in this film comes in the form of The Ancient One's rogue pupil, the sorcerer Kaecilius (Mads Mikkelsen) and his zealots who aim to unleash the power of Dormammu (also played by an uncredited Cumberbatch) of the Dark Dimension onto the world. Essentially, this will result in the loss of time paving the way for immortality and of course, plenty of death and destruction.

Doctor Strange has plenty of action to keep you entertained; there's hand-to-hand combat, mystical weapons, explosions and exciting fights, enough to avoid blinking lest you miss a moment. The Ancient One's fight scenes are perhaps the coolest with a suave Swinton barely breaking a bead of sweat while effortlessly annihilating her enemies. Then there are the mind-bending visual effects where the world literally turns on itself; one battle sequence has New York transform into a river of glass buildings. And the film's Mirror dimension brings the beauty of the comic books to life ... in technicolour.

As for Doctor Strange, Cumberbatch perfectly encapsulates his character's fall from grace and his eventual rise into a superhero who accepts his purpose to save lives using mystical abilities. It does help that he's hilarious along the way. Some of the best lines are saved for Strange trying to make Kamar-Taj's sombre librarian Wong (Benedict Wong) crack a smile. Even his climactic encounter with the big bad Dormammu has giggle-inducing moments. But perhaps most endearing is Strange's relationship with the Cloak of Levitation, a mystical coat that's alive (and kicks the bad guys' behinds), akin to a loving pet.

Marvel's Doctor Strange subverts the usual superhero trope, focusing on the universal lessons of quashing the ego and rising above it while giving the audience everything they'd need in a film adapted from a comic book. There's grandiosity, humour, cockiness, plenty of dishy actors and a plot that will capture attention. Once the adrenaline of the effects and thrills has settled, you'll see that this Marvel film doesn't simply entertain. Don't gasp just as yet, but there are plenty of spiritual undertones lurking in plain view, only if you're willing to see. If not, it's still a great two hours of non-stop fun.

Output :

```
(SBAR (WHNP (WDT that)) (S (VP (VBZ 's) (ADJP (JJ compelling)
(SBAR (S (NP (PRP him)) (VP (TO to) (VP (ADVP (RB constantly))
(VB succeed))))))))))
(NP (DT the) (JJ brilliant) (NN doctor))
(NP (DT the) (NN operation) (NN theatre))
(NP (DT The) (JJ Ancient) (CD One))
(NP (NP (DT the) (NN power)) (PP (IN of) (NP (NNP Dormammu)))
(PRN (-LRB- -LRB-) (VP (ADVP (RB also)) (VBN played) (PP (IN by)
(NP (DT an) (JJ uncredited) (NN Cumberbatch)))) (-RRB- -RRB-))
(PP (IN of) (NP (DT the) (JJ Dark) (NN Dimension))))
(NP (DT the) (NN way))
(NP (NP (RB plenty)) (PP (IN of) (NP (NN action))) (S (VP (TO
to) (VP (VB keep) (S (S (NP (PRP you)) (VP (VBD entertained) (S
(: ;) (S (NP (EX there)) (VP (VBZ 's) (NP (NP (JJ hand-to-hand)
(NN combat)) (, ,) (NP (JJ mystical) (NNS weapons)) (, ,) (NP
(NNS explosions)) (CC and) (NP (JJ exciting) (NNS fights)))
(, ,) (S (ADVP (RB enough)) (VP (TO to) (VP (VB avoid) (NP (NNS
blinking)) (SBAR (IN lest) (S (NP (PRP you)) (VP (VBP miss) (NP
(DT a) (NN moment)))))))))) (. .)) (S (NP (NP (DT The) (JJ
Ancient) (NN One) (POS 's)) (NN fight) (NNS scenes)) (VP (VBP
are) (ADVP (RB perhaps)) (NP (NP (DT the) (NN coolest)) (PP (IN
with) (NP (NP (DT a) (NN suave)) (SBAR (FRAG (NP (NNP Swinton))
(S (VP (ADVP (RB barely)) (VBG breaking) (NP (NP (DT a) (NN
bead)) (PP (IN of) (NP (NN sweat) (NN while)))) (S (ADVP (RB
effortlessly)) (VP (VBG annihilating) (NP (PRP$ her) (NNS
enemies)))))) (. .)))))) (S (ADVP (RB Then)) (NP (EX there))
(VP (VBP are) (NP (NP (DT the) (JJ mind-bending) (JJ visual)
(NNS effects)) (SBAR (WHADVP (WRB where)) (S (NP (DT the) (NN
world)) (ADVP (RB literally)) (VP (VBZ turns) (PP (IN on) (NP
(PRP itself)))))))))) (: ;) (S (NP (CD one) (NN battle) (NN
sequence)) (VP (VBZ has) (S (NP (NNP New) (NNP York)) (VP (VB
```

8. Future Scope

Each tool has numerous future applications. The developed tools are independent of one another and are developed in java platform, so each tool can be adopted independently in a project and the output of each can be independently used for next iterations.

For instance, one can utilize the Tregex tool to create a parse sentence for graph and sub-graphs demonstration.

The tools TriplesExtraction and DOTGeneration can be used together or independently in a project to find a better solution of visualization of text annotations.

One can use DOTGeneration to make graphical representation to visualize the important concept of a text file which has its triples extracted.

These tools can be used individually or in a group in a format, a user wants to help iterate the output to find a next solution to their own problem.

The tools are capable of creating a gene pool or the mutation factor of a genetic algorithm that may help in reducing the process of natural selection over iterative fitness selection in a faster and efficiently.

The manual annotation done via Brat tool could be semi-automated by creating a Character pool in advance.

9. Conclusion

The main concept of the project was to make annotation much easier and clearer for the user. In an organization, it is hard for every module developer to know every aspect of the whole project. But if someone who is able to work on a module correctly and can justify the data in such a way that every module developer can understand the output, helps the whole project to work in a better way.

The aim of this project is to help manual annotation in a form so that manually annotated texts can not only be visualized but could also be used as a way to communicate with the rest of the project in a better way.

The tool Tregex interface allows a user to create parsed sentence of a text file which helps in creating sub-tree for that text file which in turn helps in visualization of these parsed texts to a graph.

The next is tool TriplesExtraction. This is an extension to Brat Rapid tool annotation. Triple Extraction is a tool that helps in extracting Subject, Predicate and Object from the annotation file created within a Brat Annotation project. This tool is an independent extension of Brat tool which means that, after one project of annotation is completed in Brat , one can visualize triples of that project without having to write RDF format for the same annotated project.

The next tool is DOTGeneration . This tool is an extension in visualizing a text file with respect to its triples. If a user can find Triples of a text file can it is easier for one to understand the parts of that file. But to visualize in a better way a map-like representation is what this tool focuses on. DOTGeneration is a tool that can provide a map-like structure so that the logic can be visually authenticated.

As all these tools are independent of each other one can use these three different tools as required by their own projects.

10. References

- ✚ <https://www.w3.org/TR/2004/REC-owl-features-20040210/#s1>
- ✚ <https://www.w3.org/RDF/>
- ✚ <http://wifo5-03.informatik.uni-mannheim.de/bizer/pub/LinkedDataTutorial/#intro>
- ✚ https://www.w3schools.com/xml/xml_rdf.asp
- ✚ <http://linkeddata.org/guides-and-tutorials>
- ✚ http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html
- ✚ https://en.wikipedia.org/wiki/Semantic_network
- ✚ <https://blog.algorithmia.com/introduction-natural-language-processing-nlp/>
- ✚ https://en.wikipedia.org/wiki/Natural_language_processing
- ✚ <http://brat.nlplab.org/>
- ✚ <https://stanfordnlp.github.io/CoreNLP/>
- ✚ <http://graphviz.org/>