HYENA-live: Fine-Grained Online Entity Type Classification from Natural-language Text

Mohamed Amir Yosef[1] **Sandro Bauer**[2] **Johannes Hoffart**[1]
Marc Spaniol[1] **Gerhard Weikum**[1]
(1) Max-Planck-Institut für Informatik, Saarbrücken, Germany
(2) Computer Laboratory, University of Cambridge, UK
{mamir|jhoffart|mspaniol|weikum}@mpi-inf.mpg.de
sandro.bauer@cl.cam.ac.uk

Abstract

Recent research has shown progress in achieving high-quality, very fine-grained type classification in hierarchical taxonomies. Within such a multi-level type hierarchy with several hundreds of types at different levels, many entities naturally belong to multiple types. In order to achieve high-precision in type classification, current approaches are either limited to certain domains or require time consuming multistage computations. As a consequence, existing systems are incapable of performing ad-hoc type classification on arbitrary input texts. In this demo, we present a novel Web-based tool that is able to perform domain independent entity type classification under real time conditions. Thanks to its efficient implementation and compacted feature representation, the system is able to process text inputs on-the-fly while still achieving equally high precision as leading state-of-the-art implementations. Our system offers an online interface where natural-language text can be inserted, which returns semantic type labels for entity mentions. Further more, the user interface allows users to explore the assigned types by visualizing and navigating along the type-hierarchy.

1 Introduction

Motivation

Web contents such as news, blogs and other social media are full of named entities. Each entity belongs to one or more *semantic types* associated with it. For instance, an entity such as *Bob Dylan* should be assigned the types `Singer`, `Musician`, `Poet`, etc., and also the corresponding supertype(s) (hypernyms) in a type hierarchy, in this case `Person`. Such fine-grained typing of entities in texts can be a great asset for various NLP tasks including semantic role labeling, sense disambiguation and named entity disambiguation (NED). For instance, noun phrases such as “songwriter Dylan”, “Google founder Page”, or “rock legend Page” can be easily mapped to the entities Bob Dylan, Larry Page, and Jimmy Page if their respective types `Singer`, `BusinessPerson`, and `Guitarist` are available (cf. Figure 1 for an illustrative example).

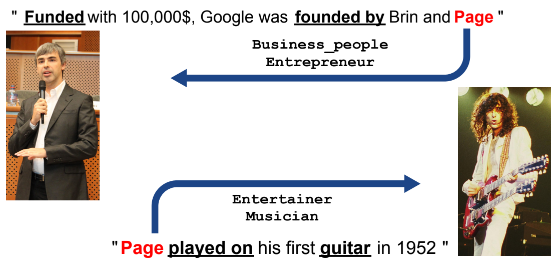
" Funded with 100,000$, Google was founded by Brin and Page "
Business_people
Entrepreneur
Entertainer
Musician
" Page played on his first guitar in 1952 "

Figure 1: Fine-grained entity type classification

Problem Statement

Type classification is not only be based on hierarchical sub-type relationships (e.g. `Musician isA Person`), but also has to do on multi-labeling. Within a very fine-grained type hierarchy, many entities naturally belong to multiple types. For example, a guitarist is also a musician and a person, but may also be a singer, an actor, or even a politician. Consequently, entities should not only be assigned the most (fine-grained) label associated to them, but with all labels relevant to them. So we face a *hierarchical multi-label classification* problem (Tsoumakas et al., 2012).

Contribution

This paper introduces *HYENA-live*, which allows an on-the-fly computation of semantic types for entity mentions, based on a multi-level type hierarchy. Our approach uses a suite of features for a given entity mention, such as neighboring words and bigrams, part-of-speech tags, and also phrases from a large gazetteer derived from state-of-the-art knowledge bases. In order to perform "live" entity type classification based on ad-hoc text inputs, several performance optimizations have been undertaken to operate under real-time conditions.

2 Entity Type Classification Systems

State-of-the-art tools for named entity recognition such as the Stanford NER Tagger (Finkel et al., 2005) compute semantic tags only for a small set of coarse-grained types: `Person`, `Location`, and `Organization` (plus tags for non-entity phrases of type time, money, percent, and date). However, we are not aware of any online tool that performs *fine-grained typing* of entity mentions. The most common workaround to perform entity classification is a two-stage process: in first applying an online tool for Named-Entity Disambiguation (NED), such as DBpedia Spotlight (Mendes et al., 2011) or AIDA (Yosef et al., 2011; Hoffart et al., 2011), in order to map the mentions onto canonical entities and subsequently query the knowledge base for their types. In fact, (Ling and Weld, 2012) followed this approach when comparing their entity classification system results against those obtained by an adoption of the Illinois' Named-Entity Linking system (NEL) (Ratinov et al., 2011) and reached the conclusion that while NEL performed decently for prominent entities, it could not scale to cover long tail ones. Specifically, entity typing via NED has three major drawbacks:

1. NED is an inherently hard problem, especially with highly ambiguous mentions. As a consequence, accurate NED systems come at a high computation costs.
2. NED only works for those mentions that correspond to a canonical entity within a knowledge base. However, this fails for all out-of-knowledge-base entities like unregistered persons, start-up companies, etc.
3. NED heavily depends on the quality of the underlying knowledge base. Yet, only very few knowledge bases have comprehensive class labeling of entities. Even more, in the best case, coverage drops sharply for relatively uncommon entities.

We decided to adopt one of the existing approaches to make it suitable for online querying. We considered five systems. In the rest of this section we will briefly describe each of them.

(Fleischman and Hovy, 2002) is one of the earliest approaches to perform entity classification into subtypes of `PERSON`. They developed a decision-tree classifier based on contextual features that can be automatically extracted from the text. In order to account for scarcity of labeled training data, they tapped on WordNet synonyms to achieve higher coverage. While their approach is fundamentally suitable, their type system is very restricted. In order to account for more fine-grained classes, more features need to be added to their feature set.

(Ekbal et al., 2010) considered 141 subtypes of WordNet class `PERSON` and developed a maximum entropy classifier exploiting the words surrounding the mentions together with their POS tags and other contextual features. Their type hierarchy is fine-grained, but still limited to sub classes of `PERSON`. In addition, their experimental results have been flagged as non-reproducible in the ACL Anthology.

(Altaf ur Rahman and Ng, 2010) considered a two-level type hierarchy consisting of 29 top-level classes and a total of 92 sub-classes. These include many non-entity types such as date, time, percent, money, quantity, ordinal, cardinal, etc. They incorporated a hierarchical classifier using a rich feature set and made use of WordNet sense tagging. However, the latter requires human interception, which is not suitable for ad-hoc processing of out-of-domain texts.

(Ling and Weld, 2012) developed FIGER, which classifies entity mentions onto a two-level taxonomy based on the Freebase knowledge base (Bollacker et al., 2008). This results in a two-level hierarchy with top-level topics and 112 types. They trained a CRF for the joint task of recognizing entity mentions and inferring type tags. Although they handle multi-label assignment, their test data is sparse. Many classes are absent and plenty of instances come with only a single label (e.g. 216 of the 562 entities were of type `PERSON` without subtypes). Further, their results are instance based, which does not guarantee that the quality of their system will be reproducible for all the 112 types in their taxonomy.

(Yosef et al., 2012) is the most recent work in multi-label type classification. The HYENA system incorporates a large hierarchy of 505 classes organized under 5 top level classes, with 100 descendant classes under each of them. The hierarchy reaches a depth of up to 9 levels in some parts. The system is based on an SVM classifier using a comprehensive set of features and provides results for all classes of a large data set. In their experiments the superiority of the system in terms of precision and recall has been shown. However, the main drawback of HYENA comes from its large hierarchy and the extensive set of features extracted from the fairly large training corpus it requires. As a result, on-the-fly type classification with HYENA is impossible in its current implementation.

We decided to build on top of HYENA system by spotting the bottlenecks in the architecture and modifying it accordingly to be suitable for online querying. In Section 3 we explain in details HYENA's type taxonomy and their feature portfolio. Later on, we explain the engineering undertaken in order to develop the on-the-fly type classification system HYENA-live (cf. Section 4).

3 Type Hierarchy and Feature Set

3.1 Fine-grained Taxonomy

The type system is an automatically gathered fine-grained taxonomy of 505 classes. The classes are organized under 5 top level classes, with 100 descendant classes under each. The YAGO knowledge base (Hoffart et al., 2013) is selected to derive the taxonomy from because of its highly precise classification of entities into WordNet classes, which is a result of the accurate mapping YAGO has from Wikipedia Categories to WordNet synsets.

We start with five top classes namely `PERSON`, `LOCATION`, `ORGANIZATION`, `EVENT` and `ARTIFACT`. Under each top class, the most 100 prominent descendant classes are picked. Prominence is estimated by the number of YAGO entities tagged with this class. This results in a very-fine grained taxonomy of 505 types, represented as a directed acyclic graph with 9 levels in its deepest parts. While the classes are picked from the YAGO type system, the approach is generic and can be applied to derive type taxonomies from other knowledge bases such as Freebase or DBpedia (Auer et al., 2007) as in (Ling and Weld, 2012).

3.2 Feature Set

For the sake of generality and applicability to arbitrary text, we opted for features that can be automatically extracted from the input text without any human interaction, or manual annotation. The extracted features fall under five categories, which we briefly explain in the rest of this section.

Mention String
We derive four features from the entity mention string. The mention string itself, a noun phrase consisting of one or more consecutive words. The other three features are unigrams, bigrams, and trigrams that overlap with the mention string.

Sentence Surrounding Mention
We also exploit a bounded-size window around the mention to extract four features: all unigrams, bigrams, and trigrams. Two versions of those features are extracted, one to account for the occurrence of those tokens around the mention, and another to account for the position at which they occurred with respect to the mention (before or after). In addition, unigrams are also included with their absolute distance ignoring whether before of after the mention. Our demo is using a conservative threshold for the size of the window which is three tokens on each side of the mention.

Mention Paragraph
We also leverage the entire paragraph of the mention. This gives additional topical cues about the mention type (e.g., if the paragraph is about a music concert, this is a cue for mapping people names to musician types). We create three features here: unigrams, bigrams, and trigrams without including any distance information. In our demo, we extract those features from a bounded window of size 2000 characters before and after the mention.

Grammatical Features
We exploit the semantics of the text by extracting four features. First, we use part-of-speech tags of the tokens in a size-bounded window around the mention in distance and absolute distance versions. Second and third, we create a feature for the first occurrence of a "he" or "she" pronoun in the same sentence and in the subsequent sentence following the mention, along with the distance to the mention. Finally, we use the closest verb-preposition pair preceding the mention as another feature.

Gazetteer Features
We leverage YAGO2 knowledge base even further by building a type-specific gazetteer of words oc-

# of articles	50,000
# of instances (all types)	1,613,340
# of location instances	489,003 (30%)
# of person instances	426,467 (26.4%)
# of organization instances	219,716 (13.6%)
# of artifact instances	204,802 (12.7%)
# of event instances	176,549 (10.9%)
# instances in 1 top-level class	1,131,994 (70.2%)
# instances in 2 top-level classes	182,508 (11.3%)
# instances in more than 2 top-level classes	6,492 (0.4%)
# instances not in any class	292,346 (18.1%)

Table 1: Properties of the labeled data used for training HYENA-live

curring in the names of the entities of that type. YAGO2 knowledge base comes with an extensive dictionary of name-entity pairs extracted from Wikipedia redirects and link-anchor texts. We construct, for each type, a binary feature that indicates if the mention contains a word occurring in this type's gazetteer. Note that this is a fully automated feature construction, and it does by no means determine the mention type(s) already, as most words occur in the gazetteers of many different types. For example, "Alice" occurs in virtually every subclass of Person but also in city names like "Alice Springs" and other locations, as well as in songs, movies, and other products or organizations.

4 System Implementation

4.1 Overview

As described in Section 3, HYENA classifies mentions of named entities onto a hierarchy of 505 types using large set of features. A random subset of the English Wikipedia has been used for training HYENA. By exploiting Wikipedia anchor links, mentions of named entities are automatically disambiguated to their correct entities. Each Wikipedia named entity has a corresponding YAGO entity labeled with an accurate set of types, and hence we effortlessly obtain a huge training data set (cf. data properties in Table 1).

We build type-specific classifiers using the SVM software LIBLINEAR (cf. `http://liblinear.bwaldvogel.de/`). Each model comes with a comprehensive feature set. While larger models (with more features) improve the accuracy, they significantly affect the applicability of the system. A single model file occupies around 150MB disk space leading to a total of 84.7GB for all models. As a consequence, there is a substantial setup time to load all models in memory and a high-memory server (48 cores with 512GB of RAM) is required for computation. An analysis showed that each single feature contributes to the overall performance of HYENA, but only a tiny subset of all features is relevant for a single classifier. Therefore, most of the models are extremely sparse.

4.2 Sparse Models Representation

There are several workarounds applicable to batch mode operations, e.g. by performing classifications per level only. However, this is not an option for on-the-fly computations. For that reason we opted for a sparse-model representation.

LIBLINEAR model files are normalized textual files: a header (data about the model and the total number of features), followed by listing the weights assigned to each feature (line number indicates the feature ID). Each model file has been post-processed to produce 2 files:

- A compacted model file containing only features of non-zero weights. Its header reflects the reduced number of features.
- A meta-data file. It maps the new features IDs to the original feature IDs.

Due to the observed sparsity in the model files, particularly at deeper levels, there is a significant decrease in disk space consumption for the compacted model files and hence in the memory requirements.

4.3 Sparse Models Classification

By switching to the sparse model representation the architecture of the whole system is affected. In particular, modified versions of feature vectors need to be generated for each classifier; this is because

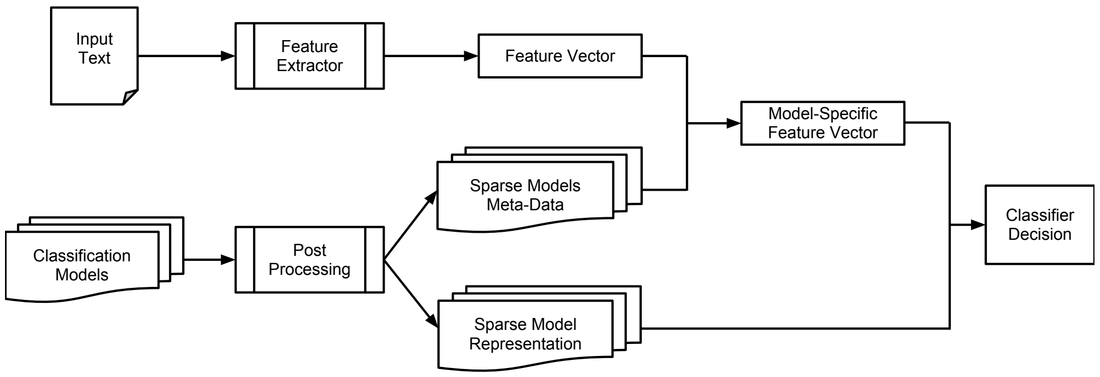

Figure 2: Modified system architecture designed for handling sparse models

a lot of features have been omitted from specific classifiers (those with zero weights). Consequently, the feature IDs need to be mapped to the new feature space of each classifier. The conceptual design of the new architecture is illustrated in Figure 4.2.

5 Demo Presentation

HYENA-live has been fully implemented as a Web application. Figure 5 shows the user interface of HYENA-live in a Web browser:

1) On top, there is a panel where a user can input any text, e.g. by copy-and-paste from news articles. We employ the Stanford NER Tagger to identify noun phrases as candidates of entity mentions. Alternatively, users can flag entity mentions by double brackets (e.g. "Harry is the opponent of [[you know who]]"). For the sake of simplicity, detected entity mentions by HYENA-live are highlighted in yellow. Each mention is clickable to study its type classification results.
2) The output of type classification is shown inside a tabbed widget. Each tab corresponds to a detected mention by the system and tabs are sorted by the order of occurrence in the input text. To open a tab, the tab header or the corresponding mention in the input area needs to be clicked.
3) The type classification of a mention is shown as a color-coded interactive tree. While the original type hierarchy is a directed acyclic graph, for the ease of navigation the classification output has been converted into a tree. In order to do so, nodes that belong to more than a parent have been duplicated. There are three different types of nodes:
   - Green Nodes: referring to a class that has been accepted by the classifier. These nodes can be further expanded in order to check which sub-classes have been accepted or rejected by HYENA-live.
   - Red Nodes: corresponding to a class that was rejected by the classifier, and hence HYENA-live did not traverse deeper to test its sub-classes.
   - White Nodes: matching classes that have not been tested. These nodes are either known upfront (e.g. `ENTITY`) or their super class was rejected by the system.

   It is worth noting that HYENA-live automatically adjusts the layouting so that as much as possible of the hierarchy is shown to the user. For the sake of explorability, this is being dynamically adjusted once the user decides to navigate along a certain (child-)node.

The system is available online at:
`d5gate.ag5.mpi-sb.mpg.de/webhyena/`.
The data transfer between the client and the server is done via JSON objects. Hence, we also provide HYENA-live as a JSON compliant entity classification Web-service. As a result, the back-end becomes easily interchangeable (e.g. by a different classification technique or a different type taxonomy) with minimum modifications required on the user interface side.

Acknowledgments

This work is supported by the 7^{th} Framework IST programme of the European Union through the focused research project (STREP) on Longitudinal Analytics of Web Archive data (LAWA) under contract no. 258105.

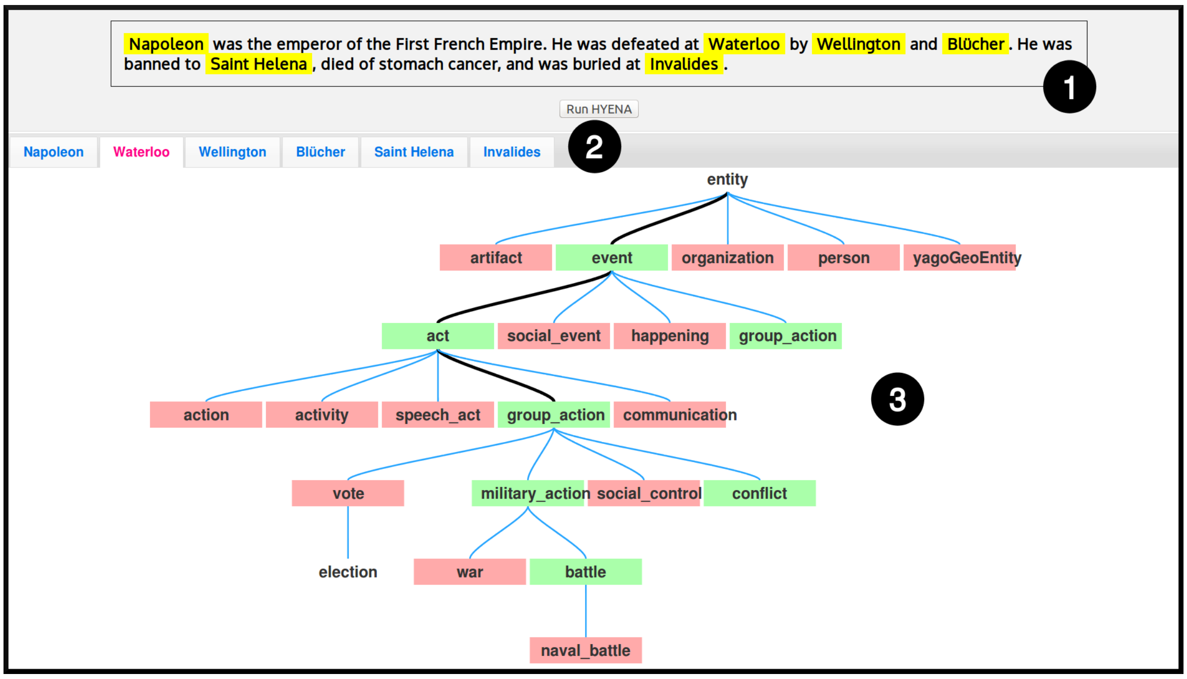
Napoleon was the emperor of the First French Empire. He was defeated at Waterloo by Wellington and Blücher. He was banned to Saint Helena, died of stomach cancer, and was buried at Invalides.

Figure 3: Interactively exploring the types of the "Battle of Waterloo" in the HYENA-live interface

References

Md. Altaf ur Rahman and Vincent Ng. 2010. Inducing fine-grained semantic classes via hierarchical and collective classification. In *COLING*, pages 931–939.

Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, and Zachary Ives. 2007. Dbpedia: A nucleus for a web of open data. In *ISWC*, pages 11–15. Springer.

Kurt D. Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*, pages 1247–1250.

Asif Ekbal, Eva Sourjikova, Anette Frank, and Simone P. Ponzetto. 2010. Assessing the challenge of fine-grained named entity recognition and classification. In *Named Entities Workshop*, pages 93–101.

Jenny Rose Finkel, Trond Grenager, and Christopher Manning. 2005. Incorporating non-local information into information extraction systems by gibbs sampling. In *ACL*, pages 363–370.

Michael Fleischman and Eduard Hovy. 2002. Fine grained classification of named entities. In *COLING*, pages 1–7.

Johannes Hoffart, Mohamed Amir Yosef, Ilaria Bordino, Hagen Fürstenau, Manfred Pinkal, Marc Spaniol, Bilyana Taneva, Stefan Thater, and Gerhard Weikum. 2011. Robust disambiguation of named entities in text. In *EMNLP*, pages 782–792.

Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, and Gerhard Weikum. 2013. YAGO2: A spatially and temporally enhanced knowledge base from wikipedia. *Artificial Intelligence*, 194(0):28 – 61.

Xiao Ling and Daniel S. Weld. 2012. Fine-grained entity recognition. In *AAAI*, pages 94–100.

Pablo N. Mendes, Max Jakob, Andrés García-Silva, and Christian Bizer. 2011. Dbpedia spotlight: shedding light on the web of documents. In *I-SEMANTICS*, pages 1–8.

Lev-Arie Ratinov, Dan Roth, Doug Downey, and Mike Anderson. 2011. Local and global algorithms for disambiguation to wikipedia. In *ACL*, pages 1375–1384.

Grigorios Tsoumakas, Min-Ling Zhang, and Zhi-Hua Zhou. 2012. Introduction to the special issue on learning from multi-label data. *Machine Learning*, 88(1-2):1–4.

Mohamed Amir Yosef, Johannes Hoffart, Ilaria Bordino, Marc Spaniol, and Gerhard Weikum. 2011. AIDA: An online tool for accurate disambiguation of named entities in text and tables. *PVLDB*, 4(12):1450–1453.

Mohamed Amir Yosef, Sandro Bauer, Johannes Hoffart, Marc Spaniol, and Gerhard Weikum. 2012. HYENA: Hierarchical Type Classification for Entity Names. In *COLING*, pages 1361–1370.