CHALMERS

Mining Relations from Git Repositories

Applying Relation Extraction Technology to Git Commit Messages

Master of Science Thesis in Software Engineering

Rikard Andersson

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, August 2014

Mining relations from git repositories
Applying relation extraction technology to git commit messages

Rikard Andersson

Supervisor: Morgan Ericsson

Examiner: Matthias Tichy

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0) 31 - 772 1000

Department of Computer Science and Engineering
Göteborg, Sweden August 2014

Abstract

Text data can contain valuable information that is unavailable at a larger scale due to the unstructured nature of free text. Git repositories and Git commit messages within them are one such category of unstructured text data. Relation Extraction (RE) has enjoyed success as a solution to similar problems for a more generic case but also for more specialized domains such as life sciences. RE does however, remain largely untested for text data from Git repositories. This thesis contributes to RE and Software Engineering research by testing RE solutions developed for the generic problem on the domain specific problem of Git commit messages. An experiment is conducted where a custom-made relation extractor is tested on hand annotated Git commit messages drawn from popular public projects on GitHub. The results show that common RE solutions and their models cannot be directly applied to data from Git commit messages due to a very domain specfic language in which these messages are expressed. This prompts for future efforts into developing domain specific tools and models.

Keywords: Relation Extraction, Natural Language Processing, Git commit messages, GitHub

Acknowledgements

I would like to thank the people at Findwise for letting me use their splendid facilities, for giving me insight into their work and for welcoming me as one of the group. Special thanks to Svetoslav Marinov who has been very generous with both time and expertise whenever I have needed it. A lot of the ideas in the relation extractor described in this thesis is his contributions as well as relation extraction as a subject for thesis.

I would also like to thank my supervisor at Chalmers, Morgan Ericsson, who have provided knowledge and guidance through though passages. Moreover, the whole idea with experiments on Git commit messages was originally his.

Rikard Andersson, Göteborg, 8 August 2014

Contents

1

Introduction

Large quantities of information around us is expressed in the form of natural language in free text. Popular examples include sources from the internet such as social media, blogs, news articles, published papers, etc. These data sources contain potentially valuable information. The information is however difficult to harvest on a large scale due to the unstructured nature of the data format. This calls for technological solutions. Such solutions would include methods, tools, and systems to automatically structure and aggregate it. Common techniques to use for these purposes reside in research areas such as information retrieval, data mining, and Natural Language Processing (NLP). The latter of these, NLP, is specifically aimed at processing text data, structuring it and building models from it.

Within a completely different branch of science, Software Engineering (SE), researchers are trying to understand software development. One way of doing this is by analysing data fetched from large software repositories. Such repositories primarily contain artifacts directly related to the end product of software development. However, they also contain metadata about the development process some of which is produced by the developers themselves mainly for collaborative purposes. This data is created as chunks of free text which describe the development process and is meant to aid the retention of a rich history of changes. This is done in components such as commit messages, issues, and bug reports. Said data could certainly be of analytical interest to developers, decision makers, and researchers alike but is largely unaccessible due to the unstructured form in which it is represented.

There is thus valuable data about software development within the grasp of potential benefactors but unavailable to them due to lack of technology. NLP technologies proposes real solutions to these problems but remain largely untested for the domain. Different fields within Life sciences have similar sources of unstructured data produced by practitioners, e.g. patient journals, from which large quantities of immensely valuable

data could be extracted. For this purpose there are tools[1], research groups, commercial ventures[2], and even a dedicated conference[3] for NLP applications within Life sciences. As such the application of NLP technology on Life science data would be considered successful. This is a promising indicator that the same kind of success could extend to applications within SE.

For the above reason we intend this thesis to evaluate the use of relation extraction systems for free text on text data generated in Git repositories during the software development process. More specifically the focus will be on Git commit messages. The idea is to take successful concepts from previous relation extraction research on the generic case with well structured English sentences and apply them onto Git commit messages. Reporting and discussing the results will contribute to evaluating Relation extraction, in its current state, as technology ready to support this kind of data and task. We formulate the following research question:

> How successfully can supervised relation extraction systems trained on English free text be applied on Git commit messages?

To answer this question, an experiment will be conducted where a system for relation extraction will be applied to commit messages from some of the largest and most active repositories available on GitHub. Results from the experiments will act as data for a quantitative and a qualitative discussion.

The remainder of this thesis will introduce related work in Section 2 followed by a background for the study in Section 3 which introduces NLP, relation extraction, and tools in these domains. The background also introduces a specific system, Relex, which will be used in the experiments conducted in the study. Section 4 present details about the methodology used for the research including data, experiments, and delimitations. Section 5 lists the results from the experiments while Section 6 contains discussion and analysis of the results in order to answer the research question. Section 7 contain an analysis of the threats to validity and finally Section 8 will provide final conclusions and future work

[1] `http://zope.bioinfo.cnio.es/bionlp_tools/get_all_bionlp_tools_out?SUBMIT=Submit+Query`
[2] `http://www.ezdi.us/ezcac/technology/`
[3] `http://2013.bionlp-st.org/`

2

Related work

Data use in Software Engineering (SE) is studied by Halkidi et al. [17] who present a rundown of different data mining techniques and how they can be applied to extract information from software repositories in the interest of SE research. The authors introduce text mining and Natural Language Processing (NLP) as a special case of data mining and mention it as a possible way of extracting data from SE activities. Parallelly Hassan and Xie [18] coin the term software intelligence as a relative to business intelligence. They suggests that up-to-date information derived from software repositories can be used not only by software developers but also by decision makers, software researchers, and educators within SE. The authors list text data as one of three important data sources on which mining algorithms can be used to extract valuable information.

From the workshop Mining Software Repositories [1] a large quantity of tools have emerged [7]. A number of these tools are developed and used to mine text data from bug reports. These include both Runeson et al. [31] and Wang et al. [37] who use text mining and information retrieval techniques to identify duplicate bug reports. Similarly Wu et al. [39] introduce a tool reliant on the same algorithms aiding bug report redundancy and autocompletion. Efforts made using other data sources include Chen et al. [8] who use topic modelling to categorise software entities (e.g. methods, files, classes, and modules) based on comments and identifier names. Doing this they find topics that are more defect prone than others. Natt och Dag et al. [25] develop a tool for requirements management using NLP techniques.

Not only is researchers using data gathered from GitHub in their research [35, 36] but there is also work being done specifically on Git commit data. In a recent research effort Pletea et al. [28] analysed commit messages using available tools for sentiment analysis. Ali et al. [2] develop a tool that aims to keep traceability links between software artifacts up to date. One of the components of this tool mines commit messages from CVS and SVN repositories in order to create links between changes and the requirements

[1] `http://2015.msrconf.org/`

affected by them. This is done with simple NLP using vector space model and similarity measurements. In all of the tools mentioned here NLP technologies, albeit simpler than relation extraction, are used to mine information from software repositories. Information that can then be used to aid the development process or research of the same.

3

Background

In structuring unstructured data in the form of natural language there are two highly relevant areas of research: semantic parsing and information extraction. Semantic parsing is based on the structure of language and goes from free text to syntactic structure to semantic meaning. The idea is that semantic meaning can be found in syntactic components found in the text. A subcategory of semantic parsing is shallow semantic parsing also called Semantic Role Labeling (SRL). In SRL the task is to find predicates of sentences and all their constituents such as object and subject. A corpus of this is PropBank where intricate structures of arguments and predicate modifiers interact to create meaning [27]. By using the predicate as relationship, and its constituents as entities, SRL can work as a relationship extraction technique [9].

Information extraction in contrast is a more direct and pragmatic approach where less computationally expensive methods can be applied. Information extraction systems can still use the signals of syntactic structure, e.g. part-of-speech tags [5] and dependency parses [41], however the information is only used as features and hence does not constitute the entire analysis. As opposed to semantic parsing where every part of the text is included in the model, information extraction may focus on the parts of text that convey meaning. Since the data under consideration in this thesis is potentially of low syntactic quality with a lot of noise we propose information extraction as a solution domain more suited to deal with our problem. This chapter will introduce relation extraction generally and Relex, a relation extractor used in the experiments of this work, more specifically.

3.1 Relation extraction

Systems doing information extraction takes as input raw text and outputs a set of relationships between entities found in that text [5]. This task can further be divided into subtasks, normally consisting of at least named entity extraction and relation extraction [11]. From the point of view of relation extraction any other subtasks of information

extraction can be seen as pre and post-processing. Named entity recognition, often a pre-processing step to relation extraction, is a separate research area in itself and is often excluded from discussion in papers about information extraction systems. These focus instead on the relation extraction algorithm and how it is combined with named entity recognition and other pre-processing steps. The relationships extracted by these systems are often referred to simply as relations or as relation tuples with reference to the two entities between which the relation exist.

Relation extraction systems are normally limited to finding relations explicitly expressed in the text [34]. Latent and implicit relations are much harder to infer and normally require a different system entirely [12]. Another common delimitation is to only look for binary relations [5, 40]. Systems that do look at n-ary or nested relations are those closer related to semantic parsing [9].

As an example of what relation extraction produces, consider the following sentence:

> *"On October 9, 2006, it was announced Google will purchase YouTube for US$1.65 billion in stock"*

In this sentence a binary relation, *will purchase*, should be found between the two named entities *Google* and *YouTube*. A nested relation would be to add the amount (*US$1.65 billion*) or the date *October 9, 2006* as a modifier to the entire relation.

Relation extraction from large scale corpora was originally restricted to one or several predefined relation types per effort. DIPRE [6] is a prominent system in this traditional relation extraction paradigm with expansions such as Snowball [1]. Both of these are limited to extracting instances of a single type of relation. The Open relation extraction paradigm was invented as a reaction to this limitation. Efforts in this domain are not restricted to a fixed number of relation types. The first system in this domain was TextRunner [5][1] later systems include ReVerb [13], WOE [38], OLLIE [32], SONEX [22], and an unnamed system that will be referred to as XU [40] in the remainder of this thesis.

Another dimension in which to categorize different relation extraction efforts is what level of supervision the system requires. Most commonly systems are either supervised such as XU or semi-supervised such as TextRunner. Additionally some systems are exploring the use of coupled sources of structured and unstructured data used for training. This is called distant supervision [23] a variant of which is used in WOE where Wikipedia articles and information boxes are compared for patterns of relations. There is also a movement of unsupervised systems which use clustering techniques to group relation tuples by which relation they express [30], SONEX is one such system.

Each individual relation extraction system vary to some degree from other system in its domain but they also share many similarities between them. The system is normally a pipeline where raw text is the input and relation tuples are generated at the end. For each pipe in the pipeline an additional piece of information is attached and later used

[1] It should be noted however, that even though Banko et al. [5] are the first to suggest open information extraction as a branch of information extraction, preceeding systems such as Jinxiu et al. [19] operate within the same domain without labeling it as such.

by other pipes in order to achieve the end result in aggregate. The relation extraction process normally work with signals from other Natural Language Process (NLP) tasks which hence needs to be performed before relation extraction starts. All of these steps can be seen as pre-processing. The relation extraction process itself normally starts with a relation candidate identification step where possible relation tuples are determined. Following this the relation tuples are assessed for quality and finally the relation tuples found to be trusted are labeled with a relation phrase. This division follows the work of Xu et al. [40] who label these two concluding steps the binary task and the triple task.

3.1.1 Pre-processing

Different systems work with different features for the relation extraction process. From a relation extraction perspective, adding information for these features to the data is considered pre-processing. Common pre-processing steps are sentence detection, tokenization, and part-of-speech tagging. A sentence detector finds sentences by looking for their ends. Similarly, a tokenizer finds individual words, called tokens, and tags them as such. During the part-of-speech tagging each token gets assigned a part-of-speech label e.g. verb, adjective, or noun. Sentence detection and tokenization is done as part of the pre-processing for all relation extraction systems in this thesis. Part-of-speech tags are not used by DIPRE and Snowball but by all the others. The authors of both TextRunner and SONEX both claim to use "light NLP" techniques where TextRunner in addition to sentence detection, tokenization, and part-of-speech also use a noun-phrase chunker. The reason for this restriction is that any more advanced techniques would not perform as well on the noisy data they anticipate.

Another common pre-processing step to use is dependency parsing which is a type of syntactic analysis applied on sentences. The analysis produces a hierarchical syntactic graph with words as nodes and relations between words as edges. The edges can either be labeled or not. This structure is called a dependency tree. The dependency tree has a root which is normally the predicate with nodes underneath being arguments and sub clauses. The dependency tree is very suitable for relation extraction and features from dependency trees are very common systems in the domain and used by ReVerb, WOE, OLLIE, and XU among others.

Commonly relations are defined as existing between two entities with a relation phrase to describe what relation holds up between them [5, 38, 40]. What an entity is can however be interpreted differently. Most of the systems discussed in this chapter interpret noun phrases as entities while other options include the use of external sources to define an entity [34] or extract entities based on regular expressions like done in DIPRE and Snowball.

Additionally there are those like SONEX that use named entities as the entities and named entity recognizers to extract entities for their relation extractors. The term Named Entity was coined to describe a task in the sixth Message Understanding Conference (MUC-6) [16]. For this task a named entity was described as a unit of information and detailed with a few examples including persons, geographic locations and organizations. While these are three common categories to test named entity recognition, others

have suggested additional categories to suit the needs of the application for which named entities are employed. A few examples include chemical names, job titles, and project name [24].

Commonly both pre-processing pipes and supervised relation extraction pipes use machine learning components trained on gold standard data. Such data is hand annotated by multiple annotators with a systematic way of peer-review [21].

3.1.2 Candidate identification

Relation candidate identification is where possible instances of relations are found in the raw text. This step is often followed by another called relation candidate extraction which is where each found instance is extracted along with its context for further processing. Such an extracted instance is often called a relation candidate. In this thesis these steps are bundled under the relation candidate identification step.

There are a few different strategies for relation candidate identification among the systems mentioned previously. The most common one is entity centric relation identification. This is where entities occurring together within a boundary are paired up to form an instance from which to draw a relation candidate. DIPRE uses this strategy bounded by a set number of maximum tokens in between the entities. Snowball, TextRunner, WOE, and SONEX all limit their bounds to entities occurring within the same sentence. XU also bounds relation candidates to within sentences but additionally require a maximum number of token between the entities. In contrast to these entity centric strategies ReVerb uses a relation centric approach where the relation phrase is extracted first. The context is then matched for entities in its surroundings using heuristics. Finally OLLIE identifies relation candidates by matching patterns for both entities and relation phrase in combination.

3.1.3 Binary task

The binary task as defined by Xu et al. [40] is to determine whether or not a relation exists between two entities. For the authors and their system, XU, this is done by classifying relation candidates as true or false. The classification is done with a Support Vector Machine (SVM) [10] using tree kernels based on dependency parses. TextRunner [5] employ the same type of classification but use a Naive Bayes classification algorithm instead of the SVM used in XU. Features include part-of-speech sequence before, between, and after entities. TextRunner's classifier is trained using relations from a type of linguistic parse called constituent parse.

Later versions of TextRunner [4] as well as WOE instead employed a graphical model called Conditional Random Fields (CRF) [20] where each token between two entities were labeled as part or outside of a relation phrase. In this design, if no relation phrase is found there is no relation.

DIPRE and Snowball both use primitive pattern matching techniques where each relation candidate is matched against generated patterns to determine if there is a relation or not. The patterns are made up of features including tokens before, between, and

after the entity pair as three separate ones.

OLLIE and ReVerb both assigns relation candidates a confidence value using a logistic regression classifier. Relation candidates with a confidence value below a threshold value can later be discarded.

SONEX, using an unsupervised approach, is structured somewhat differently compared to the other systems discussed here. For the binary task it simply assumes that a relation exists between two named entities if they occur in the same sentence. The system then use clustering algorithms to group similar relation instances with eachother. Relation instances in the same cluster are then assumed to be of the same relation type. Features used by the clustering algorithm include unigram tokens, bigram tokens, and part-of-speech sequence.

3.1.4 Triple task

Xu et al. [40] define the triple task as classifying a relation phrase as true or false with the assumption that the relation candidate truly contains a relation. This definition is good fit for how XU is structured. The triple task in XU is done by heuristically eliciting relation phrase candidates based on patterns. This relation phrase candidate is then classified as true or false with an SVM. Here we propose a slightly broader defintion of the triple task where both of these steps can be transformed into one. That would be to simply identify a relation phrase which explains the semantic relation between two entities. Using this definition we can explain the triple task solution in SONEX. Here relation instances are clustered and a relation phrase is extracted for each cluster to label all the relation instances in the cluster with. This is done by eliciting the centroid of the tokens between entities in each cluster.

The first version of TextRunner [5] extracted all tokens between entities as a candidate relation phrase and then heuristically eliminated words using part-of-speech tags. The relation phrase was then used as a feature for classifying in the binary task and accepted as relation phrase if the entire relation candidate was classified as a true relation. Later evolutions of TextRunner [4] as well as WOE take care of the triple task and binary task in parallel using a CRF model to label tokens between entities as part of the relation phrase or not. Similarly OLLIE employs pattern matching with a dependency tree against the training data to elicit both entities and relation phrases. Elicited relation candidates are then assessed for confidence as described in the binary task (Section 3.1.3). Features used by these systems are largely the same and include tokens, part-of-speech unigrams, part-of-speech sequences, and in some cases dependency trees.

The division of a systems functionality into a binary and a triple task is incompletely due to some systems simply being structured differently. ReVerb for example takes care of the triple task as part of the candidate identification. The system finds relation phrases as relation candidates and then use a set of heuristics to locate the closest noun phrase to the left and right of the identified relation phrase. These noun phrases are then used as entities between which the found relation phrase holds. ReVerb is as such a relation centric system whereas the other systems detailed here are instead entity centric. Simlarly the nature of traditional relation extraction systems like DIPRE and Snowball

where only a single or a limited set of relation types are evaluated make the triple task redundant.

3.2 The relation extractor Relex

So far we have established that semantic parsing and SRL breaks sentences down into a structure of predicates and arguments hoping to explain meaning with these components. Since this is done using syntax, assuming correctly formulated English sentences, it is reasonable to expect that when syntax is bad these techniques will fail. Considering the potentially low text quality data this thesis circulates around leads to the conclusion that information extraction is a better solution domain for the task at hand. Furthermore, as is the case with TextRunner and SONEX, a system that solves the problem will likely rely on "light NLP" techniques for the same reason of noisy data. Based on this reasoning, with influences from the systems described under the previous section (Section 3.1), we have developed a simple yet effective (see Appendix D and E) system in collaboration with an industry partner, Findwise.

The system, which we refer to as Relex [3] is a system in the open relation extraction domain. The pipeline has pre-processing with light NLP techniques, a candidate identification step, a component for the binary task (binary classifier), and one for the triple task (relation phrase extractor).[2]

3.2.1 Pre-processing

In order to operate on the data a few steps of pre-processing is made up of sentence detection, tokenization, and part-of-speech tagging. The sentence detection and tokenization splits the text into units of sentences and tokens respectively while the part-of-speech tagger assigns a part-of-speech label to each token. All these are performed with components from Apache OpenNLP[3] with free models released by Apache[4] and trained on data dervied from well structured English texts such as newswire or published papers. The relation extractor expects entities as input to extract relation candidates. Hence, Relex have capabilities to support entities extracted from both a noun phrase chunker or a named entity recognizer[5].

3.2.2 Relation candidate identification

The relation candidate identification is the first step of the actual relation extractor. The candidate identifier combines pairs of entities into relation candidates to be evaluated further. Additional rules can be added to this step in order to limit what is considered a possible relation. An example is to only allow for relations where entities co-occur

[2]Relex can be found in the project repository at `https://github.com/Rikard-Andersson/relex`

[3]`https://opennlp.apache.org/`

[4]`http://opennlp.sourceforge.net/models-1.5/`

[5]Since all entities in the experiments will be derived from annotations a noun phrase chunker or named entity recognizer is left out of the implementation

within a boundary such as a sentence or a span of a maximum number of tokens. This system by default only limits to the boundary of a sentence but have the capacity for a maximum tokens distance boundary as well.

3.2.3 Binary task

In the binary task each relation candidate identified is classified as true or false. Like XU, Relex use a supervised algorithm where the binary task is solved with a support vector machine (SVM). Contrary to XU the SVM does not use a kernel tree but instead rely on two kinds of light NLP features: bag-of-words (BOW) and part-of-speech sequences. These are collected for three different positions: before, after, and between the entities of the relation candidate. BOW is simply a feature where all the tokens found in the range of the feature are represented ignoring the order of the tokens. A part-of-speech sequence stores the total sequence of part-of-speech tags for the feature and hence takes order into account. This follows the work of such system as SONEX and TextRunner which both rely on BOW and part-of-speech tags and sequences as features.

3.2.4 Triple task

The triple task decides on how to label a relation using explicit words occurring in the context of the relation. The Relex component solving the triple task is referred to as the relation phrase extractor. The relation phrase extractor is a rule based component operating on part-of-speech tags between the entities in a relation. From the sequence of tokens in this context the last verb is selected as the relation phrase. If no verb is available it insteads finds the last noun. If none of these occurs it simply uses the last token in the sequence between the entities. Since the component rely on heuristics it can be considered unsupervised. The behaviour is similar to systems such as ReVerb which, instead of locating a relation phrase, use heuristics to locate the entities referred to by a relation phrase. XU also use heuristics to elicit a relation phrase but then goes on to classify this relation phrase as true or false.

4

Methodology

In order to provide support for a discussion and to arrive at an answer to our research question we will conduct an experiment. The experiment is centered around a relation extraction system, Relex, as introduced in Section 3.2. Relex is inspired by state-of-the-art relation extraction systems and will be used with models trained on well formed English sentences. The idea is to put Relex to work on git commit messages fetched from GitHub and evaluate the results both quantitatively and qualitatively.

4.1 Data

Two datasets are used for the experiments in this thesis. One is used to elicit training data for the binary classifier component of Relex. Another one is used to elicit GitHub commit messages which will later be used as actual test data.

4.1.1 Training data

Based on a set of relation candidates Relex performs the binary task, binary classification of the relation candidates as true or false. Models for the SVM component used in Relex to perform the binary task (Section 3.2.3) are generated feeding the system with labeled training data. The training data is derived from the PropBank-style predicate-argument annotations provided in the training and testing data for the ConLL 2011 shared task [29]. The data originally used for coreference resolution comprise of syntactically well structured English sentences from newswire sources such as news articles, broadcast conversations, magazine articles, and web data. Additional to predicate-argument and coreference the data also include annotations such as part-of-speech, word sense, and named entities. We refer to this data as the ConLL data. An exerpt from a sentence as it appears in the ConLL data can be found in Appendix C. The ConLL data is divided into a big set intended for training and a much smaller set intended for testing.

The training data is comprised of 44,687 sentences while the test data is comprised of 5,814 sentences.

From the ConLL data training set three different sets are extracted meant for training of three different models for the binary classifier Relex uses[1]. The three sets differ mainly in what types of entities relations are allowed between. The first dataset only allow relations where both the first and second arguments are named entities, this set is henceforth described as NE-NE (Named Entity - Named Entity). The second data set allows one of the arguments, first or second, to be a noun phrase, this set is henceforth known as NE-NP (Named Entity - Noun Phrase). The third and final dataset allows relations between any two nouns or noun phrases and will henceforth be known as NP-NP (Noun Phrase - Noun Phrase).

Apart from entity restrictions the datasets also differ in how relations are elicited for them. The relations for the NE-NE set are found by looking for named entities as tagged by the ConLL annotators and extracting all pairs as relation candidates. If a relation is found in the PropBank predicate-argument annotations the relation candidate is considered a true relation, representing a positive data point. All the pairs that are not connected by a predicate end up as false relations representing negative data points. Using this technique we end up with a training set of 6,677 data points with a distribution over negative and positive as seen in Table 4.1.

In the NE-NP and NP-NP datasets the negative data points are collected by discriminating based on a dependency tree generated by MaltParser [26]. As a first discriminator relation candidates are extracted from the PropBank predicate-argument annotations. Next, the distance in the dependency tree is calculated between the first and second entity. If the distance is greater than three we label the candidate as false, less than three as positive and if it is precisely three we leave it out entirely. Additionally, in order to be considered a true relation, the arguments (noun phrases or named entities) must follow the predicate pattern with a subject and an object as elicited by the dependency tree. Any other label and they are reject as a false relation. This way only explicit binary relations are elicited and nested relations and subclauses are left out. The detailed method renders the NE-NP dataset with 3,692 training points and NP-NP dataset with 10,131 training points with a distribution over negative and positive as seen in Table 4.1.

	Positive	Negative
NE-NE	1811	4866
NE-NP	1278	2414
NP-NP	3811	6320

Table 4.1: Training data distribution over different sets

[1]Corresponding sets are extracted the same way from the ConLL test sets and used for testing Relex performance on both the binary and triple task as seen in Appendix D and E respectively

The reason for less strict rules for the extraction of relations in NE-NE set is the fact that it otherwise turns out very small. The opposite goes for the NP-NP set in applying the NE-NE procedure, it would become very large and require a lot of processing power and memory. A total of 120,000 relation candidates (11,171 positive and 99,736 negative) would be the result.

Having three different sets of training data constructed using different methods enable us to experiment with a wider range of natural language techniques in order to find the most promising one for our purposes. The NE-NE dataset contain signals from relations with named entities specifically. The NP-NP set is more likely to contain directly binary relations and the NE-NP set is a mix between them both. The models produced by each dataset will be referred to as NE-NE, NE-NP, NP-NP corresponding to which dataset was used to train it. Appendix D and E show results from testing Relex on the smaller ConLL test dataset for the binary and triple task respectively. This serves as validation that Relex is at least a descent system for relation extraction on well structured English.

4.1.2 Test data

The git commit data is fetched from GHTorrent [15] which is a data collection initiative that extracts data directly from GitHub. The initiative originated from a need in a research project in 2012 and later extended to a complete and public repository. Today the complete GHTorrent contain information from the majority of events on GitHub from 2012 through to 2014 and continuous backtracking is made to include earlier data. Examples of the data stored include commit messages, commit comments, issues, and issue comments. GHTorrent's data collection is limited to the publicly available repositories and hence all repositories in private mode will not be represented.

Due to the size of the complete dataset (3.5 TB) this work have been performed on one of the smaller subsets of the same datasets, namely the "MSR 2014 Mining Challenge Dataset". This subset has the same structure as the full set and contains all data from the top-10 most starred repositories, including all their forks, from the nine most popular programming languages on GitHub. The full list of projects included in the dataset can be found in Appendix F. The resulting dataset contains 601,080 commits with corresponding commit messages. From these commits we randomly[2] selected and annotated 600 commit messages. These annotations with its underlying data is to be included as test data in our experiments and to represent GitHub commit messages. We refer to this data as the GHTorrent data.

Each commit message was annotated with entities and relations. As evident by the background (Sections 3.1.1 and 3.1.2) different systems employ different definitions of what constitutes an entity, be it a noun phrase or some other set of categories for named entities. For the application studied in this thesis useful entities would be connected to the domain, e.g. names of methods, classes, issues, and files. We propose and use a list of entity types between which useful relations could exist (see Appendix A).

Two different relation types was considered in the GHTorrent data: entity relations

[2]We used atmospheric noise from `http://www.random.org/` to generate random numbers [3].

and author relations. Entity relations are those fulfilling the normal case of relation extraction where a relation is found to exists between two named entities in the text. Entity relations are fully explicit which follows the delimitations we will discuss in Section 4.3 below. After a quick visual inspection of the data a lot of relations between an implicit author entity and an explicit entity were found in the text. Below follows three examples of these type of relations found in the actual GHTorrent test data.

First example is where the author has done something to something, e.g. changed a file:

"Update CHANGELOG to mention the json_escape change"

In this case there is a relation labeled *"Update"* between an implicit author entity and the *"CHANGELOG"*. A second example is where the author have solved an issue:

"Fixes #1097."

Where the author have fixed the issue named *"#1097"*. Note that the two examples above could also refer to the commit itself rather than the author. We allow for this variation of what the implicit author entity may refer to as long as we find an explicitly expressed relation between the implicit and the explicit entitiy. A third example is where the author have merged another branch with the current one:

"Merge branch 'master' of github.com:akka/akka"

In the above commit message there is a relation between an implicit author entity and the branch entity *"master"*. We choose to label the relation types described here as author relations. There is normally no explicit author entity in the commit message and our system handles only relations between entities which are found in the same sentence. Therefore, during relation candidate identification, we add a mock author entity at the beginning of each sentence and let author relation candidates be those between this entity and other entities found in the sentence.

Annotation was done by the author of this thesis alone. Each document was annotated in two steps. First step entailed elicitation of all entities using a list of allowed entity types. See Appendix A for the full list of entity types. In the second step the entities from the first iteration were considered for relations with each other, each with a connecting relation phrase between them. A list of rules and guidelines was followed in an effort to construct a dataset of consistent annotations in line with the delimitations of the study. The full list of guidelines can be found in Appendix B. The procedure with two passes to annotate entities and relations separately mimics the behaviour of the relation candidate identification used in Relex. It also ensures that no entities are left out because it is not in a relation tuple. Doing so would be to give Relex an unfair advantage since fewer negative relation candidates would be found. For annotation purposes we used the brat annotation tool [33]. The full annotation project is available at `https://github.com/Rikard-Andersson/GHTorrent-Brat`.

4.2 Experiments

Relation extraction as described in Section 3.1 pre-processes raw text by attaching annotations needed for the relation extraction process itself. The relation extraction process itself can then be divided into relation candidate identification, binary task, and triple task. Relation candidate identification finds possible instances of relations in the pre-processed text and marks them as relation candidates. The binary task is then to label relation candidates as true or false and the triple task takes care of finding a relation phrase for the instances labeled as true. Relex, described in Section 3.2, is structured in this exact way. Two categories of experiments were conducted on the GHTorrent data, one testing the Relex component solving the binary task and one testing the Relex component solving the triple task.

4.2.1 Binary task

For each model (NE-NE, NE-NP, and NP-NP) the binary task was tested for author relations and entity relations separately. Thus, a total of six tests were run, two for each model. The GHTorrent test data was pre-processed using the same pre-processing used in Relex with tags added for sentences, tokens, and part-of-speech. The entity annotations were derived from the handmade annotations and the relation candidate identification was done using the Relex component for this task.

Authors in the field of relation extraction, as with many other machine learning problems, use recall, precision, and f-measure to evaluate performance of their algorithms, methods, and systems [1]. The same metrics will be used to measure performance on the binary task in this work as well.

4.2.2 Triple task

Both entity relations and author relations were tested for relation phrase extraction and an additional aggregate performance was calculated. The triple task is sequentially dependent on the binary task which may mislabel false relations as true. As a result the relation phrase extractor might in a live environment attempt to find relation phrases between entities that have no relation between them. This type of faulty behaviour should be attributed to the performance of the binary classifier and will thus not be accounted for in the evaluation of the triple task. Therefore only the true relations from the annotations will be used in these tests. Thus, the triple task component of Relex, also called the relation phrase extractor, will be tested separately from the rest of the system. The true relations are passed through the relation phrase extractor and evaluated as correctly elicited only if the complete phrase is identical to the one annotated by hand.

When evaluating the triple task there are no fixed classes to assign the answers to and presenting recall and precision becomes redundant. This is due to relation phrases being one in every relation candidate and one relation phrase is extracted per relation candidate by the algorithm. If the relation phrase extracted by the algorithm is wrong

this will register as a false positive while the true relation phrase will register as a false negative. This symmetry results in precision equaling recall. Instead the performance of the relation phrase extractor is measured by accuracy, i.e. how often it finds the correct relation phrase which is incidentally the same as both recall and precision in this case.

4.3 Delimitations

The delimitations of the experiments conducted in this thesis are implied by the delimitations of the system used to perform them, Relex. Relex has in turn inherited its limitations from the systems in the relation extraction domain that inspired it.

An implicit relation can be exemplified by an instance where entity A has an explicit parent relation to both entity B and entity C, as illustrated in Figure 4.1. There is hence an implicit sibling relation between B and C [12]. As is the case with most of the systems described in the background (Section 3.1), this study is limited to only explicit relations. Moreover, this study is limited to relations where the explicit phrase expressing the relation is located between the entities in the relation candidate. This delimitation makes for a less complex task and leaves a more manageable scope.

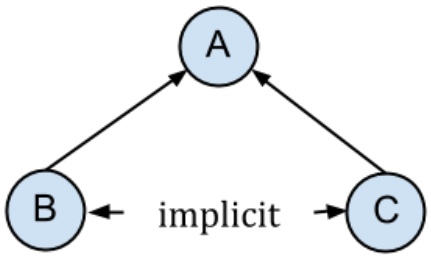

Figure 4.1: Explicit parent relations and implicit sibling relations

Binary relations are those where the relation only include two entities. N-ary relations in contrast are such that relations extend beyond the two entities and can include several other entities with intricate relations among them. Semantic role labeling can produce n-ary relations where the predicate-argument structure may include several entities. However, when these are converted into a relation extraction system they are often converted to binary relations and any more elaborate structures are discarded [9, 14]. This study is limited to binary relations.

All data and models used in Relex and in the experiments are for text data expressed in English. Relex could of course be trained and tested with data from other languages but the question is how relevant this would be in a software engineering context assuming that most software engineering activities are conducted in English.

What constitutes an entity varies across different systems. Delimiting in this dimension comes down to what is useful for the problem at hand. See Section 4.1.2 for detailed description of how the issue is handled in this thesis. The task of entity recognition or extraction is left out of the experiments and will be derived from hand annotations.

The delimitations mentioned in this section are implied for the entire study and extend to both datasets as well as the implementation of Relex. In short, delimitations on relations are to semantically explicit, binary relations between entities expressed in English text.

5

Results

In order to evaluate how well relation extraction can be applied to git commit data experiments have been conducted using the relation extractor Relex. The data used for the experiments were fetched from public projects on GitHub and annotated with entities and relations between these. The tests were conducted with each of the two main components of Relex separately. Each component deals with a specific task: the binary task and the triple task.

5.1 Binary task

The binary task is a classification task where a relation candidate is being labeled as true or false representing if the relation candidate indeed contains a relation or not. The results are divided into what relation type is being evaluated. Recall, precision and F-measure are calculated. Recall measures the proportion of predicted true relations that are actually true to all true relations. Precision measures the proportion of correctly predicted true relations to all predicted true relations. F-measure is a harmonic mean of recall and precision. For each relation type the performance of the three models NE-NE, NE-NP, and NP-NP are presented separately.

5.1.1 Entity relations

Entity relations describe relations between two explicit entities in text with an explicit relation phrase binding them together. From the 600 commit messages Relex found 678 entity relations using the relation candidate identification component. 66 of these relations are true relations (by annotations) and the remaining 612 hence false.

From tables 5.1, 5.2, and 5.3 it is evident that the NE-NE model is best able to find true relations (recall of 85 %). Compared to the other two models the NE-NE is more generous in assigning true relations and predicts well over half of the relation candidates to be actual relations while the other two are more restrictive and predict about 15-20 %

		Predicted		
		True	False	Total
Actual	True	56	10	66
	False	341	271	612
	Total	397	281	

Recall	0.848
Precision	0.141
F-measure	0.242

Table 5.1: Performance of Relex on the binary task using the NE-NE model on entity relations from the GHTorrent data

		Predicted		
		True	False	Total
Actual	True	18	48	66
	False	90	522	612
	Total	108	570	

Recall	0.273
Precision	0.167
F-measure	0.207

Table 5.2: Performance of Relex on the binary task using the NE-NP model on entity relations from the GHTorrent data

		Predicted		
		True	False	Total
Actual	True	27	39	66
	False	105	507	612
	Total	132	546	

Recall	0.409
Precision	0.205
F-measure	0.273

Table 5.3: Performance of Relex on the binary task using the NP-NP model on entity relations from the GHTorrent data

of the relation candidates to be true relations. As a result the NE-NE model produce a slightly lower precision with only about 15 % of the true predictions being actually true. Overall, precision scores are low with the NP-NP model scoring the highest with every fifth extracted relation being a true relation. On the whole, entity relations extracted by Relex with any of the three models are more likely not to be true relations by a wide margin.

5.1.2 Author relations

Author relations express a relation between an implicit author or commit entity and an explicit entity with an explicit relation phrase between them. From the 600 commit messages Relex found 1,118 author relations 338 of which are true relations (by annotations)

and the remaining 780 false.

		Predicted		
		True	False	Total
Actual	True	188	150	338
	False	589	191	780
	Total	777	341	

Recall	0.556
Precision	0.242
F-measure	0.337

Table 5.4: Performance of Relex on the binary task using the NE-NE model on author relations from the GHTorrent data

		Predicted		
		True	False	Total
Actual	True	160	178	338
	False	210	570	780
	Total	370	748	

Recall	0.473
Precision	0.432
F-measure	0.452

Table 5.5: Performance of Relex on the binary task using the NE-NP model on author relations from the GHTorrent data

		Predicted		
		True	False	Total
Actual	True	237	101	338
	False	230	550	780
	Total	467	651	

Recall	0.701
Precision	0.507
F-measure	0.589

Table 5.6: Performance of Relex on the binary task using the NP-NP model on author relations from the GHTorrent data

Compared to the performance on entity relations detailed in Section 5.1.1 Relex produce a better overall result on author relations using any of the models. From the performance presented in tables 5.4, 5.5, and 5.6 it is obvious that the NE-NE model is once again most generous in assigning true relations to candidates doing so rather unsuccessfully with a precision of roughly 25 %. It does however manage to find 56 % of the positive relations which is better than the NE-NP model with 47 %. The NP-NP model does however outperform both of the other two models quite clearly with a recall of 70 % a precision of 51 %, and a harmonic mean of 59 % making it the best results on the binary task. The relations it assigns as true are however just barely more likely to

actually be true than false. Hence, in the best case scenario, approximately half of the relations extracted by Relex would be inaccurate.

As a benchmark the results for both entity relations and author relations can be compared to those of applying Relex to solve the binary task for the ConLL test data. These results can be seen in Appendix D.

5.2 Triple task

The triple task is where the relation phrase between actual relations are to be extracted. Relex does this with a simple rule based annotator working exceptionally well on the ConLL data (see Appendix E). The performance of the triple task is measured in accuracy and the results on the GHTorrent data can be seen in Table 5.7.

Relation type	Accuracy
Entity relations	0.515
Author relations	0.675
Both kinds of relations	0.649

Table 5.7: Relex triple task accuracy on different relation types in the GHTorrent data

The relation phrase extractor is correct in about two of three cases in the author relations and about every other case in the entity relations. Just as in the binary task Relex struggles more with the entity relations than with the author relations despite the fact that the author relations are a unique case constructed for the GHTorrent data. This indicates that entity relations are indeed expressed differently in the GHTorrent data than it is in the ConLL data which is considered to be well formed English sentences. There is a significant drop from the 92-99 % accuracy of the relation phrase extractor on the ConLL data (see Appendix E).

5.3 Qualitative results of pre-processing

The pre-processing used to run Relex poses less of a problem on well formed English sentences. In git commit messages however, the syntax and grammar are not of the same standard and the vocabulary is different. It is therefore necessary to qualitatively look at the performance of pre-processing. This provides more material to use for analysis of performance of the binary and triple task components of Relex on GHTorrent data. It should be noted that in some cases in the GHTorrent data, even a human will have a hard time labeling the data due to the specialized way of language use in the commit messages.

5.3.1 Sentence detector

Here follows some qualitative results of the sentence detector. Commit message authors often logically split their statements by new lines which work as sentence delimiters. Names with periods do however sometimes present a problem like in these cases where an actual sentence is annotated as two:

"depends: update .gitignore for depends for osx and ios builds"

is annotated as two sentences: *"depends: update .gitignore"* and *"for depends for osx and ios builds"*.

"Fix check for browser.mozilla so that Safari is not flagged as mozilla"

is annotated as two sentences: *"Fix check for browser.mozilla"* and *"so that Safari is not flagged as mozilla"*.

"Lua: take 5.1.4-3 patch from MacPorts"

is annotated as two sentences: *"Lua: take 5.1.4-3"* and *"patch from MacPorts"*.

From annotations in the GHTorrent data no case occur where a sentence is split such that the relation candidate identifier is not able to pair entities into a relation candidate where there is an actual relation. However, with occurrences as those above of wrongly detected sentences, this is a possible source of future errors.

The opposite relation where two sentences are interpreted as one also occur. The two lines:

"Previous init will not work with Fedora15+
Fedora14 has been EOF 2011-12-08"

is interpreted as one sentence while it might be more appropriate to divide it into two sentences. What this does is create a relation candidate between entities that would not originally have a relation between them creating a source of possible errors for Relex.

5.3.2 Part-of-speech tagger

The part-of-speech tagging is another point where the pre-processing might be cause for errors in downstream classification steps. Consider the following four exerpts from commit messages:

1. *"This commit gets rid off code wrapping that was previously used by..."*
2. *"SI-7331 tb.parse returns unpositioned trees"*
3. *"Instead combination of templateStats/accept(EOF) is used."*
4. *"...but it was forgetting to reset ContainsLDynamicVariable in case eval is used."*

Sentence	Word	OpenNLP POS	True POS
1	*"commit"*	verb	noun
2	*"tb.parse"*	adjective	noun
3	*"templateStats/accept(EOF)"*	adjective	noun
4	*"ContainsLDynamicVariable"*	adjective	noun

Table 5.8: Examples of part-of-speech (POS) mislabelings by OpenNLP part-of-speech tagger on GHTorrent data

Each one of these sentences have tokens which are wrongly annotated by the part-of-speech tagger used in Relex. These corresponding errors are listed in Table 5.8

All of the cases use highly specialized words for the domain from which the GHTorrent data is derived. With a perfect system all of these would be labeled as nouns but clearly are not by the part-of-speech tagger used in Relex. Mislabeling also happen for other tags:

> *"This pull request doesn't attempt to generalize this approach in any way and re-use it all over the place due to the caution of possible accidental compatibility breakage."*

In this case *"re-use"* is labeled a noun where it should be labeled a verb.

Comparing the results obeserved in this chapter and the ones for the ConLL test data (see appendices D and E) it is clear that applying Relex perrform worse when applied to the GHTorrent data than when applied to ConLL data. This is true independent of which type of model is used and for both the binary and triple task. Drilling further down into vital processing components such as the sentence detector and the part-of-speech tagger both make consistent mistakes on the GHTorrent data.

6

Discussion

The relation extractor Relex have been applied to extract relations from git commit messages with severe drops in performance from other benchmarks. In order to draw conclusions from the results and generalize them to the field of relation extraction and the applicability of such techniques on git commit data we need to analyse the results further. Relex is a system inspired by the frontier of the relation extraction research field and proven to work on well structured English sentences. Moreover, the system is based on very simple features derived from light NLP techniques. As such Relex can be considered both a relative to the systems produced through relation extraction research but also a common denominator. Any shortcomings of Relex based on basic concepts will therefore be discussed as an extension to the entire family of relation extraction systems on which Relex is based.

6.1 Analysis of results

The additional data presented and analysed here, expressed as excerpts from git commit messages, is drawn from processing made with the NP-NP model. This is due to the fact that this model produced the best results and show the most promise of being part of a functional solution to the problem discussed in Section 1.

6.1.1 Sentences

From the examples presented in Section 5.3.1 of sentence detection performance it is evident that the sentence detector is sometimes failing due to the way sentences are expressed in the domain. One big problem seem to be punctuation and capitalization being used differently from well formed English sentences. Dots occurring in entities such as file or method names is causing the sentence detector to break a sentence. Another example is the following sentence where one full sentence is detected while the author

most likely intended for four different sentences, one for each semicolon and possibly a separate sentence for *"[DB MIGRATION REQ'D]"*.

> *"[DB MIGRATION REQ'D] Added plugin database tables and DAO; Added network column to relevant tables; Moved work of fetching active plugins to common/init.php"*

Relex relies on sentence detection for its relation candidate identification pipe. It uses the annotations to limit relation candidates to only include entity pairs appearing within sentences but also to find a suitable mock-entity for the author relations. If the sentence detector splits a sentence relation candidates might be overlooked. This could not however, have affected the outcome of the experiments in this study since true relations are elicited from annotations. If the sentence detector instead fail to make a split between two sentences additional false relation candidates are identified. This introduces unnecessary instability as these in cross-sentence relation candidates are evaluated and possibly falsely labeled as true. Moreover the author mock entity will be placed in the wrong context and the wrong features are extracted for classification. Both these cases cause an error each in the above sentence. *"[DB MIGRATION REQ'D]"* is classified as a true relation with *"network column"* and the author relation with *"DAO"* and relation phrase *"Added"* is labeled as false.

Other cases where the sentence detector used in Relex fails include the following sentence.

> *"extracted the built in profiling out added pp=profile-gc-time"*

It is obviously hard for a sentence detector trained on English to label this sample as anything other than one sentence while it should perhaps be two: *"Extracted the built in profiling out."* and *"Added pp=profile-gc-time."*. The author provide no signal to indicate this would be the case which illustrate a point that will recur in this analysis: Git commit messages are not well formed sentences and thus any machine learning algorithm trained on such will have to be very noise tolerant to function well.

6.1.2 Part-of-speech

The part-of-speech tagger is another component of Relex that is mislabeling the GHTorrent data. Both the binary classifier and the relation phrase extractor are dependent on part-of-speech tags. The binary classifier uses them as features and the relation phrase extractor find tokens that match part-of-speech tags in a scheme of rules. As seen in Table 5.8 the part-of-speech tagger has problems where domain specific language is used. Another example of this can be found in the commit message below in which a true entity relation can be found between *"postgresql"* and *"postgres-xc"*.

> *"postgresql conflicts with postgres-xc"*

This is arguably a rather simple relation and should hence be easy to detect but Relex classifies it as a false. The word *"conflicts"* is here labeled a noun part-of-speech tag while

verb would be more correct. Moreover both *"postgresql"* and *"postgres-xc"* are labeled with an adjective part-of-speech tag while they are infact nouns. Further complicating the classification is the fact that no tokens exists before or after the entities which would assumably not happen in the newswire based training data.

Relex performance on the triple task is also suffering due to mislabeled part-of-speech tags. This is only natural since the relation phrase extractor is exclusively reliant on part-of-speech tags to work. In the example from above the faulty part-of-speech tags does not pose a problem since the relation phrase extractor gets it right anyway, one could argue luckily so. In other cases however, tokens with wrongly assigned part-of-speech tags do cause problems:

"cluster: Rename destroy() to kill(signal=SIGTERM)"

The token *"Rename"* is labeled as an adjective instead of a verb and *"cluster"* is instead selected as relation phrase in the author relation with *"destroy()"*. In this case the sentence detector could also be to blame for including *"cluster"*.

"Merge pull request #370 from Memphiz/airplay"

In this case, which we will see later is a recurring one, *"pull"* is labeled a verb and *"merge"* a noun. The token *"pull"* being a verb is nothing strange in a normal vocabulary but *"pull request"* in this domain refers to a noun phrase while *"Merge"* is actually a verb.

The cases above and the ones presented in Table 5.8 in the results (Section 5.3.2) can all be attributed to words specific to the domain. This is hard for an annotator to handle if it is not trained on the vocabulary in question. Part-of-speech tags and sequences of these represent half of the features for the binary classifier and all of the features for the relation phrase extractor. When this pre-processing step fails any processing done behind it will falter as a result.

6.1.3 Recurring patterns

There are patterns in the GHTorrent data that are hard to interpret, both for an unknowing machine and any human not being directly familiar with what caused the pattern. The pattern below (counted by the sentence detector as one whole sentence) occur multiple times, sometimes with other branch names (in this case *"trunk"*)

"–HG–
branch : trunk"

The branch names have been annotated as entities in these cases and in each case an author relation is wrongly labeled as true by Relex. Below are two cases of another pattern that recur several times in the test data.

"svn path=/trunk/mono/; revision=111467"

"svn path=/trunk/mcs/; revision=91662"

In these cases two entities are annotated one for the path (*"/trunk/mono/"* and *"/trunk/mcs/"*) and one for the revision id (*"111467"* and *"91662"*). In each case where this pattern occur an author relation is wrongly labeled as true for the revision id.

> *"git-svn-id: http://svn-commit.rubyonrails.org/rails/trunk@5893 5ecf4fe2-1ee6-0310-87b1-e25e094e27de"*

As a final problematic pattern the one above is in each case it occurs wrongly labeled with an author relation with *"http://svn-commit.rubyonrails.org/rails/trunk@5893"*.

Each occurrence of the types of patterns described above lowers the precision as it wrongly labels non-relations as relations. If such cases can be handled with human intervention or simple rules to filter them out that would definitely increase precision. This is however, nothing that Relex currently does and such capabilities would require careful mapping of the patterns occurring in the data. Moreover, as these are recurring patterns we assume they either belong to a third party software handling git commits or a standard rigorously enforced by a team on a project. With that dependency there is a risk that heuristics would fail with new or changed standards and with altered or new third party software.

It should be mentioned that there are other types of more semantically sound patterns which are successfully labeled by Relex. Below are three samples of one such pattern.

> *"Merge branch 'master' of github.com:akka/akka"*

> *"Merge pull request #370 from Memphiz/airplay"*

> *"Merge pull request #848 from SuprDewd/issue_82"*

Each of these contain two author relations and one entity relation. The entity relation and one of the author relations are false relations extracted by the relation candidate identificator. All of the relations are correctly labeled by Relex, despite the preprocessing failing. Contrary to the wrongly labeled patterns from above these patterns are semantically interpretable for a human. When that is not the case no supervised relation extractor can be faulted for mislabeling.

6.1.4 Language use

Most of the causes for wrong annotations seen so far can be attributed to the specific use of syntax and terminology in the GHTorrent data that differ from how well formed English sentences would be written. To extend that argument there is also a use of language that differs from how it would be done in the ConLL training data but which is hard to label as faulty English per se. Sentences such as the one below contribute greatly to low precision even though all the tokens are labeled with the correct part-of-speech.

> *"(mono_save_args): Use ARGSTORE instead of TEMPSTORE to handle soft float correctly."*

Here an entity relation has been annotated between *"mono_save_args"* and *"ARGSTORE"* with the relation phrase *"Use"*. There are however no less than four other relations wrongly classified as true.

1. Entity relation between *"ARGSTORE"* and *"TEMPSTORE"*
   The relation between *"ARGSTORE"* and *"TEMPSTORE"* is relative to the relation between *"mono_save_args"* and *"ARGSTORE"*. Thus, if n-ary or nested relations were being elicited this would be a valid relation.

2. Entity relation between *"mono_save_args"* and *"TEMPSTORE"*
   There is no explicit binary relation between these two entities. It is interesting to note however, that there would be an implicit relation between *"mono_save_args"* and *"TEMPSTORE"* with a relation phrase on the form *"not use"*

3. Author relation with *"ARGSTORE"*
   In this instance we do not have an annotation saying that the author is using *"ARGSTORE"* but instead that *"mono_save_args"* is.

4. Author relation with *"TEMPSTORE"*
   Similarly to the previous case, in this instance we do not think that the author is using *"TEMPSTORE"* but instead that *"mono_save_args"* is. Again if there would be a relation between author and *"TEMPSTORE"* the relation would be implicit with a relation phrase on the form *"not use"*.

This is more likely a case of the training data differing greatly from the GHTorrent domain. The commit message above contributes severely to low score while there are others which contribute more lightly.

> *"Start login shell (fixes #59 github issue) (per Austin Clements)"*

In the sentence above Relex incorrectly finds an entity relation between *"login shell"* and *"#59"*. This case also illustrate the domain specific language use in the GHTorrent data, with differing syntax and vocabulary compared to the more standard English language in the ConLL data.

6.2 Alternatives

So far in our discussion we have been pointing to different possibilities why Relex perform poorly on the GHTorrent data. It would therefore be interesting to revisit the design choices and evaluate how other approaches may aid relation extraction on this type of data.

The main problems have been a poor match between the system and the data. With the sentence detector, part-of-speech tagger, and binary classifier all failing due to mismatch between model and data it begs to question if the system should be using other models or rely on algorithms not reliant on models at all.

In the case of pre-processing for such annotations as sentences, tokens, and part-of-speech there are to our knowledge no models trained specifically for Git commit messages. The other models we have found are either very specific to their domain or generic to well formed English sentences, in which case they are as unfitted for the GHTorrent data as the ones used in this study. We therefore assess the probability of finding models suitable for these purposes as close to zero. Another option is training such models for the domain. This is not only time consuming and expensive but may also be futile. The way language use differ between projects, third party software, and even individuals may make it impossible for a system to find clear signals in training data. Remains does training models for specific projects or individuals which we dismiss based on cost and time consumption alone.

When it comes to the relation extraction itself both the binary classifier and the relation phrase extractor could benefit from models trained on hand-annotated, domain specific data. For the relation phrase extractor a machine learning algorithm such as conditional random fields would be needed. An issue with this approach is that features available for machine learning, functional on the domain specific data, would still need domain specific models. Another problem is difficulties for an annotator to comprehend what the commit message author is trying to convey when using syntax and terminology not know to the annotator, as is the case when language strays too far from passable English.

Other options are to use different features and algorithms, preferably such which are not sensitive to noisy data. A possibility would be less supervised machine learning algorithms, although these are generally less successful compared to supervised methods. There are middle ground with semi-supervised, self-supervised, and distantly supervised alternative which have shown previous success for NLP. The latter of these would be promising considering a Git repository is essentially a source of structure data potentially useful for distant supervision. Another option which have gotten much attention lately would be deep learning. Deep learning is a mainly unsupervised technique which finds patterns in raw data without the use of intermediary features.

6.3 Overall results

Even if we focus on the best of the results, the NP-NP model applied to finding author relations, the relations that are extracted are more incorrect ones than correct ones. Looking at the entity relations the same metric, precision, is well under 20 %. In all combinations of models and relation types a clear decline in both recall and precision is evident compared to the corresponding performance on the well formed English sentences from the ConLL data. The same kind of decline is present when applying the rule based relation phrase extractor which is very successful on the ConLL data with accuracy reaching 99 % but only reaching 65 % on the GHTorrent relations.

Revisiting the introduction and looking at our motives for this study we can recall that the output is candidate for use as decision support in industry and to gather data in research. The quantitative results from the experiments coupled with what we have

seen throughout the rest of our discussion there are clear indications that Relex in its current form is not ideal for the problem. Neither can a researcher or a decision maker in a software development process be bothered to trust data that is likely to be misleading.

Thus, judging only by the results themselves, Relex would seem like an unsuitable match to handle relation extraction on Git commit messages. As for the relation extraction and natural language field in general we have seen that models trained on data drawn from well formed English sentences does not perform satisfactory. This seems to be due to a domain specific language used including different syntax and different vocabulary that causes the analysis to fail. In order to use the techniques and methods offered by NLP domain specific models need to be trained for each individual component. However, due to variations in syntax and terminology use between projects and individuals not even this is sure to produce a functional model. Other options include resorting to lower degrees of supervision which traditionally perform worse than their fully supervised counterparts. We would say that in its current state and without specific tailoring to the domain, relation extraction is not ready to be applied to the task of extraction relations between entities in Git commit messages.

7

Threats to validity

7.1 Conclusion validity

The results produced in this work are done so using our own software. Using the pipeline analogy we use open source software to help set up the architecture and as boilerplate code for the evaluations. Moreover, open source software is used for all of the pre-processing pipes. For the pipes belonging to the relation extraction itself we have made our own implementations. Faulty implementations of any of the components, our own or open source, would erode the integrity of the results. For this reason we maintain complete transparency and all of the software used in this thesis can be found for inspection through our software repository[1].

No significance tests are made on the difference between results of different tests made in the study. Additionally, the number of entity relations is rather small. However, the differences in performance between applying Relex to ConLL data and GHTorrent data is big and the number of samples is large. Therefore, the qualitative conclusions made in the thesis should be valid.

7.2 Internal validity

The entire study is made by the author alone including annotations of data which are very open for interpretation. The entire annotation project is available online [2] and two lists of guidelines for annotation of entities and relations are provided as appendices A and B respectively. The study would definitely be strengthen by more annotators and an inter-judge reliability measurement for annotations.

[1] `https://github.com/Rikard-Andersson/relex`

[2] `https://github.com/Rikard-Andersson/GHTorrent-Brat`

7.3 External validity

Only public git repositories from GitHub is included in the study and this selection is further limited by only the 90 most popular projects. We can however generalize the invalidation of the methods to the population based on a case. If it does not work on a case it does by definition not work on the entire population.

The experiments are made with one system which is a weakness and a threat to external validity since results may not generalize to the entire field. It is mitigated by Relex being a system that is inspired with properties from a large number of other systems in the field. They all carry similar properties. We also provide an analysis of results which is intended to extend to the general case. This analysis is however qualitative. Any conclusions to the generic case must be considered with this in mind.

7.4 Construct validity

In the case where Relex is not a good representative of a relation extraction system the benchmarking of its performance between different domains may not be representative of what we are trying to achieve. Not being able to compare Relex to any similar systems is hence a threat to validity.

8

Conclusion

Relation extraction is used as a solution to structure unstructured data in the form of free text. One source of such data is software repositories but the solution space of relation extraction remains largely untested for this type of data. This thesis has therefore investigated how well supervised relation extraction systems for natural language can be applied to git commit messages.

In order to do this we have conducted an experiment where git commit messages were extracted from public repositories on GitHub. These commit messages were then annotated with entities and relations of interest and fed to a relation extraction system. The results were evaluated and discussed both quantitatively and qualitatively.

The results from the experiments showed that the system had severe performance issues where the best case scenario involved a mere 25 % precision and 55 % recall. It was shown that this is a steep decline from the systems performance on well formed English sentences from newswire data. This decline can be attributed to the models upon which both the relation extraction system relies but also the models of pre-processing used. These models does not function well on the noisy syntax and grammar of git commit messages which is further complicated by a vocabulary specific to the domain.

From this we conclude that supervised relation extraction out-of-the-box can not be used on git commit messages due to lack of domain specific models.

The learnings from this work should be interpreted in the light of the effort that was made to achieve them. In this work existing models have been applied without much alterations. As discussed in Section 1, Life sciences have an entire research field within which use of existing natural language processing technologies are explored and new adaptations are made. This success could certainly be repeated for Software Engineering and data such as git repositories if the effort is made to build domain specific adaptations of techniques, tools, and models for natural language processing.

Other options include exploring the use of different levels of supervision in the machine learning algorithms. There is reason to believe that distantly supervised models

would work well since the text data of git commit messages is closely coupled with a structured source of data inherent in the git repositories themselves. Moreover, semi-supervised and self-supervised algorithms are popular within open information extraction indicating such a direction as a possible improvement. Lastly, unsupervised algorithms and deep learning with deep neural networks are getting a lot of attention lately, an area which could prove useful for application discussed in this thesis.

Bibliography

[1] Eugene Agichtein and Luis Gravano. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the Fifth ACM Conference on Digital Libraries*, DL '00, pages 85–94. ACM, 2000.

[2] Nasir Ali, Yann-Gael Gueneuc, and Giuliano Antoniol. Trustrace: Mining software repositories to improve the accuracy of requirement traceability links. *Software Engineering, IEEE Transactions on*, 39(5):725–741, 2013.

[3] Rikard Andersson. Thesis release. Aug 2014. doi: {10.5281/zenodo.11200}. URL `http://dx.doi.org/10.5281/zenodo.11200`.

[4] Michele Banko and Oren Etzioni. The tradeoffs between open and traditional relation extraction. In *Proceedings of ACL-08: HLT*, pages 28–36, Columbus, Ohio, June 2008. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P/P08/P08-1004`.

[5] Michele Banko, Michael J. Cafarella, Stephen Soderland, Matt Broadhead, and Oren Etzioni. Open information extraction from the web. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, pages 2670–2676, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc. URL `http://dl.acm.org/citation.cfm?id=1625275.1625705`.

[6] Sergey Brin. Extracting patterns and relations from the world wide web. In *The World Wide Web and Databases*, volume 1590 of *Lecture Notes in Computer Science*, pages 172–183. Springer Berlin Heidelberg, 1999. URL `http://dx.doi.org/10.1007/10704656_11`.

[7] Krishna Kumar Chaturvedi, V B Singh, and Prashast Singh. Tools in mining software repositories. In *Computational Science and Its Applications (ICCSA), 2013 13th International Conference on*, pages 89–98, June 2013.

[8] Tse-Hsun Chen, Stephen W Thomas, Meiyappan Nagappan, and Ahmed E Hassan. Explaining software defects using topic models. In *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, pages 189–198, June 2012.

[9] Janara Christensen, Mausam, Stephen Soderland, and Oren Etzioni. An analysis of open information extraction based on semantic role labeling. In *Proceedings of the sixth international conference on Knowledge capture*, pages 113–120. ACM, 2011.

[10] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[11] Aron Culotta and Jeffrey Sorensen. Dependency tree kernels for relation extraction. In *Proceedings of the 42Nd Annual Meeting on Association for Computational Linguistics*, ACL '04. Association for Computational Linguistics, 2004.

[12] Aron Culotta, Andrew McCallum, and Jonathan Betz. Integrating probabilistic extraction models and data mining to discover relations and patterns in text. In *Proceedings of the Main Conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, HLT-NAACL '06, pages 296–303. Association for Computational Linguistics, 2006.

[13] Oren Etzioni, Anthony Fader, Janara Christensen, Stephen Soderland, and Mausam Mausam. Open information extraction: The second generation. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume One*, pages 3–10. AAAI Press, 2011.

[14] Peter Exner and Pierre Nugues. Entity extraction: From unstructured text to dbpedia rdf triples. In *Proceedings of the Web of Linked Entities Workshop in conjuction with the 11th International Semantic Web Conference*, pages 58–69, 2012.

[15] Georgios Gousios. The ghtorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 233–236, Piscataway, NJ, USA, 2013. IEEE Press. ISBN 978-1-4673-2936-1. URL `http://dl.acm.org/citation.cfm?id=2487085.2487132`.

[16] Ralph Grishman and Beth Sundheim. Message understanding conference-6: A brief history. In *Proceedings of the 16th Conference on Computational Linguistics - Volume 1*, volume 96, pages 466–471, 1996.

[17] Maria Halkidi, Diomidis Spinellis, George Tsatsaronis, and Michalis Vazirgiannis. Data mining in software engineering. *Intelligent Data Analysis*, 15(3):413–441, 2011.

[18] Ahmed E. Hassan and Tao Xie. Software intelligence: The future of mining software engineering data. In *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, FoSER '10, pages 161–166. ACM, 2010.

[19] Chen Jinxiu, Ji Donghong, TanChew Lim, and Niu Zhengyu. Automatic relation extraction with model order selection and discriminative label identification. In Robert Dale, Kam-Fai Wong, Jian Su, and OiYee Kwong, editors, *Natural Language Processing – IJCNLP 2005*, volume 3651 of *Lecture Notes in Computer Science*, pages 390–401. Springer Berlin Heidelberg, 2005.

[20] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[21] Julie Medero, Kazuaki Maeda, Stephanie Strassel, and Christopher Walker. An efficient approach to gold-standard annotation: Decision points for complex tasks. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation*, volume 6, pages 2463–2466, May 2006.

[22] Yuval Merhav, Filipe Mesquita, Denilson Barbosa, Wai Gen Yee, and Ophir Frieder. Extracting information networks from the blogosphere. *ACM Transactions on the Web (TWEB)*, 6(3):11, 2012.

[23] Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 1003–1011. Association for Computational Linguistics, 2009.

[24] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26, 2007.

[25] Johan Natt och Dag, Björn Regnell, Vincenzo Gervasi, and Sjaak Brinkkemper. A linguistic-engineering approach to large-scale requirements management. *Software, IEEE*, 22(1):32–39, Jan 2005.

[26] Joakim Nivre, Johan Hall, and Jens Nilsson. Maltparser: A data-driven parser-generator for dependency parsing. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation*, volume 6, pages 2216–2219, May 2006.

[27] Martha Palmer, Daniel Gildea, and Paul Kingsbury. The proposition bank: An annotated corpus of semantic roles. *Computational linguistics*, 31(1):71–106, 2005.

[28] Daniel Pletea, Bogdan Vasilescu, and Alexander Serebrenik. Security and emotion: Sentiment analysis of security discussions on github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 348–351. ACM, 2014.

[29] Sameer Pradhan, Lance Ramshaw, Mitchell Marcus, Martha Palmer, Ralph Weischedel, and Nianwen Xue. Conll-2011 shared task: Modeling unrestricted coreference in ontonotes. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task*, CONLL Shared Task '11, pages 1–27, Stroudsburg, PA, USA, June 2011. Association for Computational Linguistics. ISBN 9781937284084.

[30] Benjamin Rosenfeld and Ronen Feldman. Clustering for unsupervised relation identification. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 411–418. ACM, 2007.

[31] Per Runeson, Magnus Alexandersson, and Oskar Nyholm. Detection of duplicate defect reports using natural language processing. In *Software Engineering, 2007. ICSE 2007. 29th International Conference on*, pages 499–510, May 2007.

[32] Michael Schmitz, Robert Bart, Stephen Soderland, Oren Etzioni, et al. Open language learning for information extraction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 523–534. Association for Computational Linguistics, 2012.

[33] Pontus Stenetorp, Sampo Pyysalo, Goran Topić, Tomoko Ohta, Sophia Ananiadou, and Jun'ichi Tsujii. Brat: A web-based tool for nlp-assisted text annotation. In *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*, EACL '12, pages 102–107, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[34] Mihai Surdeanu, Julie Tibshirani, Ramesh Nallapati, and Christopher D. Manning. Multi-instance multi-label learning for relation extraction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL '12, pages 455–465, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[35] Ferdian Thung, Tegawende F. Bissyande, David Lo, and Jiang Lingxiao. Network structure of social coding in github. In *Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on*, pages 323–326, March 2013.

[36] Jason T. Tsay, Laura Dabbish, and James Herbsleb. Social media and success in open source projects. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work Companion*, CSCW '12, pages 223–226, New York, NY, USA, 2012. ACM.

[37] Xiaoyin Wang, Lu Zhang, Tao Xie, John Anvik, and Jiasu Sun. An approach to detecting duplicate bug reports using natural language and execution information. In *Proceedings of the 30th International Conference on Software Engineering*, ICSE '08, pages 461–470, New York, NY, USA, 2008. ACM.

[38] Fei Wu and Daniel S Weld. Open information extraction using wikipedia. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 118–127. Association for Computational Linguistics, 2010.

[39] Leon Li Wu, Boyi Xie, Gail E Kaiser, and Rebecca Passonneau. Bugminer: Software reliability analysis via data mining of bug reports. In *Proceedings of the 23rd International Conference on Software Engineering and Knowledge Engineering (SEKE)*, pages 95–100. Department of Computer Science, Columbia University, July 2011.

[40] Ying Xu, Mi-Young Kim, Kevin Quinn, Randy Goebel, and Denilson Barbosa. Open information extraction with tree kernels. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 868–877, June 2013.

[41] Limin Yao, Aria Haghighi, Sebastian Riedel, and Andrew McCallum. Structured relation discovery using generative models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1456–1466. Association for Computational Linguistics, 2011.

A

List of entity types

Files A file in a file system

Directories Any directory in a file system

Classes Referring to the object oriented programming concept of a class

Object An instantiation of a class

Methods Referring to methods and functions in programming

Variables Referring to the concept in programming

Parameters Very similar to variables but often used with reference to such provided in method calls.

URL A URL

Resources A generic type to collect other types of resources used in development or in code

Issues An issue, a bug report, a bullet point on a todo list

Commits The git concept of a commit

Branches The git concept of a branch

Pull request A type of issue on github

Products Products names such as iOS, iPhone, Android, MySQL

Technical concept/Tech Misc. Entities which do not fit under other to catch a technical concept

Version A version name of a product

Tests A test case

B

Annotation guidelines

1. Relations have an explicitly stated relation phrase that occur between the two entities
2. The relation must occur directly between two entities if there is a mediating noun between, leave the relation.
   (a) <E1> done something to something that is related to <E2>
   (b) <E1> done something to something because of something in <E2>
   (c) “added support for iPhone” - Not a relations since the author relation is not directly between iPhone and the author but rather between the author and support.
   (d) “Dropped support for PostgreSQL < 8.4” - Same as above
3. Author relation
   (a) Instance where the author (not mentioned in the commit message) have done something to an entity (mentioned in the commit message).
   (b) Instance where a commit or the application of a commit (not mentioned in the commit message) have done something to an entity (mentioned in the commit message)
4. If something is ambiguous (you are not sure about the actual relation) leave it out
5. Ignore chunks of code

C

ConLL 2011 shared task data example

In Table C.1 is an excerpt from the tables provided in the ConLL 2011 shared task as training and testing data. The table is incomplete and originally provides additional data about each sentence which is left out for simplicity. Each table can contain additional columns for ProBank-style relations where the example in Table C.1 only has two.

Token	**Part-of-speech**	**Named entity**	**Relations**	
Rear	NNP	*	(ARG0*	(ARG1*
Admiral	NNP	*	*	*
Mark	NNP	(PERSON*	*	*
Fitzgerald	NNP	*)	*)	*
,	,	*	*	*
who	WP	*	(R-ARG0*)	*
visited	VBD	*	(V*)	*
the	DT	*	(ARG1*	*
Cole	NNP	(PRODUCT)	*)	*)
,	,	*	*	*
was	VBD	*	*	*
stunned	VBN	*	*	(V*)
by	IN	*	*	(ARG0*
the	DT	*	*	*
damage	NN	*	*	*)
.	.	*	*	*

Table C.1: Excerpt from ConLL 2011 shared task data

D

Relex binary task performance on ConLL data

Table D.1, D.2, and D.3 each present data for how the SVM classifier in Relex score for the binary task when tested on all three subsets of the ConLL data NE-NE, NE-NP, and NP-NP respectively. For the test data all pre-processing annotations, including the named entity tags, were derived from the gold annotations in the ConLL data.

		Predicted		
		True	False	Total
Actual	True	229	50	279
	False	33	622	655
	Total	262	672	

Recall	0.821
Precision	0.874
F-measure	0.847

Table D.1: NE-NE performance on ConLL NE-NE testing data

		Predicted		
		True	False	Total
Actual	True	152	29	181
	False	52	348	400
	Total	204	377	

Recall	0.840
Precision	0.745
F-measure	0.790

Table D.2: NE-NP performance on ConLL NE-NP testing data

		Predicted		
		True	False	Total
Actual	True	365	86	451
	False	95	797	892
	Total	460	883	

Recall	0.809
Precision	0.793
F-measure	0.801

Table D.3: NP-NP performance on ConLL NP-NP testing data

E

Relex triple task performance on ConLL data

The triple task (relation phrase extraction) was tested in separate runs for each of the ConLL subsets NE-NE, NE-NP, and NP-NP. During these tests only true relations were considered, hence the relation phrase extractor was the only part of the system being tested in complete isolation. Since the relation phrase extractor itself need no training data the triple task use the larger training set from the ConLL data to test on with results seen in Table E.1.

Dataset	Accuracy
NE-NE	0.927
NE-NP	0.998
NP-NP	0.998

Table E.1: Performance of relation phrase extractor per dataset

F

List of projects in GHTorrent

Below follows a list of projects that are used to draw data for the GHTorrent MSR 2014 dataset available at `http://ghtorrent.org/msr14.html`. The projects are selected from the top-10 starred projects for the nine most popular programming language. Each project listed below is listed as [user]/[project] so in order to visit the projects page just type `https://github.com/[user]/[project]`

- akka/akka
- hadley/devtools
- johnmyleswhite/ProjectTemplate
- mavam /stat-cookbook
- facebook/hiphop-php
- yihui/knitr
- rstudio/shiny
- facebook/folly
- mongodb/mongo
- TTimo/doom3.gpl
- ariya/phantomjs
- TrinityCore/TrinityCore
- mangos/MaNGOS

- bitcoin/bitcoin
- keithw/mosh
- xbmc/xbmc
- joyent/http-parser
- kr/beanstalkd
- antirez/redis
- liuliu/ccv
- memcached/memcached
- openframeworks/openFrameworks
- libgit2/libgit2
- vmg/redcarpet
- joyent /libuv
- SignalR/SignalR
- hbons/SparkleShare
- moxiecode/plupload
- mono/mono
- NancyFx/Nancy
- ServiceStack/ServiceStack
- AutoMapper/AutoMapper
- restsharp/RestSharp
- ravendb/ravendb
- SamSaffron/MiniProfiler
- nathanmarz/storm
- elasticsearch/elasticsearch
- JakeWharton/ActionBarSherlock
- facebook/facebook-android-sdk
- clojure/clojure

- Bukkit/CraftBukkit
- netty/netty
- github/android
- joyent/node
- jquery/jquery
- h5bp/html5-boilerplate
- bartaz/impress.js
- mbostock/d3
- harvesthq/chosen
- FortAwesome/Font-Awesome
- mrdoob/three.js
- zurb/foundation
- symfony/symfony
- EllisLab/CodeIgniter
- facebook/php-sdk
- zendframework/zf2
- cakephp/cakephp
- ginatrapani/ThinkUp
- sebastianbergmann/phpunit
- codeguy/Slim
- django/django
- facebook/tornado
- jkbr/httpie
- mitsuhiko/flask
- kennethreitz/requests
- xphere-forks/symfony
- reddit/reddit

- boto/boto
- django-debug-toolbar/django-debug-toolbar
- midgetspy/Sick-Beard
- divio/django-cms
- rails/rails
- mxcl/homebrew
- mojombo/jekyll
- gitlabhq/gitlabhq
- diaspora/diaspora
- plataformatec/devise
- joshuaclayton/blueprint-css
- imathis/octopress
- vinc/vinc.cc
- thoughtbot/paperclip
- chriseppstein/compass
- twitter/finagle
- robey/kestrel
- twitter/flockdb
- twitter/gizzard
- sbt/sbt
- scala/scala
- scalatra/scalatra
- twitter/zipkin