Erschienen in: Journal of Integrative Bioinformatics ; 9 (2012), 2. - 190
https://dx.doi.org/10.2390/biecoll-jib-2012-190

IDPredictor: predict database links in biomedical database

Hendrik Mehlhorn[1,*], Matthias Lange[1,*], Uwe Scholz[1] and Falk Schreiber[1,2]

[1]Leibniz Institute of Plant Genetics and Crop Plant Research (IPK), Gatersleben, Germany
{mehlhorn|lange|scholz}@ipk-gatersleben.de

[2]Institute of Computer Science, Martin Luther University Halle-Wittenberg, Halle, Germany
falk.schreiber@informatik.uni-halle.de

Summary

Knowledge found in biomedical databases, in particular in Web information systems, is a major bioinformatics resource. In general, this biological knowledge is worldwide represented in a network of databases. These data is spread among thousands of databases, which overlap in content, but differ substantially with respect to content detail, interface, formats and data structure.

To support a functional annotation of lab data, such as protein sequences, metabolites or DNA sequences as well as a semi-automated data exploration in information retrieval environments, an integrated view to databases is essential. Search engines have the potential of assisting in data retrieval from these structured sources, but fall short of providing a comprehensive knowledge excerpt out of the interlinked databases. A prerequisit of supporting the concept of an integrated data view is to acquire insights into cross-references among database entities. This issue is being hampered by the fact, that only a fraction of all possible cross-references are explicitely tagged in the particular biomedical informations systems.

In this work, we investigate to what extend an automated construction of an integrated data network is possible. We propose a method that predicts and extracts cross-references from multiple life science databases and possible referenced data targets. We study the retrieval quality of our method and report on first, promising results. The method is implemented as the tool *IDPredictor*, which is published under the DOI 10.5447/IPK/2012/4 and is freely available using the URL: `http://dx.doi.org/10.5447/IPK/2012/4`.

1 Motivation and Related Work

High throughput biotechnologies, like next generation sequencing, proteomics and metabolomics techniques produce a massive amount of data [1]. In order to support the functional annotation and to discover the functional context of lab data, the relationship to existing biological knowledge stored in world wide distributed databases has to be explored. Investigations in [2, 3] argue that web based, *in silico*, research is a major challange in life sciences. A study in [4] claims that 37% of all scientist use over 80% of their time for Web investigations. For this, the World Wide Web and all other HTTP-based services (web services, file downloads etc.) are the most popular media.

One use case is to facilitate Web information systems or desktop tools for linking wet-lab data to this world wide data cloud. Methods like sequence similarities such as BLAST [5], text

[*] Both authors contributed equally to this work.

similarities queries like Entrez text queries [6] or eTBLAST [7] are database search operations, which are commonly used in life science dry labs.

In this context, a general top-down approach is the use of biomedical search engines [8, 9]. In order to rank search results for their relevance, the most popular methods are the ranking field importance, word combinations, and term frequencies. Weighting and ranking search results on the amount (or importance) of cross-references is still an open issue for life science database integration.

In order to extract the Web content, the dry lab scientist use the following workflow:

1. **search the most relevant root data entries**: search databases using keyword search or match data pattern based on data similarities, e.g. sequence homology
2. **explore the root data surrounding knowledge**: follow hyperlinks manually, i.e. use the HTML hyperlinks or database cross references

Search in life science databases exhibits some fundamental differences from the way people search in the Web or in a general purpose digital library. First of all, links play a central role: not only a single article to a specific entity is of relevance, but all articles on this entity - though articles just mentioning this entity may be irrelevant. Second, life science databases are domain centric organized – usually around specific entity types (e.g. metabolomics). Here, is easy to extract all domain information related to an entity. But it is very difficult to collect information on an entity if the knowledge is spread across entities of different domains e.g. genome structure focused databases versus metabolite or pathway centric ones.

The exploitation of links across entities in different databases is impeded by the lack of an agreed-upon convention for referencing data records. Proprietary identifiers such as so called accession numbers are designed as unique combination of alphanumeric characters. For example, the proprietary identifier Q8W413 in the UniProt database [10] stands for the protein Beta-fructofuranosidase[1]. The enzyme 3.2.1.26[2] points to the same entry, but is interpreted as standard nomenclature for enzymes. In "The Arabidopsis Information Resource" (TAIR) we find the locus tag At2g36190[3] as identifier for the coding gene of the same protein in the species Arabidopsis thaliana (prefix *At*). Furthermore, the gene synonym AtFruct6 is an example for a semantically enriched acronym of an gene: *At* stand for Arabidopsis thaliana and *Fruct* for Beta-fructofuranosidase.

In this paper we discuss aspects that has to be faced by scientists doing data exploration. In particular, we present methods for cross reference extraction to retrieve linked information for any kind of database entities (enzyme, gene, protein etc).

Efforts and strategies for the integration of life science data is the research focus in several projects. Using several methods and approaches the majority is focused on particular analytic scenarios and applications, but none of them provided a holistic, uninterpreted view to the world wide interlinked knowledge. Popular techniques, like data warehouses, multi database query languages or database mediators provided powerful query interfaces and helpful tools [11].

[1] http://www.uniprot.org/uniprot/Q8W413
[2] http://www.expasy.org/enzyme/3.2.1.26
[3] http://www.arabidopsis.org/servlets/TairObject?type=locus&name=AT2G36190

The navigation through and the benefits of exploiting the data networks has been motivated in graph based systems. The three systems BioDBnet [12], BioNavigation [13] and BioGuideSRS [14] support scientists in exploring the content of integrated life science databases. Here, the core idea is a graph based browsing and query building on top of integrated databases.

BioXRef [15] is an example of a method that extracts cross-references from multiple life science databases by combining targeted crawling, pointer chasing, sampling and information extraction. There, the authors studied the retrieval quality of the proposed method and the relationship between manually crafted relevance ranking and relevance ranking based on cross-references.

To summarize, using the World Wide Web or social networks as inspiring example, the basic idea behind all those approaches is to compute a network of biomedical knowledge by taking a set of database entries as input, analyzing the entries and their attributes and identifying potential cross-references in the same and in other databases. In this contribution we propose *IDPredictor*, an algorithm that predicts cross-references from multiple life science databases and thus sets the basis for an enhanced information retrieval over biomedical data. We discuss to what extend *IDPredictor* can be used as method for an efficient and precise prediction of database cross-references. In Section 2 we present the underlying machine learning methods of *IDPredictor*. In Section 3 we discuss training methods and prediction performance measures. In Section 4 we discuss the prediction performance, prelimary results and the application to database networks.

2 Methods

2.1 Introduction to neural networks

A neural network, or more precisely an artificial neural network, is a graphical model motivated by biological neural networks. Neural networks can be used as an universal approximator being capable to learn patterns in a given set of data or to find complex relationships between sets of input data and output data [16, 17]. In the frame of this paper we use neural networks to learn and recognize patterns in the format of identifiers from biological databases.

Neural networks consist of nodes denoted as neurons, which are organized in consecutive layers, so that exclusively neurons of successive layers are connected by interconnections.The first layer is called the input layer comprising the input synapses, which receive the input of the model from a feature vector, which is a scalar vector with the same dimension as the number of input neurons. The last layer is called the output layer comprising the output synapses, which transmit the output of the model as the output vector. This output vector is being interpreted depending on the application of the neural network. The remaining layers are called hidden layers and connect the input layer with the output layer. Each neuron of the neural network receives a set of scalar values as input from the neurons of the previous layer or the feature vector and decides with the help of a given activation function how to fire a set of scalar values as output for the neurons of the subsequent layer or the output vector.

The parameters of a neural network are weights for all interconnections, which afford the adaption of the neural network to certain requirements. The general procedure of using neural networks starts with the training, where the weights of the interconnections are being adapted. Once the neural network is trained, it is ready to infer predictions.

Table 1: The ten proposed feature extraction strategies. We apply the five feature extraction methods *Position specific*, *Symbol specific*, *Symbol specific (partially grouped)*, *Symbol specific (grouped)*, and *Word statistics* either to the whole word string indicated by a 'No' in the second column, or to three word substrings individually indicated by a 'Yes' in the second column. In the third column, we present the dimension of the feature extraction vector.

Feature extraction approaches	Word substrings	Feature vector dimension
Position specific	No	30
Position specific	Yes	90
Symbol specific	No	256
Symbol specific	Yes	768
Symbol specific (partially grouped)	No	41
Symbol specific (partially grouped)	Yes	123
Symbol specific (grouped)	No	3
Symbol specific (grouped)	Yes	9
Word statistics	No	7
Word statistics	Yes	21

2.2 Feature extraction

In order to supply scalar input values for the input neurons of the neural networks, we extract features from words represented by numerical values. These numerical values constitute the elements of the feature vector. For there are different ways to represent the properties of identifiers, we test ten feature extraction strategies as summarized in table 1.

We represent words as a vector of symbols, which each have a unique ASCII code. We denote each ASCII code value as character type. In addition, we denote the identifiers from different databases as identifier type.

Feature extraction focused on positions

We apply a position specific feature extraction method in order to represent the property of many identifier types of being designed as a defined succession of symbols. For instance, TAIR identifiers such as 'At1g15850', 'AtMg01190', and 'ATCG01000' consist of the characters 'At' or 'AT' at the first two positions, a digit, 'M', or 'C' at the third position, 'g' or 'G' at the forth position, and five digits at positions five to nine. We keep the position information by extracting the character type of the symbol of each word string position. The dimension of the feature vector is fixed, thus we limit the position specific feature extraction method to a maximum of 30 positions and choose the feature vector to a dimension of 30. If a given word is longer than 30 positions, we ignore the remaining string positions and if the given word is shorter than 30 positions, we fill the supernumerous positions of the feature vector with zeros. In case of the position specific feature extraction method, neural networks have 30 input synapses, which are ordered and being supplied with the character type of the symbol of the according word position. We denote this feature extraction method as *Position specific*.

Feature extraction focused on symbols

We apply a symbol specific feature extraction method in order to represent the property of many

identifier types of comprising a specific set of characters. For instance, EC numbers such as '1.1.1.1', 'EC:3.4.24.56', and 'ec:6.4.1.-' exclusively consist of the set of characters {'.', '-', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ':', 'C', 'E', 'c', 'e'}. We represent the symbol information by extracting the character type of all word symbols and cumulating these in three ways to test different levels of granularity.

In the first level of granularity we create one feature vector element for each of the 256 possible characters types. In this level we count the character type abundance of all possible word symbols, thus this level is the most fine graned symbol specific feature extraction method. Here, the feature vector has the dimension 256 and the neural networks have 256 input synapses. We denote this feature extraction method as *Symbol specific*.

In the second level of granularity we create one feature vector element per digit, one feature vector element per letter, one feature vector element for each of the character types '-', '.', ':', and '|', and one feature vector element for all remaining character types. In this level we group all character types in one feature vector element, which are very uncommon within the identifers of the most types to limit the feature vector dimension as well as the number of input synapses of the neural networks to 41. We denote this feature extraction method as *Symbol specific (partially grouped)*.

In the third level of granularity we create one feature vector element for all digits, one feature vector element for all letters, and one feature vector element for all remaining character types. This is the most rough level, limiting the feature vector dimension as well as the number of input synapses of the neural networks to 3. We denote this feature extraction method as *Symbol specific (grouped)*.

Feature extraction focused on word statistics

We apply a word statistic specific feature extraction method in order to represent the property of many identifier types of being composed of well defined symbol proportions, symbol successions, and a fixed length. For instance, UniProt identifiers such as 'Q48HF8', 'E3E768', and 'A0MYM4' consist of two to four letters, two to four digits, and have a length of 6. We represent these word statistics by extracting the following seven features, which constitute the elements of the resulting feature vector. We extract the length of the word string (i), the propotion of letters (ii), the proportion of digits (iii), the proportion of whitespaces (iv), the proportion of the set of symbols {'-', '.', ':', '|'} (v), the proportion of all remaining characters (vi), and the number of letter-digit pairs, which stand together (vii). In case of the word statistic specific feature extraction method, the feature vector has a dimension of 7 and the neural networks have 7 input synapses. We denote this feature extraction method as *Word statistics*.

Feature extraction methods applied to word substrings

We apply the five presented feature extraction methods to word substrings in order to increase the position specificity of these methods. For instance, KEGG Gene identifiers such as 'ath:AT4G27720.1', 'ptm:GSPATT00025144001', and 'ani:AN2412.2' mainly consist of a prefix, a main part, and a suffix. In this case, the prefix represents a species code ('ath:', 'apn:', and 'ani:'), the main part represents the gene locus ('AT4G27720', 'GSPATT00025144001', and 'AN2412'), and the suffix represents a sequence version ('.1', '', and '.2'). We represent this identifier composition by splitting the given words in three substrings of preferably equal

Table 2: The six variables for the configuration of the neural network models. The variables *number of hidden layers* and *number of neurons per hidden layer* regard the neural network structure, the variable *type of activation function* regards the signal processing of the neural networks, the variables *number of identifiers per database* and *number of training cycles* regard the neural network training, and the variable *feature extraction strategy* regards the generation of the neural network input.

Variable	# of values	Set of variable values
# of hidden layers	2	$\{1, 2\}$
# of neurons per hidden layer	2	$\{10, 50\}$
Activation function	6	{gaussian, linear, log., sigmoid, sine, tanh}
# of training IDs per database	2	$\{100, 1000\}$
# of training cycles	2	$\{100, 500\}$
Feature extraction strategy	10	See table 1

lengths (in case of 'ath:AT4G27720.1': 'ath:A', T4G27', and '720.1') and applying the five proposed feature extraction methods to each substring individually. We combine the resulting three substring feature vectors consecutively to one feature vector, which size is three times the size of each substring feature vector. The number of input neurons of the neural networks is equal to the resulting feature vector dimension.

We apply the five introduced feature extraction methods to whole word strings as well as to word substrings. We denote the resulting ten strategies for the generation of feature vectors as feature extraction strategies (see table 1).

3 Case study

We test 960 neural network model configurations by combining the six variables *number of hidden layers*, *number of neurons per hidden layer*, *type of activation function*, *number of identifiers per database* for training, *number of training cycles*, and *feature extraction strategy* (see table 2). Subsequently, we describe the design of the case study in detail.

3.1 Data

We extract identifiers by parsing the dumps of the KEGG database (release: 02/2011) and the UniProt database (release: 10/2011) [18, 10]. From these identifiers, we select the $T = 51$ most abundant identifier types and prepare a identifier dataset with up to 1000 randomly choosen identifiers of each identifier type. The resulting data set comprises 51000 (identifier, identifier type) - pairs and is denoted as *identifier data set*.

We extract the vocabulary of the WordNet database and randomly choose 102.000 words [19]. We denote this set of words as *background data set*.

3.2 Neural network design

We choose a typical design of neural networks, where all neurons of the i-th layer are connected to all neurons of the $(i + 1)$-th layer. We choose the number of input neurons equal to the dimension of the feature vector. We choose the number of neurons per hidden layer each equal. We choose the number of output neurons equal to the number of identifier types, so that the t-th output neuron stands for the t-th identifier type, where $t \in [1, T]$.

In this study we test various neural network structures and activation functions in order to obtain a widespread view on the applicability of different combinations of neural network properties. We choose the variable *number of hidden layers* as 1 or 2, the variable *number of neurons per hidden layer* as 10 or 1000, and the variable *type of activation function* as gaussian, linear, logarithmic, sigmoid, sine, or hyperbolic tangent. The combination of these three variables yields in 24 different neural network designs.

3.3 Training and testing

We choose the well known backpropagation algorithm for the neural network training in order to adapt the neural networks to patterns within T identifier types. The backpropagation training is an supervised learning strategy, which follows an iterative approach trying to minimize the error of the neural networks output vectors for a given set of feature vectors [20]. We choose the variable *number of training cycles* as 100 or 500 in order to check for differences of the prediction quality for different training efforts.

For training and testing we select identifer sets of different sizes in order to examine the neural networks prediction quality as a function of the training set size. We choose the variable *number of identifiers per database* as 100 or 1000 and randomly select this many identifiers per identifier type from the *identifier data set*. We denote this set as *positive identifiers*. We randomly divide the identifiers of each identifier type from the *positive identifiers* set in 80% *positive identifiers for training* and 20% *positive identifiers for testing*. From the *background data set* we randomly select twice as many words as the size of the set *positive identifiers*. We denote this set as *negative identifiers*. We randomly devide this set in 80% *negative identifiers for training* and 20% *negative identifiers for testing*. We combine the set *positive identifiers for training* and the set *negative identifiers for training* to the *training set* and combine the set *positive identifiers for testing* and the set *negative identifiers for testing* to the *testing set*. For the training of the neural networks, we use the feature vectors from our ten feature extraction strategies applied to the identifiers from the *training set*. For the evaluation of the trained neural networks, we supply the identifiers of the *testing set* as input and compute for each identifier type the average and standard deviation of the p-value for each output neuron.

3.4 Result evaluation

We apply a scoring function in order to rank 960 combinations of the six variables *number of hidden layers*, *number of neurons per hidden layer*, *type of activation function*, *number of identifiers per database* for training, *number of training cycles*, and *feature extraction strategy*.

We compute the evaluation score for a certain variable combination by

$$\sum_{t_1=1}^{T}\sum_{t_2=1}^{T}\begin{cases}(T-1)\cdot p(t_1,t_2) & t_1 = t_2\\ -1\cdot p(t_1,t_2) & t_1 \neq t_2\end{cases}, \tag{1}$$

where $p(t_1, t_2)$ gives the average p-value of output neuron t_2 for the identifiers of type t_1 from the *testing set*. By this scoring function, we intend to equally reward the average p-value of the t-th output neuron for input of the t-th identifier type and penalize the average p-value of the remaining output neurons.

According to equation (1), the variable combination with the highest score is *number of hidden layers* $= 2$, *number of neurons per hidden layer* $= 50$, *type of activation function* $=$ sigmoid, *number of identifiers per database* $= 100$, *number of training cycles* $= 500$, and *feature extraction strategy* $=$ *Symbol specific* applied to word substrings. For a detailed evaluation of this variable combination see Appendix A.3.

4 Results and Discussion

4.1 Result statistics

We compute statistics for all values of all variables (see table 2) to get an overview. For each variable value, we compute the sum of scores of all variable combinations, where this variable value is present (see table 3 and table 4).

For the evaluation of different neural network structures, we vary the variables *number of hidden layers* and *number of neurons per hidden layer*. We find, that the average score of neural networks comprising 1 hidden layer is higher compared to neural networks comprising 2 hidden layers, but the standard deviation is also slightly increased. This is surprising, for the recognition power of neural networks comprising 2 hidden layers should be at least as big as that of neural networks comprising 1 hidden layer. We find, that the average score of neural networks comprising 50 neurons per hidden layer is significantly higher compared to neural networks comprising 10 neurons per hidden layers, but the standard deviation is increased as well. From the theory of neural networks, we expect a correlation of the recognition power with the number of neurons in the hidden layers.

The choice of the variable *type of activation function* is crutial for the routing of signals through the neural networks. We find, that the average scores of neural networks with different activation functions partitions in three fields. Neural networks with the well estrablished sigmoid activation function clearly outperform all neural networks with other activation functions. The second best activation functions are the hyperbolic tangent and the logarithm. The gaussian function, the linear function, and the sine function are inferior. However, the quantities of the standard deviations are arranged in an equal order.

For the evaluation of the impact of different training procedures, we vary the variables *number of identifiers per database* for training and the *number of training cycles*. We find, that the average score of neural networks trained with 100 and 1000 identifers per database is highly comparable, but the standard deviation of neural networks trained with 1000 identifers per database is slightly smaller. The higher standard deviation of neural networks trained with 100

Table 3: Evaluation of five variables for the training and design of neural networks. In the first column, we list the five variables. In the second column, we list the applied values for each variable by quantity or lexicographically. In the third and forth column, we present the average and standard deviation of the evaluation score of equation (1) normalized to a sum of 1 within each variable category and rounded to two positions after the decimal point.

Feature	Feature value	Rel. Average	Rel. std dev.
# of hidden layers	1	0.62	0.55
	2	0.38	0.45
# of neurons per hidden layer	10	0.40	0.40
	50	0.60	0.60
Activation function	gaussian	0.01	0.01
	linear	0.01	0.01
	logarithmic	0.24	0.27
	sigmoid	0.45	0.37
	sine	0.01	0.02
	tanh	0.29	0.33
# of training IDs per database	100	0.50	0.54
	1000	0.50	0.46
# of training cycles	100	0.41	0.42
	500	0.59	0.58

identifers per database might be caused by the higher chance of sampling identifiers from the training set, which are not representative. We find, that the average score of neural networks trainied for 500 training cycles is higher compared to neural networks trainied for 100 training cycles and that the standard deviation of neural networks trainied for 500 training cycles is lower compared to neural networks trainied for 100 training cycles. Thus, more training cycles make the predictions of the neural networks more reliable.

The evaluation of the variable *feature extraction strategy* is summarized in table 4. We find, that the feature extraction strategies *Symbol specific* and *Symbol specific (partially grouped)* both applied to whole word strings and applied to word substrings yield the best average scores and that all other feature extraction strategies are significantly inferior. In addition, the application of the feature extraction methods to word substrings increase the average scores in every case compared the same feature extraction methods applied to the whole word string.

4.2 Extraction of cross-reference from biomedical databases

As shown in section 3.3 we validated the prediction performance of *IDPredictor* using a set of database identifiers. In order to demonstrate the applicability of the method and to discuss possible drawbacks, we will subsequently show results of cross reference prediction in the UniProt database record CP20C_ARATH[4]. The UniProt Knowledgebase (UniProtKB) acts as a central hub of protein knowledge by providing an unified view on protein sequences and functional information. We choose this entry because it shows a high spectrum of different types of referenced databases and therefore comprises a representative number of syntactic variations of ID tokens. In particular, we saw identifiers from more than 100 databases and

[4] `http://www.uniprot.org/uniprot/P34791.txt`

Table 4: Evaluation of the variable *feature extraction strategy*. In the first and second column, we list the ten feature extraction strategies introduced in section 2. In the third and forth column, we present the average and standard deviation of the evaluation score of equation (1) normalized to a sum of 1 and rounded to two positions after the decimal point.

Feature extraction method	Word substring s	Rel. Average	Rel. std dev.
Position specific	No	0.00	0.00
	Yes	0.00	0.00
Symbol specific	No	0.20	0.20
	Yes	0.25	0.24
Symbol specific (partially grouped)	No	0.19	0.19
	Yes	0.26	0.22
Symbol specific (grouped)	No	0.02	0.03
	Yes	0.06	0.09
Word statistics	No	0.01	0.01
	Yes	0.02	0.03

because of the well structured format, it is relatively easy to manual validate the prediction quality. Nevertheless, our approach can be applied to any database record comprising plain text.

First of all, we used the best performing parameter configuration for the predictor network presented in section 3.4. This trained network was applied to the set of tokens, we extracted by decomposing the UniProt entry. Intuively, we used for decomposition the standard whitespace definition[5]. But programming language use not necessarily homogeneous definition of an whitespace. After draft inspection of the resulted token list, we saw obvious insufficiently decomposed tokens. Pattern for such problematic "super token" are key-value pairs like "*MEDLINE=20083487*", whereas some special characters act as "glue". Examples are `':'` or `'='`. Those key-value pairs are very database specific and would result in to much database specific ID pattern.

In consequence, we redefined the whitespace delimiter for biomedial data records. To do so, we counted the occurence of characters in true positive ID token in our training data. We found a high number of characters show up rarely (less than in 5%) in IDs. As shown in appendix A.2 there are 182 characters which do not show up in any ID, whereas 54 where selected in the range of frequently used 7-bit ASCII subset. Those are promising candidates for a redefinition of the so far used definition of whitespace. Using this new enhanced whitespace definition, the result was a set of 1,803 tokens.

As next step, we used the trained network to predict for each token the propability whether a it is an ID and if so, to which type of database it is probably linked. In order to evaluate the prediction quality, we inspected each of the token whether it is a real ID (true positive) or a false prediction (false positive). As result of the inspection, we tagged 118 tokens as real IDs and 1,685 as non ID tokens. The complete evaluation result is included in appendix A.1.

In order to test the performance of the ID token classification, we used different cut-offs for the p-value. We assume a minimal p-value of 0.5, which is a reasonable lower limit. This results in a maximum of 531 predicted identifier tokens. We end up with 184 for a cut off 1.0. The full

[5] `http://en.wikipedia.org/wiki/Whitespace_character`

prediction benchmark is summarized in Table 5.

Table 5: Prediction performance of the *IDPredictor* classifier for UniProt entry CP20C_ARATH. The columns cut-off, tokens, TP, TN, FP, FN, R and P are the p-value cut-off for the predicted ID tokens, the number of ID tokens has better p-value than the cut-off, the true positives, true negatives, false positives, false negatives, recall and precission

cut-off	tokens	TP	TN	FP	FN	R	P
0.5	531	91	1.154	440	27	0.77	0.72
0.6	520	88	1.165	432	30	0.75	0.73
0.7	494	85	1.191	409	33	0.72	0.74
0.8	425	81	1.260	344	37	0.69	0.79
0.9	362	77	1.323	282	41	0.65	0.82
1.0	184	67	1.501	117	51	0.57	0.92

We conclude, that a cut-off of 0.9 for the p-value of a predicted ID token is a good tread-off between to much wrong predicted IDs and missed IDs for the use case of the construction of database networks. We see a precission of 82% with a recall of 65%. In this config we miss 41 out off 118 manually confirmed IDs but predicted only 282 tokens erroneously as IDs. Practically, one would reduce those 282 false positives by query them in the predicted linked database. If the ID is not found, the token can be removed from the list of predivted IDs. In order to increase the stability of *IDPredictor* and to support a parctical application a for cross reference extraction, we suggest subsequently further enhancements.

4.3 Enhancements for cross-reference extraction

The prediction quality leads to the conclusion that the presented method is applicable in practice for an unsupervised cross reference extraction, if we consider the following enhancements:

1. **suboptimal tokenization**: Decompose text into tokens is the first step in presented id prediction pipeline. The result is a set of tokens. This task is error prone, especialy for non-natural language text. It could lead to "super" tokens, combining a sequence of stand alone words or may on the other hand fragment real tokens into sub-tokens. As consequence, the decomposition should use a smart definition of token delimiter as suggested in Section 4.2.

2. **ambiguous tokens**: Here we saw different semantics for similar token. The most prominent class of such tokens are numbers. Using the presented *IDPredictor* method it is hard to predict if a number is a numerical database ID or a measurement described in the text. For example, publication year or mol weight was matched as PubMed-ID. In general, those false positives were caused by ignoring the textual context, which could be implemented using text mining methods based on natural language processing or thesauri.

3. **very short tokens:** The obvious problem of tokens with a short length is the less information content. Thus, the neural network can not give a relieable ID prediction. An examples are abbreviations of authors first name, which result in a high number in 2 letter tokens. Thus we suggest to exclude tokens with a length less than 4 from token list and define them as stop words.

4. **raw data tokens:** Raw data, like sequence fragments or geometry data of protein structure are commonly mixed with textual content. Those raw data tokens share a lot patterns with IDs, e.g. non natural language words, special characters or high frequency of digits. To recognize whether a token is a part of a raw data section or not would help to avoid lot of false positives, but is challenging. One heuristic could be to check the frequency of predicted ID tokens in a sliding window. If we see in the window an accumulation of ID tokens compared to non ID tokens, we could assume a block of raw data.

5 Conclusion and Outlook

We presented *IDPredictor*, a method that predicts and extracts cross-references from biomedical databases. We motivated this approach by discussing advantages and drawbacks of existing approaches, like database integration, ID mapping services, scripting or regular expressions.

The presented method of training a neural network as token classifier was presented and manually benchmarked using an example from the UniProt protein database. For this data set we predicted possible cross-references and mapped these to the most probable source databases. We found and discussed false negatives and false positive predictions and presented enhancements to improve the prediction quality. We suggested improvements for text tokenization as well as rules to avoid ambiguous and false positve tokens predicted in raw data sections.

This contribution is intended to give a proof-of-principle of identifier prediction in biomedical databases using neural networks as classifier. Beside the presented application of cross-reference extraction, it would be possible to support more comprehensive scenarios in information retrieval. More use cases, an increased data basis for cross-reference extraction, and applications to other domains such as social networks are on the road map.

Because of the generality of *IDPredictor*, it is also possible to use this algorithm for detecting cross-references in non-life sciences databases. Another option is to apply *IDPredictor* to database management systems, e.g. a relational ones. Doing so, we could process the database schema by only tokenizing those attributes that contain cross-references. Further plans are to implement a user friendly and flexible tool and web service for link extraction over submited datasets as well as the extraction of comprehensive data networks by tokenizing the content of datacenters like the NCBI, EBI and GenomeNet. In this context, one typical use case would be the functional annotation of sequence data, the improvement of search engines as well as the structural analysis of data networks. Finally, an interesting use case of *IDPredictor* is to periodicaly compute a network of interlinked data that could be used to connect knowledge and bridge the gap of heteregeneous databases.

The tool *IDPredictor* is implemented in *JAVA*. A command line version is freely available for download and published (DOI:10.5447/IPK/2012/4)[6].

[6]For a direct download please use the URL: `http://dx.doi.org/10.5447/IPK/2012/4`.

Acknowledgements

We thank Thomas Münch for giving support for the execution of computation jobs at the IPK compute cluster *Brocken*. This work was supported by the European Commission within its 7th Framework Program, under the thematic area "Infrastructures", contract number 283496.

References

[1] M. Y. Galperin and X. M. Fernández-Suárez. The 2012 Nucleic Acids Research Database Issue and the online Molecular Biology Database Collection. *Nucleic Acids Research*, 40(D1):D1–D8, 2012.

[2] L. D. Stein. Creating a bioinformatics nation. *Nature*, 417(6885):119–120, 2002.

[3] R. Stevens, C. Goble, P. Baker, and A. Brass. A classification of tasks in bioinformatics . *Bioinformatics*, 17(2):180–188, 2001.

[4] A. Divoli, M. Hearst, and M. A. Wooldridge. Evidence for Showing Gene/Protein Name Suggestions in Bioscience Literature Search Interfaces. In *Pacific Symposium on Biocomputing*, volume 13, pages 568–579, 2008.

[5] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–3402, 1997.

[6] G. D. Schuler, J.A. Epstein, H. Ohkawa, and J.A. Kans. Entrez: Molecular Biology Database and Retrieval System. *Methods in Enzymology*, 266:141–161, 1996.

[7] M. Errami, J. D. Wren, J. M. Hicks, and H. R. Garner. eTBLAST: a web server to identify expert reviewers, appropriate journals and similar publications. *Nucleic Acids Research*, 35(suppl 2):W12–W15, 2007.

[8] Z. Lu. PubMed and beyond: a survey of web tools for searching biomedical literature. *Database*, 2011:baq036, 2011.

[9] M. Lange, K. Spies, J. Bargsten, G. Haberhauer, M. Klapperstück, M. Leps, C. Weinel, R. Wünschiers, M. Weißbach, J. Stein, and U. Scholz. The LAILAPS Search Engine: Relevance Ranking in Life Science Databases. *Journal of Integrative Bioinformatics*, 7(2):110, 2010.

[10] M. Magrane and UniProt Consortium. UniProt Knowledgebase: a hub of integrated protein data. *Database*, 2011:bar009, 2011.

[11] Z. Lacroix and T. Critchlow. *Bioinformatics: Managing Scientific Data*. Morgan Kaufmann Publishers, 2003.

[12] U. Mudunuri, A. Che, M. Yi, and R. M. Stephens. bioDBnet: the biological database network. *Bioinformatics*, 25(4):555–556, 2009.

[13] Z. Lacroix, K. Parekh, M.-E. Vidal, M. Cardenas, N. Marquez, and L. Raschid. BioNavigation: Using Ontologies to Express Meaningful Navigational Queries Over Biological Resources. In *CSB Workshops*, pages 137–138. IEEE Computer Society, 2005.

[14] S. Cohen-Boulakia, O. Biton, S. Davidson, and C. Froidevaux. BioGuideSRS: querying multiple sources with a user-centric perspective. *Bioinformatics*, 23(10):1301–1303, 2007.

[15] A. Bachmann, R. Schult, M. Lange, and M. Spiliopoulou. Extracting cross references from life science databases for search result ranking. In *Proceedings of the 20th ACM international conference on Information and Knowledge Management*, CIKM '11, pages 1253–1258, New York, NY, USA, 2011. ACM.

[16] E. Oja. A simplified neuron model as a principal component analyzer. *Journal of mathematical biology*, 15(3):267–273, 1982.

[17] T. Galla. Theory of Neural Information Processing Systems. *Journal of Physics A: Mathematical and General*, 39(14):3849, 2006.

[18] M. Kanehisa and S. Goto. KEGG: kyoto encyclopedia of genes and genomes. *Nucleic Acids Research*, 28(1):27–30, 2000.

[19] G. A. Miller. WordNet: a lexical database for english. *Commun. ACM*, 38(11):39–41, 1995.

[20] P. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, Cambridge, MA, 1974.

A Supplemental Material

A.1 Prediction performance for UniProt entry CP20C_ARATH

An evaluation of the *IDPredictor* ID classifier for the UniProt entry CP20C_ARATH can be found via the Digital Object Identifier (DOI) 10.5447/IPK/2012/9.
The columns are: extracted token, predicted database cross-reference, p-value, manual evaluation (1: true posive; 0: false positive). Overall we decomposed the entry into 1,803 tokens. The table comprises only those 531 tokens which had a minimal p-value of 0.5.

A.2 Distribution of ASCII characters in database IDs

The distribution of character frequencies in a sample of 51,000 IDs from 51 biomedical databases can be found via the Digital Object Identifier (DOI) 10.5447/IPK/2012/8.
The three columns show the ASCII code of the character, the printable form and the frequency of occurence. We hide the ASCII codes 128 - 255, because they never showed up in the sample IDs. Overall we see 182 characters do not show up in any ID (54 in the range of frequently used 7-bit ASCII subset)

A.3 Prediction performance for the variable combination with the best score

The detailed evaluation of the variable combination *number of hidden layers* $= 2$, *number of neurons per hidden layer* $= 50$, *type of activation function* $=$ sigmoid, *number of identifiers per database* $= 100$, *number of training cycles* $= 500$, and *feature extraction strategy* $=$ *Symbol specific* applied to word substrings can be found via the Digital Object Identifier (DOI) 10.5447/IPK/2012/0.