

# A Support Vector Machine Classifier for Gene Name Recognition

Steffen Bickel, Ulf Brefeld, Lukas Faulstich,  
Jörg Hakenberg, Ulf Leser, Conrad Plake, Tobias Scheffer  
Humboldt-Universität zu Berlin, Department of Computer Science  
Unter den Linden 6, 10099 Berlin Germany  
Corresponding author: hakenberg@informatik.hu-berlin.de

## Abstract

This summary describes our solution for task 1A of the BioCreAtIvE Challenge Cup 2003. Essentially, we reduce the entity recognition problem to the problem of classifying single words using a Support Vector Machine followed by a term expansion. Our research question is therefore to find those types of features that eventually yield the highest precision and recall. We implemented and evaluated different features and combinations of features, such as n-grams, neighborhood defined by a sliding window, classification results of preceding words, appearance of special characters or digits, or appearance of the word in a dictionary. Multi-word entity names are gathered in a context-sensitive post-processing step. Our best set of features on the training set leads to a precision of 71.4% and a recall of 72.8%, corresponding to an F-measure of 72.1%, for the closed division.

## 1 Introduction

Task 1A of the BioCreAtIvE challenge [4] is a *named entity recognition* problem where entities are *gene* or *protein names*. Compared to named entity recognition problems in other domains, we observe three characteristic difficulties: (a) Many gene names are “nondescript” or include nondescript terms, such as “fat”, “it”, or “ca” that frequently occur outside gene names as well. (b) A substantial amount of gene and protein names – mostly names that are derived from the chemical structure of the protein – occur only once in the corpus. (c) Many gene and protein names are artificial names, such as “p50”, “31kD”, or “D21P”.

We see the task as a problem of labeling elements  $w_t$  (tokens) of sequences  $w_1, \dots, w_n$  (sentences) with class values of  $y_t = -1$  (not a gene or protein) or  $y_t = +1$  (gene name). There can be strong dependencies between two values  $y_t$  and  $y_{t'}$  (in particular, when  $t'$  is in close proximity to  $t$ , i.e., the words are co-located), as well as between  $y_t$  and all  $w_{t'}$ . Our solution is based on a sliding window approach. For feature generation, we design a transformation  $f((w_1, \dots, w_n), t, y_{t-1})$  that maps a sentence  $w_1, \dots, w_n$ , a token position  $t$ , and the class label  $y_{t-1}$  of the preceding token to the feature vector  $x_t$  for word  $w_t$ . We create a training set  $\langle (x_1, y_1), \dots, (x_T, y_T) \rangle$  by applying this transformation  $f$  to every token position of every sentence of the annotated training corpus; the labels  $y_t$  correspond to the annotation (+1 for NEWGENE, -1 for untagged tokens).

## 2 A Support Vector Classifier

From the training set  $\langle (x_1, y_1), \dots, (x_T, y_T) \rangle$ , we learn a classifier  $c : x \mapsto y$  with  $y = \text{sign}(wx + b)$ , using the Support Vector Machine [9, 3] SVM<sup>light</sup> [5, 6] with linear kernel and default parameter settings to find parameters  $w$  and  $b$ . Using a Support Vector Machine is a natural choice because the attribute vectors are very high dimensional and sparse. The resulting sequence classifier is applied to new sentences by iteratively applying, from left to right,  $f$  to each token position  $t$ , and invoking  $c$  to obtain  $y_t$ . Our system furthermore invokes a set of context-sensitive post-processing rules that label additional tokens as genes. We focus our attention on engineering the transformation  $f$  – i.e., finding those features that allow for the best classification. We tested a large number of different features. Results for other features and their combinations will be described in the full paper.

Most features of  $x_t$  depend only on tokens  $w_{t-\lambda}, \dots, w_{t+\lambda}$  within a window of width  $2\lambda + 1$  – hence, the term “sliding window”. Context information is only contained in the feature  $y_{t-1}$ , in the dictionary

feature, and in the special feature “distance to the nearest keyword”. The latter is not limited to the window bounds. The post-processing step is also context-sensitive.

### 3 Feature Generation

Table 1: Feature classes used in the Vector Space Model representation of tokens. “\*”: feature classes used in the final system.

Feature	Example	Range	Short name
Token *	Sro7, where		Token
Unseen token *		{0, 1}	UToken
n-grams of token *	S, Sr, Sro, Sro7, ro, ..		2G, 3G, 4G
Previous & next tokens *			PToken, NToken
n-grams of tokens in window			2PG/2NG/3PG..
Special symbols *	ICAM-1		Spec
Figures *	p50, HSF1		Figure
Upper case letters *	In1C, GUS		Upper
Initial upper case *	Msp		Initial
Upper case (skip first) *	MSPRP2		Upper2
All chars are upper case *	MMTV		AllUpper
Characters and figures *	p50		CharFig
Long(est) consonant chain	dpp, Crc2		LCC
Keyword distance *		[0..1]	KeyDist
Dictionary, case sensitive *		[0..1]	Dict
Dictionary, case insensitive		[0..1]	DictIgn
Dictionary, compound part		[0..1]	DictCP
Prev./next token is NEWGENE		{0, 1}	PTG, NTG
Prev./next POS-tags		{NN,VB,JJ}	PPN/PPV/PPJ..

We define and test different classes of features that comprise our attribute vector (see Table 1). In the following, we describe the definitions of all classes.

**Token and unseen token** The attribute vector starts with all features from the class Token. For every word that occurs in the corpus, a feature is reserved, plus an additional feature indicating an unseen token (UToken). As every token in the training corpus is represented in the attribute vector, we needed a possibility to deal with unseen tokens we might encounter in the test corpus. UToken is a single boolean feature and states whether the current token was seen before or not. With a certain probability identified by cross-validation using the training and test corpora, we replace a “false” with “true” when computing the feature vectors for the training data. The system then finds some “unseen” tokens and learns how to react to those.

**n-gram of token** Many entities – even those that occur in the corpus only once – can be identified by considering properties of the name itself. Every possible letter n-gram is therefore an additional feature (from the class nG) – up to  $n = 4$ . The n-grams of size five and above did not improve results of the overall system. The value for each letter n-grams is determined by counting how frequently it occurs within the current window. This term frequency is weighted by the *inverse document frequency* whose definition we modified slightly to cover the frequency of letter combinations in words.

**Previous and next token** We triple the potential attribute vector space by defining features for each token from the corpus occurring as a token preceding (PToken) or following (NToken) the current token, respectively.

**n-grams in window** For all tokens in the window, we compute their respective weighted n-grams and add them to the attribute vector space as new features (2WG, 3WG, 4WG).

**Simple surface clues** As many gene names contain numbers, upper case characters and combinations of these, we define binary features for certain types of combinations. These are simple surface clues, and we check whether they apply to the token or not. A token can contain special symbols (Spec), figures (Figure), upper case letters (Upper), upper case letters not only at the beginning (Upper2), characters and figures (CharFig), or start with an upper case letter (Initial), or a token can consist completely of upper case letters (AllUpp). Additionally, we check for the longest consonant chain in a token (LCC), a possible int on “uncommon” terms. For examples, see Table 1.

**Keyword distance** An additional feature (KeyDist) measures the distance to the nearest keyword from a hand-crafted list of 25 terms, such as “receptor”, or “kinase”, deduced from the training corpus. The weight of this features is  $1/distance$  if a keyword was found, and zero otherwise.

**Dictionaries - some gazetteer features** For the closed division of task 1A, we build a dictionary using all gene names occurring in the `training` and `devtest` corpora. For the open division, we added additional gene names derived from synonym lists for four different frequently cited organisms (i.e., *Homo sapiens*, *Mus musculus*, *Saccharomyces cerevisiae*, and *Drosophila melanogaster*). Employing more synonym lists yields a higher recall value, but precision drops significantly by predicting too much false positives. Since all sentences of the training set contain only gene names that occur in the dictionary (unlike sentences outside the training corpus), we set the dictionary feature to zero, with a probability that we determined by cross validation, comparable to the “unseen” token feature. We use a suffix tree indexing scheme to identify whether a token is contained in a compound name phrase that is listed in the dictionary. Several dictionary features indicate the quality of a match, differing in their spelling sensitivity and case sensitivity (Dict/DictIgn). An additional feature checks whether the complete phrase a token appears in can be found in the dictionary (DictCP). As all dictionaries consist of lists of entities only, we refer to them as *gazetteers*.

**Tag of previous and next token** We try to make use of the information if a preceding (PTG) or following (NTG) token would be classified as a `NEWGENE`. In a first step, we classify a sentence token-by-token just as before, and in a second step do the same including the information on each token’s neighbors. To do so, we need two different classifiers, one that has learned to use this information, and one that has not. Additional features store the default part-of-speech tag for the previous and next tokens for nouns, verbs, and adjectives (e.g., PPN, PPV, NPV, NPJ).

## 4 Post-Processing for Name Expansion

Many gene names consist of multiple terms. We find that the SVM classifier recognizes these often incompletely – frequently, unseen nouns and unseen or nondescript adjectives are missing in predicted names. We employ a post-processing step in which we apply hand-crafted rules to expand names. Before these rules are applied, we invoke an instance of the Brill part-of-speech tagger [1] that has been trained on the training corpus to get the default POS tags. The rules refer to the tokens, their labels generated by the SVM, and their POS tags. As a rule of thumb, when a noun phrase contains a gene name, then the whole noun phrase is typically a gene name – except for a number of special nouns and adjectives which are never or seldom part of a gene name. We gathered these nouns and adjectives from the tagged training corpus by collocation analysis. Every word appearing more frequent as a part of a gene name than preceding a gene name (and vice versa) is included in a `NEWGENE` phrase. This led to the application of two exclusion lists for nouns - we expand a phrase, if the noun is none of 372 or 222 particular terms, respectively, (see Table 2). We decided to use a negative list, as we found much more nouns included in gene names than appearing directly before of after such a name. With adjectives, it is quite the opposite, though. Adjectives may be parts of gene names or qualify a gene. We expand a phrase to include a preceding or following adjective, on the other hand, if the latter appears in a list of 778 particular terms.

As an additional false positive filter, we remove 23 single words tagged as `NEWGENE`, that are seldom gene names when occurring by themselves. Examples for such words are “alpha”, “gene”, “mRNA”, and “subunit”.

Table 2: Rules used for the post-expansion step to switch certain part-of-speech tags to `NEWGENE` tags. We excluded 372/222 nouns, and included only 778 particular adjectives in the expansion of noun phrases. NN\* includes nouns, proper nouns, plurals; CD: cardinal digit; JJ: adjective.

Former POS pattern	Expanded pattern	Limitation
NEWGENE NN*	NEWGENE NEWGENE	exclude 372 nouns
NN* NEWGENE	NEWGENE NEWGENE	exclude 222 nouns
JJ NEWGENE	NEWGENE NEWGENE	include only 778 adjectives
NEWGENE JJ	NEWGENE NEWGENE	dito
NEWGENE DT NN*	NEWGENE NEWGENE NEWGENE	
NEWGENE CD	NEWGENE NEWGENE	
NN* / NEWGENE	NEWGENE NEWGENE NEWGENE	
NEWGENE / NN*	NEWGENE NEWGENE NEWGENE	

## 5 Results

We tested all feature classes and varieties of combinations. The set of the feature classes used in the final system is marked (\*) in Table 1. Surprisingly, most features invoking context information could not prove their presumed values. Even information on the (predicted) class of the neighboring tokens could not improve results.

On the provided `devtest` corpus, our system is able to obtain a precision of 71.4% and a recall of 72.8%, corresponding to an F-measure of 72.1%, for the closed division. By altering the absolute position of the learned discriminating hyperplane, we are able to shift precision or recall in either direction, see Figure 1. A recall of 100% can never be achieved, though. As shown in the figure, the plot drops down from 57%/81% to 30%/70% and then to 3%/0%. Shifting the hyperplane in the direction of the negative samples at a certain point results in too many terms predicted as positive samples. These terms then form gene phrases (at some point expanding over the whole sentence). The long phrase results in a false positive hit, and at the same time we miss all true gene names.

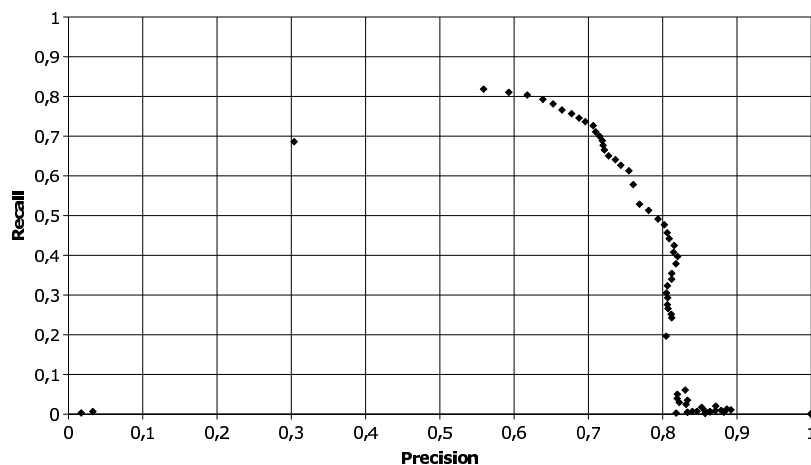


Figure 1: Precision/recall-curve obtained by manually shifting the hyperplane.

Final results for four runs led to precisions and recalls as shown in Table 3.

Table 3: Final results for four runs of the BioCreAtIvE Challenge Cup 2003.

Description	Precision	Recall	F-measure
Closed division, default settings	71.8%	70.6%	71.2%
Closed, manual hyperplane shift to increase precision	76.3%	61.8%	68.3%
Closed, invocation of dictionary lookup	69.8%	71.9%	70.8%
Open, dictionary consists of training data plus four organism specific synonym lists	72.2%	72.7%	72.4%

## 6 Discussion

Future plans to enhance our system include the implementation of an organism-specific version. It should reveal possible differences between usage of nomenclatures and syntactical rules given by them within the communities researching a particular organism. For instance, the yeast nomenclature is very compact and underlies a strict syntax (e.g., “YNL015W”). Gene names from the fruitfly, on the other hand, differ to a much greater extend and often consist of common terms (“a”, “wingless”, or “little imaginal discs”).

Assisting database curators with semi-automated systems would not in all cases require tagging each multi-gene name completely. We evaluated our SVM approach with a less strict recognition requirement.

If we consider as hits even parts of compound names, our system reaches a precision (on the `devtest` set) of 82.8% at 84.1% recall. Of course, other systems would probably benefit comparably from loosing this requirement.

As our approach more or less neglects the context gene names appear in, we tried to evaluate whether this approach can carry us any further or not. We therefore conducted an experiment. We sent seven different biologists a randomly selected list of 50 false negatives and 50 false positives from the hits generated by our classifier, asking each researcher to tag these pure (though possibly compound) names as either gene/protein, none-gene/protein, or unknown. We found that in only 45% of the cases human experts recognize the entity names as such, which is even worse than random drawing. This figure is almost the same for false negative and false positive hits. We also estimated the inter-annotator agreement and found that in only 11 out of 100 cases all annotators voted for the same answers.

This data gives us a slight evidence that our system already outperforms average biologists that are neither trained annotators nor experts in a special field. The inter-annotator agreement gives an idea on how good automated systems can or should become at all. Any method exceeding the degree of inter-annotator agreement is apparently tuned towards the particular group of annotators – which can be a good or a bad sign depending on the trust put into this group. However, our data can only give hints and is not representative enough to allow for global conclusions.

## Related work

The three most related projects to our approach are described in [2], [7], and [8].

GAPSCORE [2] scores single words based on a statistical model of gene and protein names that quantifies their appearance, morphology and context. The authors compared naïve Bayes, maximum entropy, and support vector machine classifiers, and found that the SVM outperforms the others slightly. Detected gene name candidates are extended to preceding and following tokens based on POS information. GAPSCORE achieved an F-measure of 57.6% on the Yapex corpus for exact matches, and 82.5% for sloppy matches.

The system described in [7] uses so called surface clues to detect potential protein name fragments, and additionally applies a false positive filter. A probabilistic model expands single names to compounds. For exact matches, the system achieves an F-measure of 63.6% on the Yapex corpus, and 81.4% for the sloppy evaluation.

[8] train a support vector machine on sets of different orthographic features to identify names belonging to the classes protein, DNA, RNA, and source, respectively. The best combined-class performance evaluated using a 10-fold cross-validation scored an F-measure of 74.23% on a corpus of 100 journal abstracts.

## References

- [1] Eric Brill. A simple rule-based part of speech tagger. In *Proc Conf Applied Natural Language Processing, ACL*, Trento, Italy, 1992.
- [2] Jeffrey T. Chang, Hinrich Schütze, and Russ B. Altman. Gapscore: finding gene and protein names one word at a time. *Bioinformatics*, 20(2):216–225, 2004.
- [3] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, Cambridge, 2000.
- [4] BioCreAtIvE Challenge Cup, 2003. <http://www.mitre.org/public/biocreative/>.
- [5] Thorsten Joachims. Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In *Proc Europ Conf Mach Learn*. Springer, 1998.
- [6] Thorsten Joachims. *Learning to Classify Text Using Support Vector Machines*. Kluwer, 2002. Dissertation.
- [7] Kazuhiro Seki and Javed Mostafa. A Probabilistic Model for Identifying Protein Names and their Name Boundaries. In *Proc Comp Sys Bioinf (CSB)*, 2003.
- [8] Koichi Takeuchi and Nigel Collier. Bio-Medical Entity Extraction using Support Vector Machines. In *Proc ACL 2003 Workshop on NLP in Biomedicine*, 2003.
- [9] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, 1995.