TakeLab

Laboratorij za analizu teksta i inženjerstvo znanja
Text Analysis and Knowledge Engineering Lab
Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva
Unska 3, 10000 Zagreb, Hrvatska

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

BACHELOR THESIS no. 5326

Entity Recognition and Classification for a Natural Language Database Interface

Ivan Mršić

Zagreb, srpanj 2017.

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING
BACHELOR THESIS COMMITTEE

Zagreb, 3 March 2017

BACHELOR THESIS ASSIGNMENT No. 5326

Student:	**Ivan Mršić (0036480350)**
Study:	Computing
Module:	Computer Science

Title:	**Entity Recognition and Classification for a Natural Language Database Interface**

Description:

A natural language database interface allows the users to query the database in a controlled natural language. Such interfaces rely on natural language processing and machine learning for analyzing the query. One of the steps in query analysis is the extraction of key information from the query, such as named entities.
The topic of this thesis is the automatic extraction of named entities from user queries over a database of famous people. Study the methods for named entity extraction, with an emphasis on machine learning methods for sequential labeling. Devise and implement a method for named entity extraction from user queries in English, which will also cover the classification of each entity as either a target or reference entity with respect to the query intent. Compile a suitable collection for model training and testing, which will include query examples with pre-labeled named entities. Carry out an experimental evaluation of the model, a comparison against a baseline, a statistical analysis of the results, and an error analysis. All references must be cited, and all source code, documentation, executables, and datasets must be provided with the thesis.

Issue date:	10 March 2017
Submission date:	9 June 2017

Mentor:

Associate Professor Jan Šnajder, PhD

Committee Chair:

Full Professor Siniša Srbljić, PhD

Committee Secretary:

Assistant Professor Tomislav Hrkać, PhD

I would like to thank my parents and sister who were always there when I needed them. I would also like to thank my mentor Assoc. prof. Jan Šnajder for help and guidance during the course of this academic year. Lastly, great and special thanks goes to Fran and Juraj for their support and help during the last few months.

Contents

1. Introduction

Throughout the history of coexistence between humans and computers, there always was a big focus on computer interpreting data – making useful conclusions from it and choosing the right way of displaying that conclusion so people could understand it better. As an effect in recent times there has been a surge in graph databases popularity. In addition to storing data, advantage of such database type lies in its ability to visualize data, as seen in Figure 1.1 to make interpretation easier, and use some graph theory properties to perform variety of useful actions such as finding the closest way between two nodes (entities). Another way to enhance communication between human and computer could be made with ability to allow machine to understand user queries in natural language, rather than the user having to input at times complicated queries in *Cypher*, language for *Neo4j* Graph databases.

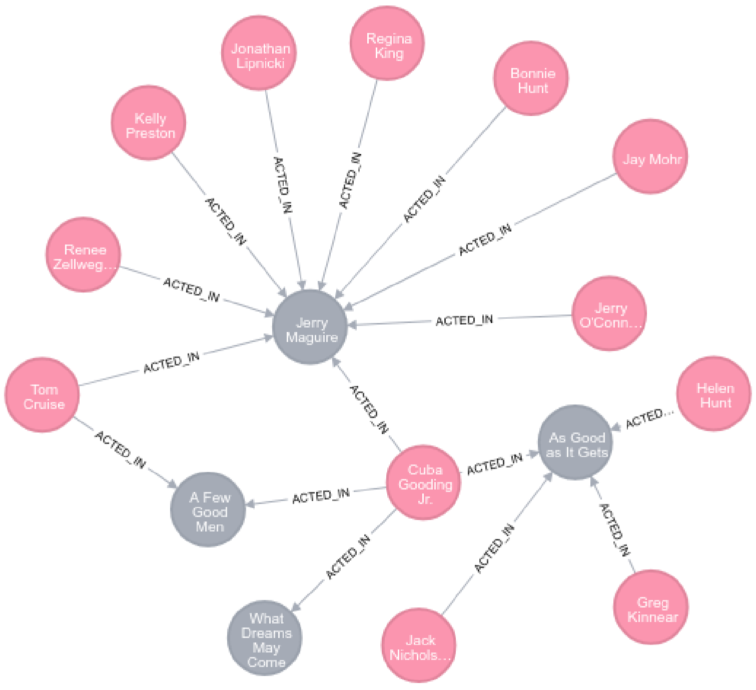


Figure 1.1: Example of movies Neo4j graph database

Creating such communication system is one of tasks and challenges of Natural language processing, *NLP* in later text. NLP is a field of computer science which focuses on computers correctly understanding natural language. Other NLP tasks outside of communication systems are natural language generation, connecting language and machine perception, etc. Statistical NLP relies heavily on machine learning. In such systems computer makes educated guesses based on experience. Often quoted definition of machine learning by Tom M. Mitchell states: “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E” [11]. In recent times, NLP systems started using deep learning to achieve state-of-art results in terms of tagging and parsing human text.

One of the ways to achieve human-computer communication over a database would be to use one of off-the-shelf tools such as *API.AI*[1] or *Luis.ai*[2] to create a simple chat bot and tune it towards our needs. However that approach although it seems it could work at first runs into many problems; most notably spam detection, named entity recognition, and deducting from conversation context. For this reason custom natural language interface should be constructed if we want to make communication between man and database as natural as possible.

The goal of this thesis is construction of entity extractor for such an interface. Another motivation for creating such a system lies in fact that, since it is specialized for database communication it could learn a lot faster specific types of requests that would occur in such systems. Final reasoning for a cunstom solution lies in fact that relationships and entities being searched for can be thought of some more abstract ways, e.g. *Can you give me everyone who starred in Pulp Fiction.* Although it is obvious to us that entity being searched for are people, namely actors, it cannot be extracted explicitly from query. Rather, since different entities aren’t that many, it can be though of as intent; what info does person want to get out of database; taking that approach we classify queries into $count(classes) + 1$. The constructed interface is used for the database with information about actors and movies. Those are very popular subjects online, and the data inside can prove to be convenient for demonstrating different approaches in entity extraction.

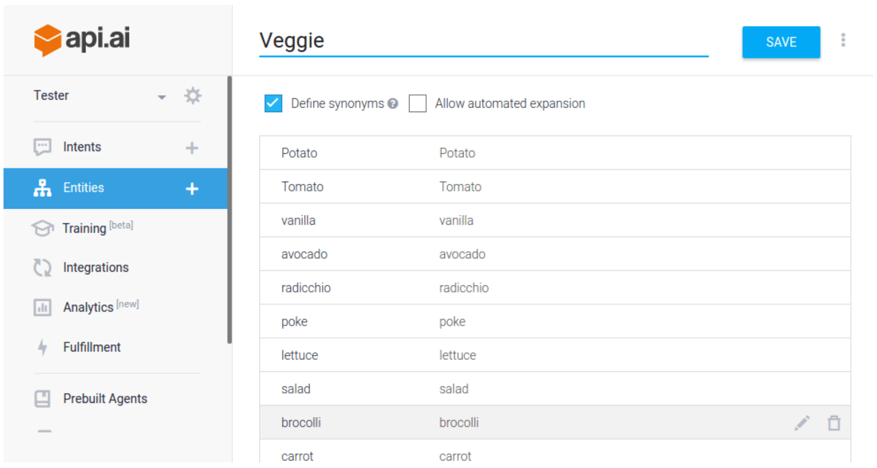
This thesis will first tackle the related work in the entity extraction area. Following that, the thesis will describe the dataset, followed by the description of the different models used to tackle the problem . After that comes brief section about implementation, and in the end results and methods of evaluation are shown.

[1]`https://api.ai/`

[2]`https://www.luis.ai/`

2. Related Work

Inspiration for this system are make your own chatbot systems such as *API.AI* (seen in action on Figure 2.1), *Converse.AI*, and *Luis.ai*. In those systems conversation is split into three parts. The first part is intent detection, the second part entity extraction, and the third (optional) optional part context detection as in does the conversation build on last few queries. Those systems require all entities we want to extract to be specifically named in complex menus, without the ability to learn new entities for extraction from queries where those values are used. Furthermore those systems only learn intents, while entities and intent connections for context purposes all have to explicitly named. While all entity values could be extracted from a database, such way of entity recognition does not cover the cases when entity is missing from a database and the user would expect the entity to be present in database. The final reasoning for for such a system lies in the fact that such systems do not offer the ability to detect entity relationship from a single query.



Veggie

[✓] Define synonyms [ ] Allow automated expansion

Potato	Potato
Tomato	Tomato
vanilla	vanilla
avocado	avocado
radicchio	radicchio
poke	poke
lettuce	lettuce
salad	salad
brocolli	brocolli
carrot	carrot

Figure 2.1: API.AI entity addition

2.1. Matched and Relationship Classifier

There has been a substantial amount of work done on intent and question classification, a problem similar to one presented of relationship and matched entities. Intent detection is a query classification task that can be formulated as $y^{'} = \mathrm{argmax}_y\, p(y \mid w_1, w_n)$, where w_i is i-th word in a query and y is the intent. Such state-of-the-art system use word embeddings [3], word representations with highly dimensional vectors computed from word context. Word enriching with information from semantic lexicons with synonyms and antonyms is used so words would become closer to their synonyms and further away from their antonyms in a vector space. These methods are used to improve scores on word analogy and antonym detection tasks. However, researchers recently tried to use such methods to improve scores on intent classification [3]. Question classification systems on the other hand construct models that use bag of words, semantic, and syntactic features and support vector machine as learning algorithm [8].

2.2. Named Entity Extraction

As for *named entity recognition* (NER), here it is divided into three smaller tasks and used to extract entities being searched by. Such systems can use a wide variety of algorithms, as showcased in CoNLL-2003 (Conference on Computational Natural Language Learning) shared task [18] :

- AdaBoost.MH,
- Memory based learning,
- Transformation-based learning,
- Support vector machines,
- Conditional random fields (CRF).

At CoNLL-2003, the task was to detect appearance of locations, person, organization, and miscellaneous named entities. Also showcased at CoNLL-2003 was that using external knowledge sources such as WordNet [2] or gazetteers can help reduce error rate quite a lot [18]. Almost all participants used lexical features (words), part-of-speech (*POS* for short), orthographic features, and affixes. Their best scores were around 85–90 for precision, recall, and F_1 score on English text. However, the strength of such systems was shown in ability to also extract named entities from German text, with best scores in 80–84 range for precision, 64–66 for recall, and 70–72 for F_1 score. It was also shown that using *label-label* and *word-label* features that CRFs offer greatly helps improve systems performance [17]. Another system that used CRFs was ABNER (A Biomedical Named Entity Recognizer), where entities from quite a specific domain (biomedical domain) had to be extracted [16], similar to domain scope of databases. Depending on entity type being extracted precision, recall and F_1 scores were in the 60–75 range. Making ABNER's results more impressive is fact that its default features comprised orthographic and contextual features only, mostly based on regular expressions and neighboring words, without syntactic or semantic features. Another interesting aspect of ABNER was its speed, being able to tag 33 sentences per second on 500 MHz Pentium III running Linux with 512 MB memory. Further point of interest is how would our natural language interface perform on informal text, database queries. As shown in team name and sport name extraction from such text using CRF results can be surprisingly good, having F_1 score in range 70–90 [10]. As with other CRF systems, features included orthographic and contextual features showing their importance in such systems.

More recently there have been attempts to fuse named entity recognition systems with state-of-the-art word embedding models, using previously trained models on more general corpus such as Portugese Wikipedia [15] or previously trained skip-n-grams on that specific corpora [4] . Such systems performed well slightly beating out CoNLL shared task scores on the same corpora with inclusion of few additional features. As for numeric entities, most of year, age, and unit extraction is done by regular expressions, and assigned probabilities based on words surrounding the number in that query.

3. Dataset

To test the idea of natural language interface a dataset of user queries had to be constructed. This was done by the three students involved in the creation of the interface: Juraj, Fran and myself. It was done in about a week. The constructed dataset consists of 305 positive and 47 negative queries. Only positive queries are considered, since negative ones are used only for spam classification. Out of 305 positive ones only 277 are actually used in the end, since 28 do not follow guidelines set up for dataset construction. The guidelines were to write the whole conversation consisting of about 5 queries, and then to tag entities in those queries. One entity of each kind (matched entity, relationship entity, and searched by entity), and intent were allowed per query. Overall dataset consists of 1507 words, which averages about 5.44 words per query. Queries are grouped in conversations split by blank line between them. Dataset is stored as `.csv` file. Examples of the dataset are shown in the next chapter.

3.1. Dataset Tagging

Each query was manually tagged with 4 external tags for query and entity classification; additional internal tag was used for query extraction. Examples of tagged query is:

- *Show me countries where those <Begin>actors<property | NN> live in, UNION, Country, LIVES_IN, TRUE.*

With the next few queries showing the whole conversation:

- *List all actors in the movie <Begin>Imitation game<movie | NN>,WHERE | POSITIVE,Person,STARRED_IN,*
- *Remove those born <Begin>before 1985<years | N>,WHERE | NEGATIVE,,,*
- *Show only those that live in <Begin>London<City | NN>,WHERE | POSITIVE,,,*
- *Add those living in <Begin>New York<City | NN>,UNION,,,*

– *Sort them by the number of <Begin>movies<property | NN> in which they played,ORDER BY | DESC,,STARRED_IN,*

3.1.1. External Tags

Four external tags used for classification with their short descriptions can be seen in Table 3.1.

Table 3.1: External tags

Tag type	Description
INTENT	Tag that marks wanted action over graph database
MATCHED ENTITY	Shows entity being matched in database
RELATIONSHIP	Signalizes relationship between matched and entity being matched by
PARAPHRASE	Signalizes whether query is paraphrase

From those four tag types, the second and the third type are used for entity classification, namely matched entity (one being searched for in the query) and the relationship between matched and entity being searched by. In both of those cases, in addition to entities present in database, a special NONE tag was used when it was deemed the classifier should not be able to extract matched entity or relationship explicitly, rather conversation context should be used to determine them. There are five matched labels in total:

– Person,

– Movie,

– Country,

– Continent,

– NONE.

and there are also five relationship labels:

– STARRED_IN,

– LIVES_IN,

– DIRECTED_BY,

– IS_PART_OF,

– NONE.

3.1.2. Internal Tag

Named entities, ones used as clauses in Cypher queries, needed to be tagged inside the queries themselves. Internal tags are in following form: <Begin> **entity** <tag type | numeric tag>. Such tagging was used to most easily identify where entity starts and what type is it. The second part of tag was split in two parts using |. The first part is used to determine exact entity type, while the second part was used to determine whether entity is numeric or non numeric type since different methods were used for extractions in both cases. Numeric tags and their short descriptions can be seen in Table 3.2. Examples of first part of tags were age, name, movie, etc.

Table 3.2: Numeric tags

Tag type	Description	Examples
N	Signalizes numeric entity	age, numbers, height etc.
NN	Signalizes non numeric entity	name, movie etc.

3.2. Dataset processing

3.2.1. Query Processing

After loading `.csv` file containing queries, queries are processed one by one. Query processing consists of a number of subtasks:

– Tokenization,

– POS tagging,

– Lemmatization,

– Labeling them for supervised learning

  - simple 0-n class labeling for external tags ,
  - BIO scheme for inline tags.

Tokenization, POS tagging, and lemmatization is also done for new (user input) queries and all three processes are done using NLTK toolkit [1]. POS tags in NLTK tagger are ones from Penn Treebank Project [6]. Lemmatization is done using WordNet [2] lemmatizer. It is the process of converting a morphologically inflected word form into its canonical, or dictionary, form (e.g., infinitive for verbs, nominative singular for nouns, etc.). *BIO* (Begin Inside Out) scheme for named entity labeling means that every word in a query was tagged with one of three tag types. BIO tags legend is shown in Table 3.3. Words with begin and inside tags were never labeled as such: they

Table 3.3: BIO tagging legend

Tag type	Description
B	Signalizes word is the beginning of a named entity
I	Signalizes word is inside of a named entity
O	Signalizes word is outside of a named entity

always contained in them what named entity type they were tagging, e.g. , B-name, I-country. A tagged query should look like 3.4.

Important restriction put in place during dataset creation was decision that only a single named entity would be allowed per query, so a user couldn't search for movies both Leonardo Di Caprio and Scarlett Johansson, but instead he would have to search first for movies Leonardo was in and then for movies Scarlett starred in. As for possible

Table 3.4: BIO scheme example

t	Word	BIO tag
1	Give	O
2	me	O
3	all	O
4	actors	O
5	from	O
6	United	B-country
7	Kingdom	I-country

entity classes and their value ranges, they can be seen in Table 3.5. There are in total

12 named entity types, so 25 different labels were needed (one outside, 12 begin, and 12 inside).

Table 3.5: Entity type

Entity type	Value range or examples
age	$[0 - 100]$
continent	Asia, Europe
country	Croatia, United Kingdom
gender	male, fame
maritial status	married, divorced
movie	Pulp Fiction
name	Mila Kunis
nominations	$[0 - 10]$
property	name, revenue
rating	$[0 - 100]$
revenue	$[0 - \infty]$
role	main

3.2.2. Dataset Structure

At the data structure level, `Dataset` is a class which contains a list of conversation objects. Each conversation object has a list of all queries over the database from that user-database conversation. Both `Dataset` and `Conversation` objects have a number of useful methods for help with dataset analysis. A query object has a number of member variables, namely query as a single, non tagged string, query as word list, query as list of tuples (word, POS), query in BIO scheme, lemmatized query, and a boolean signalizing whether named entity inside of query is numeric.

4. Models

To solve the problem of entity extraction from natural language queries, four models are constructed. One for entity being searched for, henceforth “matched model”, one for relationships between entity being searched for, henceforth “relationship model”, one for determining whether entity being searched by is numeric or non numeric, henceforth “numeric model”, and for model for extraction of non numeric (named) entities being searched by, henceforth: “entity model”.

4.1. Matched and Relationships Classifier

Even though matched entity and relationship classification are two separate problems, they are viewed together because most of the features are used in both models, and the same classification algorithms are used in both cases. Both are viewed mostly as intent classification tasks since nothing from the query had to be explicitly extracted, only the entity class a user is looking for. Another reason for viewing these problems together is similarity between problems in special case when it isn’t expected of classifier to determine the entity, instead it should be deducted from the context, or in case of relationship there doesn’t have to be one present in the query at all – therefore, classifier should classify that case as NONE.

4.1.1. Algorithms

To tackle matched classification problems two supervised machine learning algorithms are considered: *SVM* (Support vector machine) and gradient boosting. Two kernels are considered for SVM: a linear kernel and an *RBF* (radial basis function) kernel.

SVM is described in [19]. SVM is machine learning algorithm which has been successfully applied to many real world classifying problems, such as handwritten digit recognition, object recognition, and, importantly text classification [19]. It can also be used for regression problems. We first define an SVM for the binary classification problem. Given binary divided dataset of n feature-vector - label pairs, $((\vec{X}_1, \vec{y_1}), \ldots, (\vec{X}_n, \vec{y_n}))$, an SVM goal is to find "maximum range hyperplane", where a normal vector of hyperplane $\vec{w}$ is defined with:

$$\vec{\mathbf{w}} \cdot \vec{\mathbf{x}} - b = 0 \tag{4.1}$$

with hyperplane dividing points in vector space into two classes. The margin is defined with $\frac{\vec{b}}{||\vec{\mathbf{w}}||}$. If data is indeed linearly separable, the margins can be written as:

$$\vec{\mathbf{w}} \cdot \vec{\mathbf{x}} - b = \pm 1 \tag{4.2}$$

and on Figure 4.1 margins can be seen as dotted lines. This case is known as *hard-margin*. There also exists another formulation of SVM, which is used to classify ex-

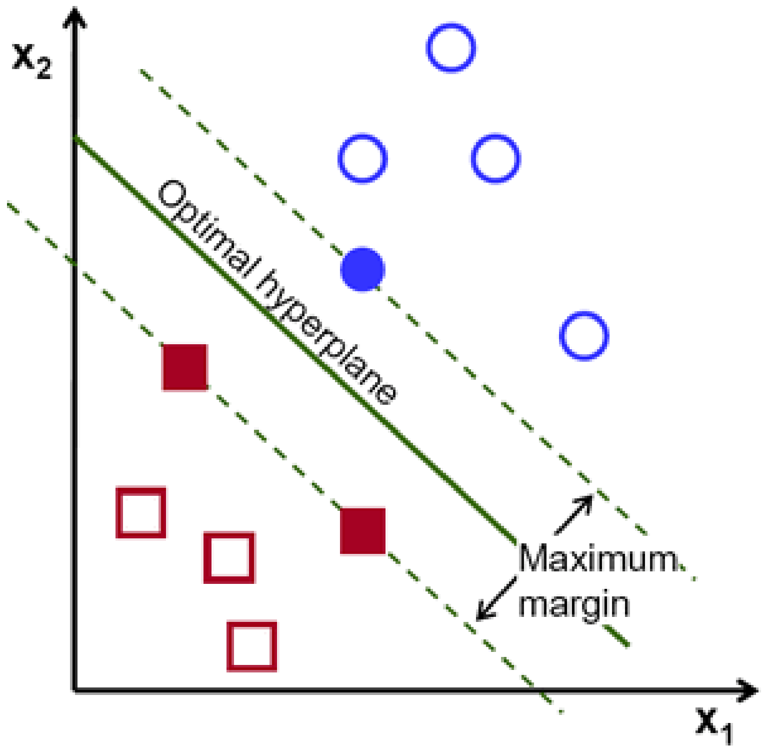


Figure 4.1: Binary SVM classifier, illustration taken from http://docs.opencv.org/

amples where data is not linearly separable with linear classifier, approach also known as *soft-margin*. Goal of the soft-marigin is to minimize the following function:

$$[\frac{1}{n}\sum_{i=1}^{n} max(0, 1 - y_i(\vec{\mathbf{w}} \cdot \vec{\mathbf{x_i}} - b))] + \lambda||\vec{\mathbf{w}}||^2 \tag{4.3}$$

where for small λ values classifier will behave almost identically to hard-line classifier. It penalizes wrongly classified points on the wrong side of the hyperplane or within the margin depending on how far the point is from the hyperplane. For a larger distance between point and hyperplane the larger penalty is given.

Kernel trick enables us to use the different kernel functions to easily map classifier to more dimensions, thus solving non linear problems. Kernel is defined in the following way:

$$K(\vec{\mathbf{x}}, \vec{\mathbf{y}}) = (\phi(\vec{\mathbf{x}}), \phi(\vec{\mathbf{y}}) \tag{4.4}$$

.

Examples of the kernel functions used in this thesis are the linear kernel:

$$k(\vec{\mathbf{x}_\mathbf{i}}, \vec{\mathbf{x}_\mathbf{j}}) = (\vec{\mathbf{x}_\mathbf{i}}, \vec{\mathbf{x}_\mathbf{j}})^d \tag{4.5}$$

and RBF kernel:

$$k(\vec{\mathbf{x}_\mathbf{i}}, \vec{\mathbf{x}_\mathbf{j}}) = \exp(-\gamma ||\vec{\mathbf{x}_\mathbf{i}} - \vec{\mathbf{x}_\mathbf{j}}||^2) \text{ for } \gamma > 0 \tag{4.6}$$

Gradient boosting is described in [7]. It involves three elements:

1. A loss function to be optimized,
2. A weak learner to make predictions,
3. An additive model to add weak learners to minimize the loss function.

The loss function used depends on the problem type being solved; an example would be the logarithmic loss for classification. One of the gradient boost benefits is that it can work with any loss function. The most commonly used weak learners are the decision trees in gradient boosting, and they are constructed in greedy manner based on purity scores like Gini score. Larger trees have 4–8 levels. While they are being added one at the time, a gradient descent is used to minimize the loss. The output for the new tree is then added to the output of the existing sequence of trees in an effort to correct or improve the final model output; always a fixed number of trees are added. One of the main gradient boosting downsides is the fact that it is prone to overfitting, which is to say it can learn too well and fail to generalize when given new examples [7].

4.1.2. Features

Since it was shown that word embeddings have good performance on intent classification [3] tasks, they are the first feature considered for classification. To obtain embedding for each particular word, pre-trained vectors trained on part of Google News dataset (about 100 billion words) were used [9]. The model contains 300-dimensional vectors for 3 million words. If a word isn't in the vocabulary, 300 zeroes are put in instead of word embedding from the pre-trained model. Embedding for query is calculated by appending vectors of all words in query. Furthermore it was also shown that POS and synonym features tend to perform well [3]. Synonyms are acquired from WordNet [2], noun synonyms for matched entities and verb synonyms for relationships. The reasoning behind this is a convection in Neo4j that entities should be nouns and relationships should be verbs. Although it is not recommended to mix word embeddings and BOW type features, BOW type features are also constructed. Bag of words is a simple binary feature showing whether the given word appears in a document. Features are shown in Table 4.1.

Entity synonyms were used only when testing matched models and relationship synonyms were used only when testing relationship models.

4.1.3. Model Testing

For both the SVM classifier and gradient boosting classifier scikit-learn [13] implementations is used. For both algorithms dataset is split into a train and test set with a ratio of $80 : 20$, with classifier learning on 80% with 20% of data unseen to classifier being used for testing. In case of SVM, both linear and RBF kernels are considered. For linear kernel, the optimal value of hyperparameter C is considered from range $[2^{-15}, 2^{-14}, \ldots, 2^{14}, 2^{15}]$. That same range is used for hyperparameters C and γ in the RBF kernel. In case of gradient boost, *the number of boosting stages* and *maximum depth* of individual estimators is optimized. Number of boosting stages is searched for in set $\{200, 220, 240, 260, 280, 300\}$, while the maximum depth search range is $\{2, 3, 4, 5, 6\}$. Grid search over a Cartesian product of parameters is performed. Search is performed using a 10-fold bin cross validation, meaning the test set is split into 10 parts and the classifier learns with each combination of parameters over all 10 splits. Hyperparameters with the best average score are taken for final training on the whole train set before testing the classifier on new, unseen train set. In the end, classifier learns over all data available, trying to construct the best possible model.

4.2. Numeric Classifier

Before extracting the entity being searched by, another step had to be undertaken: classifying whether that entity would be numeric or non-numeric. This classification is

Table 4.1: Matched and relationships features

Python module	Description	Value range
`adjectiveratio`	Ratio of adjectives in the query	$x \in [0, 1]$
`adverbratio`	Ratio of adverbs in the query	$x \in [0, 1]$
`bigrams`	Apperance of a word pair w_i, w_{i+1}, in that order in query	$x \in [0, 1, \dots)$
`bow`	Whether the word w_i appears in query	$x \in [0, 1, \dots)$
`intent_tags`	Feature that shows what is the query intent	$x \in \{0, 1\}$
`lemm_bow`	Whether word w_i appears in the lemmatized query	$x \in [0, 1, \dots, n)$
`nounratio`	Ratio of nouns in the query	$x \in [0, 1]$
`pronounratio`	Ratio of pronouns in the query	$x \in [0, 1]$
`propnounratio`	Ratio of proper nouns in the query	$x \in [0, 1]$
`querylen`	Number of words in the query	$x \in [1, 2, \dots, n)$
`synonyms_entities`	Whether entity synonym s_i appears in the query	$x \in [0, 1, \dots, n)$
`synonyms_relation`	Whether relationship synonym s_i appears in the query	$x \in [0, 1, \dots, n)$
`verbratio`	Ratio of verbs in the query	$x \in [0, 1]$
`word2vecquery`	Word embedding w_i	$x \in [0, 1]$

important, because depending on the entity type methods for extraction were different.

4.2.1. Algorithms

Although checking whether a query contains numbers can be done using an expert system, decision tree is used because of interpretability such model gives when graphically displayed, which can easily point out to cases when such system could fail, allowing for easier conclusion why such a system might fail to classify an entity correctly to help us determine future course of action. Decision trees are described in [5]. **Decission trees** are a predictive model that generates decision rules on which model bases its decisions. Decision trees are a supervised learning method used in both regression and classifcation. Given points x_i, a decision tree recursively partitions the input space in a way that points with same label are grouped together. In each step it chooses a split that leaves the least ammount of impurity, impurity being messured in gini score or as cross-entropy. Exact decision tree implementation in scikit-learn uses *CART* (Classification and Regression Trees) algorithm [13].

4.2.2. Features

Since this can be done by expert systems, only a handful of simple features are constructed. Features resemble rules that would have been used in such a system. All features are simple binary features that indicate whether expression being searched for is present in query q_i. The features are shown in 4.2 :

Table 4.2: Numeric classifier features

Python module	Description	Value range
`hascardinal`	Searches q_i for cardinal numbers in it based on POS a tag	$x \in \{0, 1\}$
`hasnumber`	Simple regular expression to look for numbers in q_i	$x \in \{0, 1\}$
`haswrittennumber`	Searches q_i for match inside of handcrafted list of number words	$x \in \{0, 1\}$

4.2.3. Model Testing

Once again the dataset is split $80 : 20$ into train and test set. Testing for the best tree depth is done by using cross validation over 10 random folds for each tree depth. Tree depths considered are $3, 4, 5, 6$. That range is chosen because feature number is small and simple, easy to understand system separating queries with numeric and non-numeric entities is desired, so complicated tree isn't something we want. The complete cross-validation process takes very little time, around 20 seconds, showcasing model simplicity.

4.3. Named Entity Extraction Model

4.3.1. Algorithms

As evidenced by CoNLL shared task, conditional random fields CRF [17] gave good results on the task of of entity extraction, so a decision was made to use it as machine learning algorithm for named entity extraction. CRFs are also described in [17].

CRF is machine learning method applied to structured learning, enabling us to learn not only from feature vector for word w_i itself but also from neighboring data. In this particular case a linear chain CRF was used. By definition, a linear CRF is

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{t=1}^{T} \exp \sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, x_t) \tag{4.7}$$

Here we calculate the conditional probability of vector $\mathbf{y}$ (given vector $\mathbf{x}$ as an input, with $\mathbf{y}$ to be classified into a single, most probable class) as a product of an exponential function with exponent being dot product of a parameter vector Θ and the feature vector f of (y_t, y_{t-1}, x_t), them both being of length K, over all T observations. This is in the end divided by the Z(x), where Z(x) is an input-dependent normalization function.

$$Z(\mathbf{x}) = \sum_{y} \prod_{t=1}^{T} \exp(\sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, x_t)) \tag{4.8}$$

Even though it sums over all possible state sequences, an exponentially large number of terms, it can be computed efficiently with forward-backward algorithm. Notice that a linear chain CRF can be described as a factor graph over x and y with simple substitution :

$$\psi_t(y_t, y_{t-1}, x_t) = \exp \sum_{k=1}^{K} (\theta_k f_k(y_t, y_{t-1}, x_t)) \tag{4.9}$$

A factor graph is a bipartite graph G = (V,F,E) in which one set of nodes $V \epsilon$ {1,|Y|} indexes the random variables in the model, and the other set of nodes $F \epsilon$ {1,A} indexes the factors, with such a graph representing factorization of a function. Factorization is done in a following way: a distribution p(**y**) factorizes according to a factor graph G if there exists a set of local functions Ψ_a as:

$$p(\mathbf{y}) = Z^{-1} \prod_{a \epsilon F} \Psi_a(\mathbf{y}_{N(a)}) \tag{4.10}$$

Factor graph G describe an undirected model in the same way a collection of subsets does, useful for grouping multiple words into a single, most probable named entity.

Such a way of representing data enables us to extract not only information from feature vector, but also from the connections between labels [17], shown in Figure 4.2. A PyStruct implementation of linear chain CRFs is used.[1]

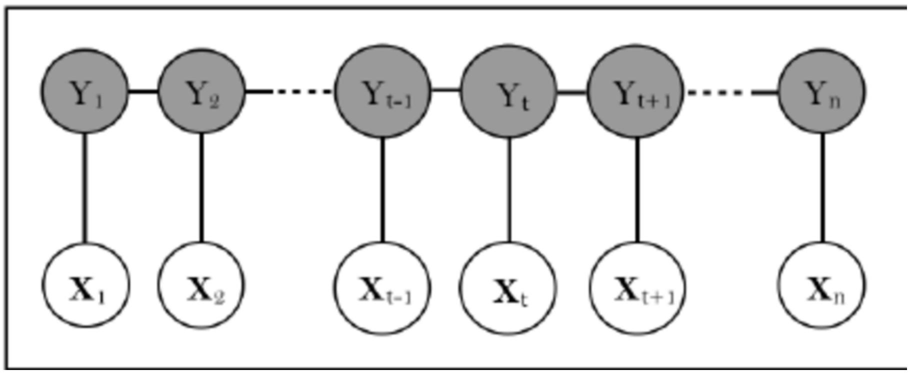


Figure 4.2: Conditional random fields, illustration taken from https://www.researchgate.net

4.3.2. Features

For the task of named entity extraction, for each word w_t of query q_i the features are constructed and they can all be seen in Table 4.3. Once again, the first feature is Word2Vec word embedding from the Google news model. Other features included gazetteers, bag of words, regular expression matching certain word patterns, POS features, etc [12]. There are also two features where neighboring words w_{t-1} and w_{t+1} in vicinity of a word w_t were looked at. For that purpose, two special tokens were added: <*START*>, which symbolizes word is at the beginning of a query, and <*END*>, which symbolizes the word is at the end of a query. The relative word position in the query is calculated as $\frac{position(w_t)}{length(q_i)}$.

4.3.3. Model Testing

The dataset is split again into train to test ratio of $80 : 20$. Cross validation is done again here with 10 folds.

[1]https://pystruct.github.io/

Table 4.3: Matched and relationships features

Python module	Description	Value range
`bow`	Word is equal to another one $w_t = v$	$x \in \{0, 1\}$
`digitword`	w_t contains digit	$x \in \{0, 1\}$
`endsindot`	w_t ends with dot	$x \in \{0, 1\}$
`isadjective`	w_t is adjective	$x \in \{0, 1\}$
`isadverb`	w_t is adverb	$x \in \{0, 1\}$
`iscapitalized`	w_t is capitalized	$x \in \{0, 1\}$
`iscaps`	w_t is whole in uppercase	$x \in \{0, 1\}$
`iscountry`	w_t is in list of countries	$x \in \{0, 1\}$
`iscontinent`	w_t is in list of continents	$x \in \{0, 1\}$
`isnoun`	w_t is noun	$x \in \{0, 1\}$
`ispronounn`	w_t is pronoun	$x \in \{0, 1\}$
`ispropnoun`	w_t is proper noun	$x \in \{0, 1\}$
`isstopword`	w_t is in stop word list	$x \in \{0, 1\}$
`isuperlower`	w_t is written in mixed letter cases	$x \in \{0, 1\}$
`isverb`	w_t is verb	$x \in \{0, 1\}$
`prefix`	w_t starts with one of most common prefixes in English	$x \in \{0, 1\}$
`sufix`	w_t ends with one of most common noun suffixes in English	$x \in \{0, 1\}$
`w2v`	300 dimensional w_t Word2Vec embedding	$x \in [0, 1]$
`wordafter`	bow-like feature for $w_t t + 1$	$x \in \{0, 1\}$
`wordbefore`	bow-like feature for $w_t t - 1$	$x \in \{0, 1\}$
`wordlen`	character length of w_t	$x \in [0, 1, \ldots, n]$
`wordposistion`	relative position of w_t	$x \in [0, 1]$

4.4. Numeric Entities Extraction

For numeric entities such as age, years or, revenue extraction is done using regular expressions and expert system, which is based on number range and the appearance of other keywords, such as *over*, *between*, *older* etc . After extracting the words of interest, system creates entity object for few most likely entities and passes them to a user to choose from them if it determines they are too similar to make prediction on type. An example would be *Over 50*; although from context it may be clear to us if that represents age or revenue, without the context the entity cannot be deducted.

5. Implementation

The entity extraction system has been built in Python and is split into four main components:

- Dataset processing,
- Feature extraction,
- Model learning,
- Using predicted models in future.

Whole process of taking a new query and using it to train a model or simply extracting the entities from the query is visualized in Figure 5.1. There are two different feature extractions in the end. One for matched, relationship and, numeric model, and another one for the named entity recognition model. All features use single query versions of feature extraction, so when a new query comes, the same code which was used to extract features from the whole dataset can be used to extract feature from a single query. After feature extraction, four models are trained, tested ,and, trained again, but this time on whole dataset to get as many examples as possible. These models are then serialized as `.pkl` files, which are used to make predictions on new queries. Serialization is done because training new models every time would be too time consuming. The complete output is in shown in Listing 5.1.

```
{
    'AGGREGATION': None,
    'DISTINCT': None,
    'LIMIT': None,
    'MATCH': ('country',
    {
        'numeric': False,
        'operator': None,
        'probability': 1.0,
        'units': country,
        'value': US
    })'MATCHED': Person,
    'MULTIPLE\_FITLERS'[

    ],
    'OPERATOR': 'REDUCE',
    'ORDER\_BY': NONE,
    'REDUCE\_FILTER': NONE,
    'RELATIONSHIP': LIVES\_IN.'TOP': None,
    'WHERE': POSITIVE,
    'src': 'Actors from US'
}
```

Listing 5.1: Sample JSON output

After a new query is processed and entities extracted from it, it would form part of dictionary sent to context.

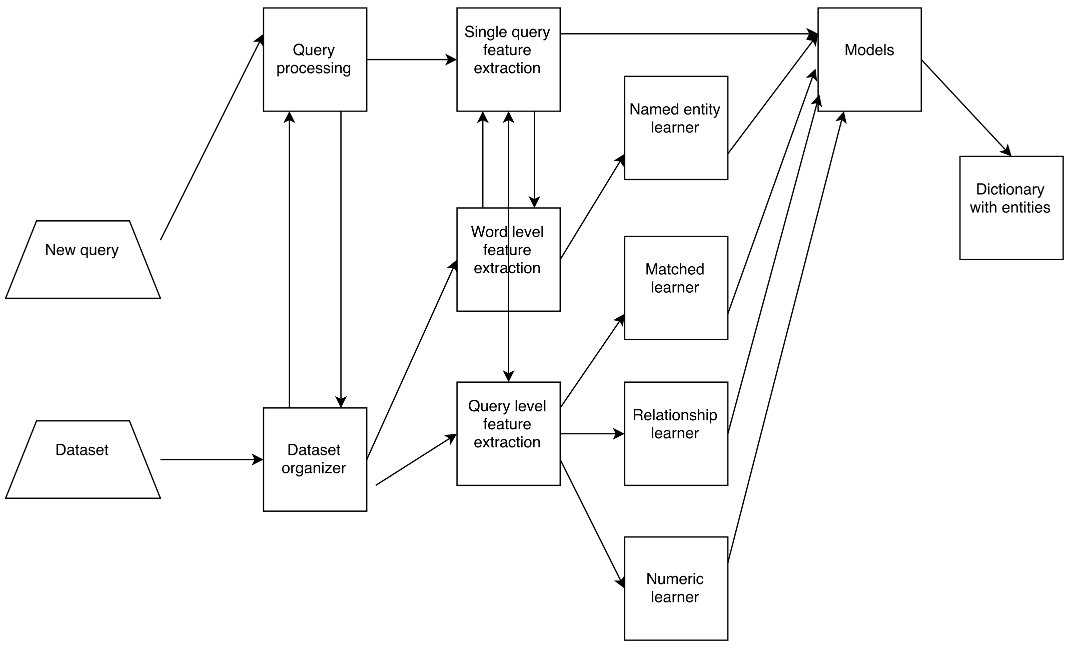


Figure 5.1: Pipeline of query input, process and model training/entity extraction

6. Results

Results have been rather encouraging, showing that even with a small amount of training data, a lot can be precisely deduced about the query.

6.1. Evaluation Methods

Four evaluation scores will be shown for each model:

- accuracy,
- F_1 score,
- Precision,
- Recall.

Before defining metrics by [14], following four terms have to be defined [14]:

- True positive,
- False negative,
- False positive,
- True negative.

True positive (TP) is a correctly predicted positive example, *false negative* (FN) is a wrongly predicted positive example, positive example predicted as negative, *false positive* (FP) is wrongly predicted negative example, negative example predicted as positive, and a *true negative* (TN) is correctly predicted negative example as show in Figure 6.1. With that, the definition of accuracy is as follows:

$$Accuracy = \frac{\sum TP + \sum TN}{\sum TP + \sum FN + \sum FP + \sum TN} \tag{6.1}$$

While that metric is quite useful, it performs badly on unbalanced datasets. To tackle that problem, precision and recall are defined:

$$Precision = \frac{\sum TP}{\sum TP + \sum FP}. \tag{6.2}$$

$$Recall = \frac{\sum TP}{\sum TP + \sum FN}. \quad (6.3)$$

The intuition behind precision lies in the fact that we want to be right in predicting correct samples, not to predict negative ones as positive, while the intuition behind recall is that we want to predict correctly as many positive samples as we can, to leave as little positive samples predicted as negative. Ideally, we would like to have both, and this is what F_1 score stands for– simply a harmonic mean between precision and recall:

$$F_1 = \frac{2 \cdot precision \cdot recall}{precision + recall}. \quad (6.4)$$

It should be noted that F_1 is only a version of F_β score with $\beta = 1$. With β serving only to adjust importance of precision and recall, the score is defined in the following manner:

$$F_\beta = \frac{(1+\beta^2) \cdot precision \cdot recall}{(\beta^2 \cdot precision) + recall}. \quad (6.5)$$

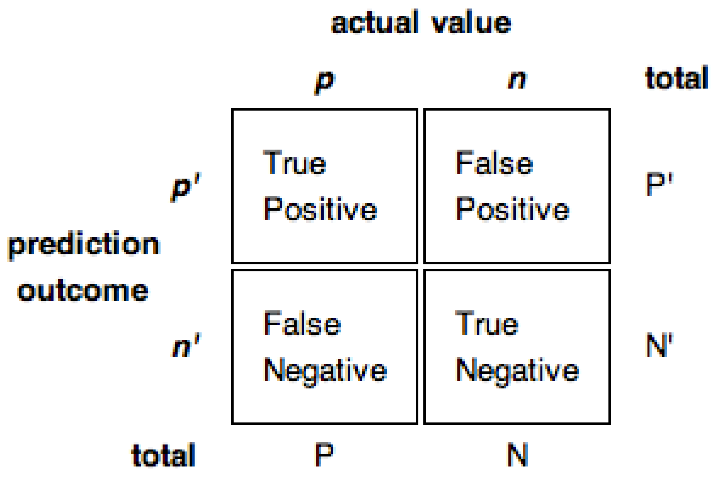


Figure 6.1: Error types, illustration taken from https://alliance.seas.upenn.edu/

6.2. Matched and Relationship Models

Having chosen F_1 score as the scoring function, the final F_1 score is calculated as a mean of F_1 scores for all prediction classes. Scores are also averages between 5 folds in $80 : 20$ split for the bootstrap, meaning the nested cross-validation was performed. Benchmarks for both matched and relationship models are SVM with linear kernel trained with default setting and Word2Vec embedding as the only feature. After performing cross-validation over all three algorithms, the results are quite interesting. SVM with linear kernel gives the best result, although not by much; SVM with RBF kernel performs slightly worse although it performs better than gradient boosting; with results of those experiments shown in Tables 6.1 and 6.2. Because of this results, the SVM with linear kernel is chosen to be algorithm to perform feature selection for. After feature selection, it turns out that best features for both matched and relationship model are the same ones. Features that seem to give the best result are: lemmatized bow, Word2Vec representation of a query, POS ratios, and respective synonyms for both models. As shown in Table 6.1, results have progressed quite a lot from benchmark we started with, but with feature selection it seemed not to improve much outside of F_1 micro score. It should be noted that recall is quite low compared to all other metrics. There are two possible reasons for this: similarity between queries that should be classified with exact entity and ones that should be classified with NONE is quite small and another human factor that comes into play during annotation. While results

Table 6.1: Matched results

Model	Accuracy	F_1 macro	Precision	Recall	F_1 micro
Baseline	64.29%	19.56	16.07	25.00	64.15
SVM linear	89.20%	83.81	96.34	74.17	85.17
SVM RBF	87.50%	81.87	95.45	71.67	85.17
Gradient boosting	85.71%	80.98	91.25	72.78	83.21
SVM linear + feature selection	89.29%	84.21	96.42	74.76	89.29

for matched entities have been quite good, results for relationships are rather disappointing as seen in Table 6.2 . Although benchmark fares relatively better, subsequent improvements have been far less noticeable than for matched entities. Another interesting thing to note here is the fact that the linear and RBF kernels gave exactly the same results. Here precision and recall are more evened out, even though they aren't high. All scores for the matched and the relationship model are averaged across 5 splits of dataset to a train and the test set, this was done because of the bootstrap usage. That in turn means hyperparameters calculation each time was done with a nested 10-fold cross-validation.

Table 6.2: Relationship results

Model	Accuracy	F_1 macro	Precision	Recall	F_1 micro
Baseline	71.42%	20.83	18.18	24.39	69.67
SVM linear	80.38%	66.84	57.69	79.46	79.46
SVM RBF	80.38%	66.84	57.69	79.79	79.46
Gradient boosting	78.57%	50.71	44.24	59.39	77.24
SVM linear + feature selection	82.14%	67.40	58.02	80.38	82.00

Non-parametric method bootstrap is used to show whether there really was any improvement over benchmark at the end. Bootstrap takes samples from data provided (in this case it takes samples from the test set, 20% of the total dataset) so it can give estimates of property of estimator; here difference between F_1 scores (to be more precise, we are testing whether there is any statistically significant difference between results of both models). Samples are sorted by score and based on percentile we give our estimator estimates. For both matched and relationship zero hypothesis is the same; there is no difference between the benchmark and the final model F_1 score. In both cases 10000 iterations were used.

$$H_0 = 0 \tag{6.6}$$

Having chosen a 5% two-sided test we are looking if at least 5% of the pooled data is more extreme than H_0, and by counting those occurrences and then dividing them by the number of samples we obtain p-value. For the matched model p-value of the F_1 macro score is 0.0160, and for the F_1 micro score p-value is 0.0014. Relationship models yield similar values with the F_1 macro p-value being 0.0033, and the F_1 micro p-value score being 0.0037. None of those p-values are 0.05 (our chosen significance level) or larger which leads to the rejection of H_0 and conclusion there is a statistically significant difference between benchmark and final models.

6.3. Numeric Model

Since there is a comparable number of both numeric and non-numeric parameters to be extracted in dataset, accuracy is chosen as scoring function. Model results can be seen in Figure 6.2. Good results shown in Table 6.3 coming from such a simple system as this one should be taken with care: in the dataset there are no addresses or any other similar entities consisting of both numeric and non-numeric part. With accuracy around 91%, there is still place for improvement (adding address regex, checking if cardinal number is written in capital letters as in a movie name, etc.) and decision tree visualization can help us with that.

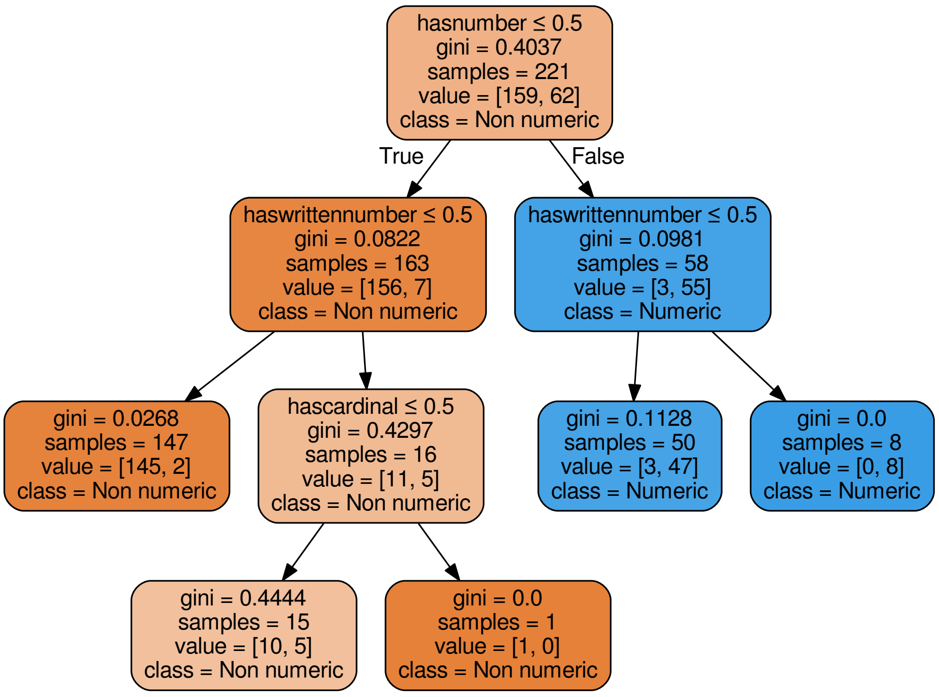


Figure 6.2: Numeric / non numeric decision tree

Table 6.3: Numeric results

Model	Accuracy	F_1 macro	Precision	Recall	F_1 micro
Decision tree	91.07.%	89.90	89.40	90.49	91.07

An interesting thing to note from Figure 6.2 is that, whenever a written number occurs, it is far more likely that the entity is non numeric if that number is a cardinal number.

6.4. Named Entity Extraction Model

Named entity recognition produced very good results, considering how little training data there actually was. Overlapping (showing whether the right words are chosen as named entities) F_1 score is 91.27 and F_1 score for overlapping and actually predicting the correct label is 83.58. The reasoning behind why there is a lot of examples were the classifier guessed correctly that given words were part of named entity but failed to classify them in the correct class could be the fact that there aren't many training examples, for instance multiword country names such as South African Republic are classified as name instead of country. Another concern is that, while the results are quite good, the classifier will match multiple named entities in the same query when it should really only classify one, as this is the convention by which the dataset was annotated. For instance, *All movies from United States*. While it will correctly predict United States as a country, the word movies will also be classified as beginning of the property, not "Outside".

7. Conclusion

With the ability to access greater and greater sources of knowledge, the ability to present it in the right way is also becoming important. One way of modeling and visualizing data are graph databases. As current interfaces are to complicated for the ordinary user, the ability to input queries in natural language, to talk to a database, would be of great help to the end user. The approaches where one creates custom chatbots, such as API.AI fail in this endeavor.

The topic of this thesis was the task of entity recognition and extraction from natural language queries to a database. Four models for extraction have been developed to extract in the end, to extract matched, relationship, numeric, and non numeric named entities. While models work reasonably well given the amount of data, there is still work for improvement. Scores for named entity extraction are especially good, considering prior work, which reports performance in the 85–90 range[18]. One possibility to improve named entity recognition would be to make sure only one entity is being extracted from the query at the time. As for matched and relationship classification, it would be interesting to try deep learning methods with already trained models and trying to improve them with synonyms and antonyms learning [3].

One of the lessons learned was that building a system for named entity extraction from natural language queries is much easier if the system is focused on a specific database, rather than a general one because general one would require way more examples, where perhaps lexicons with domain specific knowledge could be used to enhance performance.

There are a number of interesting experiments that come to mind when thinking where one could go next: creation of this interface for another language, e.g. Croatian, German, etc. , and other languages of interest, or to take another domain-specific and see if there were any domain specific features here that conditioned the performance.

This interface could also be enhanced with a better user interface and speech recognition capability.

BIBLIOGRAPHY

[1] Steven Bird. Nltk: the natural language toolkit, 2006.

[2] Christiane Fellbaum. Wordnet, 1998.

[3] Joo-Kyung Kim, Gokhan Tur, Asli Celikyilmaz, Bin Cao, i Ye-Yi Wang. Intent detection using semantically enriched word embeddings, 2016.

[4] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, i Chris Dyer. Neural architectures for named entity recognition, 2016.

[5] Oded Z Maimon i Lior Rokach. Data mining with decision trees: Theory and applications (series in machine perception and artifical intelligence), 2008.

[6] Mitchell P Marcus, Mary Ann Marcinkiewicz, i Beatrice Santorini. Building a large annotated corpus of english: The penn treebank, 1993.

[7] Llew Mason, Jonathan Baxter, Peter L Bartlett, i Marcus R Frean. Boosting algorithms as gradient descent., 1999.

[8] Donald Metzler i W Bruce Croft. Analysis of statistical question classification for fact-based questions, 2005.

[9] Tomas Mikolov, Kai Chen, Greg Corrado, i Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[10] Einat Minkov, Richard C Wang, i William W Cohen. Extracting personal names from email: Applying named entity recognition to informal text, 2005.

[11] Tom M. Mitchell. Machine learning, 1997.

[12] David Nadeau i Satoshi Sekine. A survey of named entity recognition and classification, 2007.

[13] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python, 2011.

[14] David Martin Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation, 2011.

[15] Cicero Nogueira dos Santos i Victor Guimaraes. Boosting named entity recognition with neural character embeddings, 2015.

[16] Burr Settles. Abner: an open source tool for automatically tagging genes, proteins and other entity names in text, 2005.

[17] Charles Sutton, Andrew McCallum, et al. An introduction to conditional random fields, 2012.

[18] Erik F Tjong Kim Sang i Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition, 2003.

[19] Simon Tong i Daphne Koller. Support vector machine active learning with applications to text classification, 2001.

Entity Recognition and Classification for a Natural Language Database Interface

Sažetak

Zbog uvijek prisutne ljudske potrebe za razumijevanjem što većih količina podataka i rasta računalne snage baze grafova dobivaju na popularnosti. Još jedan korak k boljoj integraciji čovjeka i računala jest izrada jezičnog sučelja za komunikaciju s takvom bazom. U ovom radu razvijen je i ispitan sustav za izvlačenje entiteta iz upita u svrhu izgradnje takvog sučelja, pogotovo imenovanih entiteta te su u tu svrhu izgrađena četiri modela. Dobiveni rezultati su ohrabrujući, te pokazuju da se i na vrlo malom skupu podataka mogu naučiti modeli koji rade sa specijaliziranim upitima.

Ključne riječi: Obrada prirodnog jezika, Izvlačenje imenovanih entiteta, stroj potpornih vektora, stabla odluke, CRF, jezično sučelje, graf baza podataka.

Entity Recognition and Classification for a Natural Language Database Interface

Abstract

Because of the always present human need for understanding big quantities of data and computer power growth, graph databases are gaining in popularity. Another step in a better connection of human and the computer would be a natural language interface for communication with such a database. This thesis describes the development and evaluation of a entity extraction from queries, which is part of such natural language interface. Special attention was given to named entity extraction. Four models were built for this purpose. End results are encouraging, showing that even on small datasets models can be created to deduct from specialized queries.

Keywords: Natural language processing, Named entity extraction, support vector machine, decision trees, CRF, natural language interface, graph database