Combining term extraction and topic word detection for text categorization — a natural language processing perspective

Günter Neumann
Michael Kappes
DFKI
Saarbrücken, Germany

May 24, 2002

1 Introduction

The growth and dynamics of the Internet with respect to the huge amount of accessible natural language (NL) texts makes the exploration of methods based on the combination of natural language processing (NLP) and Machine learning (ML) a worthwhile venture. Coming basically from the NLP and computational linguistics community, a main interest of our research is concerned with the question about the impact of NLP methods on the performance of ML methods.

One domain of interest is automatic text categorization – the assignment of NL texts to one or more predefined categories. In (Neumann/Schmeier:99) we have already reported on first experiments concerning the combination of the shallow NL analysis of texts with the exploitation of tags and some knowledge about the domain at hand. Shallow NL analysis was performed with the system SMES, an information extraction core system for real world German text processing, (Neumann:97). We then used a number of different ML algorithms in order to check for the best combination of NLP and ML settings. Experiments were carried out on annotated German NL texts in two different industrial research projects using different linguistic information computed by SMES (i.e., morphological analysis, decomposition of noun compounds, chunk parsing). Despite the fact that we could show that for some ML algorithms the performance could indeed be improved when provided with linguistically analysed NL texts, our main result was that answers to questions such as how deeply the texts have to be analysed and how the ML algorithms must be parameterized are highly domain and data dependent. In fact – at least for us – finding the optimal parameter was often very difficult[1]. This also made the exact interpretation of the

[1]Note that determining the possibly optimal parameter settings is really necessary. It would make no

influence the used NLP methods had on the overall results not transparent. This means that although we found out that morphological analysis contributed to the improvement of the overall results, we cannot explain exactly why. Furthermore, even if we will chose only one ML algorithm it might be that the parameters have to be readjusted when different NL components are used for performing the NL analysis of texts.

Then, motivated by a paper from Jill Lehman, (Lehman:94), we started to re-consider text categorization from a linguistic perspective. From the results of this re-interpretation a simple new text categorization algorithm emerged, called SimpleCat which — we hope — is specifically suited for exploring the impact of different NLP methods on the task of text categorization. Before we describe SimpleCat in detail, we will first consider text categorization from a linguistic perspective.

2 A linguistic perspective on text categorization

Many NL tasks are already concerned with classification. For example morphological processing (among others) tries to assign a possible part-of-speech (POS) to a word form, word sense resolution tries to decide among the multiple possible meanings for a polysemous word, or sentence recognition tries to classify a sentence as being or not being part of the language defined by some grammar. A text categorization (TC) system tries to assign a category to a document. In the simple case a document is considered as consisting of a set of words (assuming word independence). Thus seen, a text category can also be viewed as consisting of a set of words. This is similar to the definition of a POS (e.g., noun or verb) by enumerating the possible word forms of the POS under consideration (e.g., defining a noun lexicon). If one is interested in exploring the kind of influence linguistic processing can have on the words of a category, then it is useful to consider the words from two perspectives:

- from a surface oriented perspective (morphological)
- from a content-based perspective (semantical)

This would be analogously to say that the POS noun bears syntactic as well as semantic properties. Thus, what we mean is that elements of a text can be interpreted either on the basis of their explicit textual representation (e.g., character sequences, word sequences) or through information that is only implicitly available in the words of a category (e.g., the relationship to other words). The explicit representation of a word could then be used as a means for discriminating between the different categories (or in other words, for defining the degree of category ambiguity of a word), where the implicit information could be used for determining the relevance of each word of some category, i.e., for determining which words are more typical for some category. Seen so we might assign two sorts of information to the words of a corpus:

sense to just use the default values of each ML algorithm because it might be the case that by accident the defaults of one ML algorithm is more suited for the current data and domain then the default values of the others. In that case no fair comparison would be possible.

- in how many different categories it occurs (i.e., its degree of ambiguity wrt. different categories)
- how typical it is for each assigned category.

Analogously we can define a category by means of two word lists:

- index list: the list of its most discriminating words
- topic list: the list of its most typical words

Given an annotated corpus, determination of the first information source is quite straightforward, and determination of the second source of information can be done on the basis of likelihood. New documents might then be classified according to a combination of this two sorts of information.

Now, why might this be so interesting for NLP? It would now be possible to explore which of the different lists is best be computed with what kind of NLP method. For example, it might be useful to use no NLP component for computing the index list, but a semantic analysis tool for computing the topic list.

3 The structure of SimpleCat

In the following section we will show how SimpleCat works on a given corpus. (It is now assumed that the corpus is already splitted into a training set and a test set of documents. For information about corpus processing, see 4.1.1 and 4.2.1)

The following basic steps are performed:

1. Training phase: Collecting data from the training set
2. Application phase: Application on the test set
3. Evaluation : Using Yang-schema to evaluate the performance

3.1 Training phase

The first step is to collect data from the training set. These data are used to compute category specific values which we need in the application phase.

Create word lexicon and category table. As mentioned above, each document and each category is seen as a set of independent words. A document *doc* is a tuple ($text$,cat), where *text* is the set of words in the given document and *cat* is the category the document belongs to. In most corpora there are documents which belong to more than one category. In these cases (e.g. *doc* belongs to cat_1 and cat_2) we multiply the tuple ($text$,cat_1,cat_2) to the tuples ($text$,cat_1) and ($text$,cat_2).

Two data-structures are built: word lexicons (INDEX-LEXICON and TOPIC-LEXICON) and a category table (CAT-TABLE). In INDEX-LEXICON and TOPIC-LEXICON words (tokens or lemmas) of the training set are keys and have word-specific values. Analogously, the keys of CAT-TABLE are the categories which occur in the training set. Its values are category specific data ('category profile'). Both tables (LEXICONS and CAT-TABLE) together define a bidirectional mapping of words and categories.

Creating LEXICON and CAT-TABLE simply means to count the occurences of words in specific categories, the number of documents in specific categories, etc. and then to compute relevant values with the collected data. These terms are entered into the corresponding data structure (LEXICONS or CAT-TABLE).

Entries of lexicons. INDEX-LEXICON and TOPIC-LEXICON have the same data-structure. But why do we use two lexicons? Because in training phase we are able to extract different information from the texts in the corpus, e.g. in several tests we simply took the tokens of the words in the INDEX-LEXICON and the lemmas of nouns and adjectives in TOPIC-LEXICON. Our aim is to show that index-words and topic-words provide different possibilities to extract relevant information from new documents.

Each entry $w \in$ LEXICON is of the form ($TotalTF(w)$,$ICF(w)$,$Cats(w)$), where $TotalTF(w)$ is the total number of occurence of w in the training set.

$Cats(w)$ is the list of different categories that have been assigned to the word w, and $ICF(w)$ a value that relates the number of all different categories in the training set ($TotalCatNum$) to the number of categories in which w occurs at least once: $ICF(w) = \log \frac{TotalCatNum}{|Cats(w)|}$ (Inverse Category Frequency) [2]. $ICF(w)$ is a measure for category ambiguity of w. Words with a low category ambiguity (equivalent to high ICF-value) are supposed to have a high discrimination value.

For each word-category-pair (w,cat) in LEXICON several further values are computed which are needed to create the word lists $IndexList$ and $TopicVector$ and to categorize documents in the application phase:

- $TF(w, cat)$: Occurence of w in cat. This value is needed later to compute the relative occurence of a word in a category $relTF(w, cat) = \frac{TF(w,cat)}{WordNum(cat)}$, where $WordNum(cat)$ is the total number of words in cat (see entries in CAT-TABLE).
- $DocF(w, cat)$: Number of documents of the category cat in which the word w occurs.

In ealier versions of SimpleCat the following values were part of INDEX-LEXICON ($TFICF$) and TOPIC-LEXICON (tpc). Now they are computed when index-lists and topic-vectors are created.

- $tpc(w, cat)$: The topicality of w in cat. This measure decides whether a word is proper for beeing put into the topic vector; it is proportional to both the relative

[2] We compute ICF in analogy to standard IDF-value, that relates the number of all documents in the training set to the number of documents in which w occurs at least once: $IDF(w) = \log \frac{TotalDocNum}{TotalDocF(w)}$

document frequency in *cat* and the relative term frequency of a word-category-pair: $tpc(w, cat) = \frac{TF(w,cat)}{WordNum(cat)} * \frac{DocF(w,cat)}{DocNum(cat)}$,
where $DocNum(cat)$ is the number of documents in this category (see CAT-TABLE).

- $TFICF(w, cat)$: Combination of $TF(w, cat)$ and $ICF(w)$. We use two measures:
  - ordinary : $TFICF = TF(w, cat) * ICF(w)$
  - smart : $sTFICF = 0.5 * (1 + \frac{TF(w,cat)}{T_{max}(cat)}) * ICF(w)$

  This measure is important for application of index-method. ($T_{max}(cat)$ denotes the frequency of the most frequent word in *cat*)

Entries of category table ('category profile'). This table contains category specific values and the word lists ($IndexList(cat)$ and $TopicVector(cat)$ which are still empty) for each category *cat* in the training set.

- $WordNum(cat)$: total number of words in *cat*
- $DocNum(cat)$: total number of documents which belong to this category
- $IndexList$: list of most discriminating words of *cat* (see below)
- $Index - T_{max}(cat) = max\{TF(w, cat)|w \in$ INDEX-LEXICON $\}$. This value is useful to compute frequencies in relation to the most frequent word.
- $TopicList$: list of topic candidates
- $TopicVector$: list of most typical words of *cat*. We call it vector because in the application phase (*TopicDistance* - method, see 3.2.3) this list is used similar to vectors in a vector space modell and because for all categories its length is limited by the same parameter.
- $Topic - T_{max}(cat) = max\{TF(w, cat)|w \in$ TOPIC-LEXICON $\}$.
- a,b,c,d : These parameters are used during application in order to perform automatic evaluation (see 3.3 and Yang:99).

Create index list. The $IndexList$ of a category contains all different words of this category except stop words (see below). In the application phase the words of $IndexList$ are weighted with different values. We are interested in those words of a category which have a high discriminating value and therefore can be used to find the most probable category for a new document.

Create topic vector. First, for each category cat a topic candidate list is built which contains potential topic words of cat. Like in the case of $IndexList$ we first filter stop words and then weight the remaining words with their topicality $tpc(w, cat) = \frac{TF(w,cat)}{WordNum(cat)} * \frac{DocF(w,cat)}{DocNum(cat)}$.

Topic words are supposed to be the most typical words for a given category cat. If the topicality is a relevant measure, it should be possible to limit the number of considered topic words for each category in the application phase. Hence, the parameter $n = TopVecLength$ denotes the length of the limited list of topic words of each category; this list is called $TopicVector(cat)$ and contains the n topic candidates of a category with the highest topicality value.

Stop words. Stop words are words which are supposed to be irrelevant for classification or which deteriorate the results (e.g. words which occur in all categories). We distinguish index-stop-words and topic-stop-words. There are several possibilities to filter stop-words; SC provides parameters for both index-stop-word-filtering and topic-stop-word-filtering (see SC-manual).

3.2 Application phase

3.2.1 General idea

Application is the process of assigning a category to a text of the test set of documents. In our approach, this step is very easy: Scan the new text. Built a table with all different words w_j and their frequencies (note that $DocTF(w_j) = DocTF_j$ is the frequency of w_j in a new document while $TF(w_j, cat)$ denotes the frequency of w_j in the category cat in the training set).

$$DocVector = \begin{pmatrix} w_1 & w_2 & \ldots & w_n \\ DocTF_1 & DocTF_2 & \ldots & DocTF_n \end{pmatrix}$$

This $DocVector$ is the representation of the new text. All methods of SimpleCat use this table to classify new documents.

SimpleCat is an m-ary classifier (see Yang:99); that means, for each new document it builts a ranking list of all categories of the training set. To obtain a binary categorization we have to use thresholding strategies which make possible YES/NO-decisions for each category in the ranking list. For the moment, we use $Rcut$ with n=1 (Yang:99), that means we only assign the best category in the ranking list. Reasons for proceeding this way are given below (3.2.4). Further work will be done to use $Scut$ for optimization.

To built a category ranking list for a new document two methods are used in application phase:

- $IndexList$-ranking : The $IndexLists$ of the categories are used to built a category ranking list $CatRank_{ind}$. This method is simply called *Index*.

- $TopicVector$-ranking : This method builts a ranking list $CatRank_{top}$ using $TopicVectors$ of categories. $TopicVector$-ranking has two variants:
  - *TopicCount*
  - *TopicDistance*

That means we obtain two ranking lists $CatRank_{ind}$ and $CatRank_{top}$ which we later combine to a resulting list $CatRank_{com}$.

3.2.2 IndexList-ranking

IndexList-ranking is the process of creating a category ranking list $CatRank_{ind}$ for a new document. It is based on the $IndexLists$ of the categories.

The resulting list is of the form

$$CatRank_{ind} = \begin{pmatrix} cat_1 & cat_2 & \dots & cat_{CatNum} \\ rank_1 & rank_2 & \dots & rank_{CatNum} \end{pmatrix}$$

where $CatNum$ is the number of categories in the training set.

To assign a category to a new document *doc* we run through $DocVector$. If $w_j \in IndexList(cat_i)$ we increment the rank of cat_i ($i \in \{1, \dots, CatNum\}$) depending on which parameters are set. We consider the following increments per appearance of w_j in *doc* : (parameter *inc* $\in$ {1,a,r,i,t})

- '1' : consider only recognition
- 'a' : absolute frequency of w_j in cat_i
- 'r' : relative frequency of w_j in cat_i
- 'i' : $ICF(cat_i)$
- 't' : $TFICF(w_j, cat_i)$
- 's' : $sTFICF(w_j, cat_i)$

For the rank of a given category $rank_i = rank(cat_i)$ we obtain

$$rank_i = \begin{cases} \sum DocTF(w_j) & inc =' 1' \\ \sum DocTF(w_j) * TF(w_j) & inc =' a' \\ \sum DocTF(w_j) * \frac{TF(w_j, cat_i)}{WNum(cat_i)} & inc =' r' \\ \sum DocTF(w_j) * ICF(cat_i) & inc =' i' \\ \sum DocTF(w_j) * TFICF(w_j, cat_i) & inc =' t' \\ \sum DocTF(w_j) * sTFICF(w_j, cat_i) & inc =' s' \end{cases}$$

for j$\in \{1, \dots, n\}$ and $w_j \in IndexList(cat_i)$, where n is the number of different words in *doc*.

This makes it possible to examine the influence of the different values on the precision of the result (see 4.1.2).

3.2.3 TopicVector-ranking

As mentioned above, another ranking list is built:

$$CatRank_{top} = \begin{pmatrix} cat_1 & cat_2 & \dots & cat_{CatNum} \\ rank_1 & rank_2 & \dots & rank_{CatNum} \end{pmatrix}$$

The ranks for the categories are found out by using the $TopicVectors$ and, to a certain extent, the related values in the LEXICON. We have tested two variants of $TopicVector$-ranking:

- *TopicCount*
- *TopicDistance*

TopicCount. This method is very simple; given a new document *doc*, the rank for a category cat_i is simply the sum of occurences of topic words in *doc*:

$$rank_i = \sum_{w_j \in TopicVector(cat_i)}^{j \in \{1,\dots,n\}} DocTF(w_j)$$

This seems similar to $IndexList$-ranking with parameter *inc*='1'. But there is an important difference; in this case, the word list $TopicVector$ is limited. Hence, the assumption is that only the selection of the best topic words will supply a pregnant category profile.

There is a disadvantage of this method: the rank of any category is always a discrete number, e.g. a document *doc* contains 15 topic words of cat_i and 15 topic words of cat_j. That means that $rank(cat_i) = rank(cat_j)$. If there are many categories (e.g. more than 90 in Reuters corpus, see below) the probability is high that there is more than one category with the same rank value. Nevertheless we made tests with TopicCount that yielded respectable results.

TopicDistance The aim of this method is to manage the problems of *TopicCount* without essential increment of expense and quantity of data.

To classify a new document, we have to decide which representation we choose for new documents and for the categories. In *TopicCount* we had the $DocVector$ and the $TopicVectors$ of the categories. For *TopicDistance* we use further values. For each entry w_j in $DocVector$ we compute the relative frequency of w_j in *doc*. Analogously, for each topic word tw_i in $TopicVector(cat)$ we compute the relative frequency of tw_i in *cat*. We assume that these values are comparable because in our approach both documents and categories are seen as sets of independent words.

Hence, in *TopicDistance* the representation of a document is a modified $DocVector$ and the representation of each category is a $TopicTFVector$ of the following form:

$$DocVector = \begin{pmatrix} w_1 & w_2 & \dots & w_m \\ relDocTF_1 & relDocTF_2 & \dots & relDocTF_m \end{pmatrix}$$

$$TopicTFVector(cat) = \begin{pmatrix} tw_1 & tw_2 & \dots & tw_n \\ relCatTF_1 & relCatTF_2 & \dots & relCatTF_n \end{pmatrix}$$

where $n = TopVecLength$ is the length of the $TopicVectors$ of categories and $m = DocVecLength$ is the length of $DocVector$ which is now limitable ($relDocTF_i = \frac{DocTF_i}{WordNum(doc)}, relCatTF_j = \frac{TF(tw_j,cat)}{WordNum(cat)}$).

We obtain the ranks of the categories in the ranking list by comparing $DocVector$ and $TopicTFVector(cat)$ for each $cat \in$ CAT-TABLE. We compute a kind of 'distance' between two vectors (not in a strict mathematical sense) resulting in low values for similar vectors and high values for different vectors:

$$rank_i = \sum_{w_j \in TopicTFVector(cat_i)}^{j \in \{1,\dots,m\}} (relCatTF(w_j) - relDocTF(w_j))^2 \quad (1)$$

$$+ \sum_{w_j \notin TopicTFVector(cat_i)}^{j \in \{1,\dots,m\}} relDocTF(w_j) \quad (2)$$

The first line of the formula is similar to euclidic distance; the omission of the square in the second line was found to be a better value than the strict euclidic distance. Because $0 \leq relTF \leq 1$ for all computed relative frequencies, the omission of the square means a harder 'punishment' for those categories whose $TopicTFVector$ does not contain the word w_j of the new document.

3.2.4 Classifying a new document

Classifying a new document means to assign one or more categories to a document. The category ranks of the previous section will be used to master this problem. After applying Index- and Topic- methods, we have two category ranking lists $CatRank_{ind}$ and $CatRank_{top}$ where $CatRank_{top}$ can be the result of *TopicCount* or *TopicDistance*.

Using a single ranking list. For evaluating the effectiveness of the single methods (*Index, TopicCount, TopicDistance*) it may be useful to perform categorization by using a single ranking list. In our experiments we used *Rcut* with $n = 1$ to select only the 'best' category. That means, given a new document, the category with the highest rating in the list is assigned. *Rcut* was used here to compare the results with those of combined ranking list.

Combining ranking lists. In order to yield better results we combined $CatRank_{ind}$ and $CatRank_{top}$ for *Index+TopicCount* and for *Index+TopicDistance*. Figures 1, 2 and 3 show the ranking lists of the three methods for one document of the Reuters corpus.

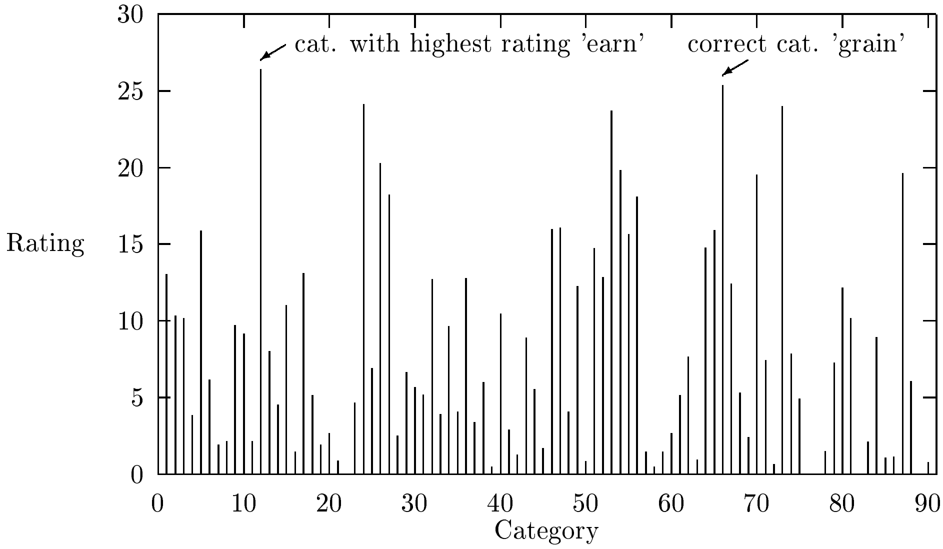

Figure 1: CatRank of Index-method

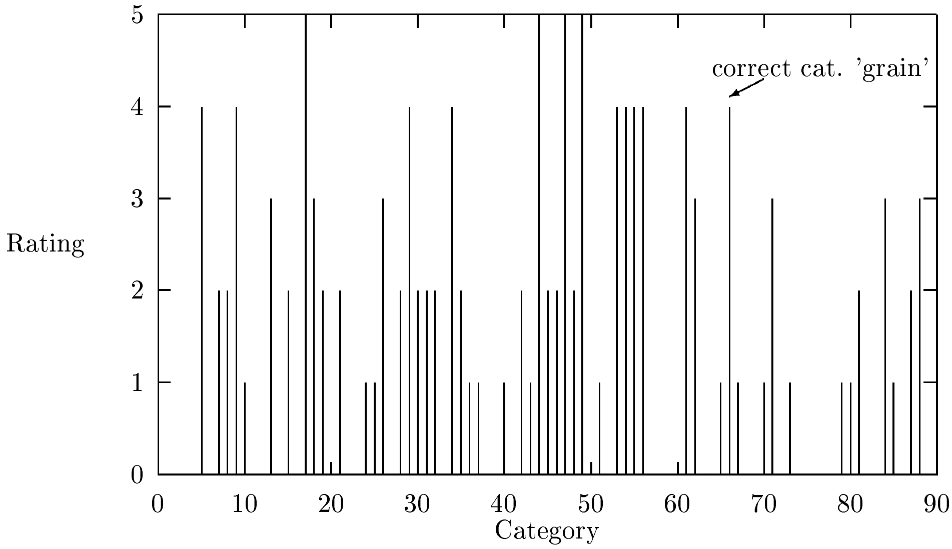

Figure 2: CatRank of TopicCount-method

As we can see, none of the methods finds the right category 'grain' (no.66). *Index* (fig. 1) assigns category 'earn' (no.12), *TopicCount* (fig. 2) finds four wrong categories (no.16,44,47,49) and would assign – in the case of $Rcut(n = 1)$ – one of them, *TopicDistance* (fig. 3) assigns the category 'sunseed' (no.44).

Although the right category 'grain' is not the best category in any method, it is rated

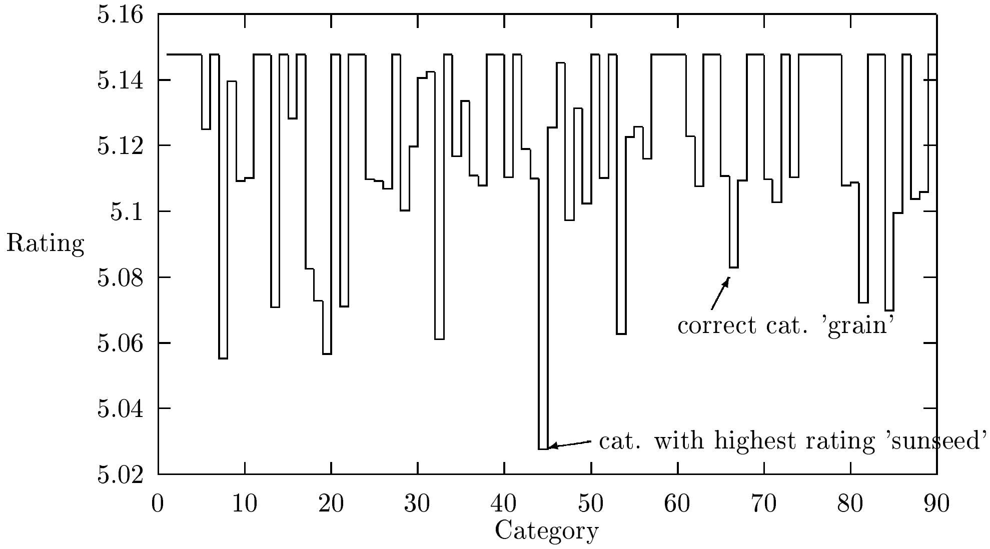

Figure 3: CatRank of TopicDistance-method

with a relative high value (that means a low value in *TopicDistance*) in all methods. The idea is to combine the wrong ratings to a correct one. Let us consider for example the lists of *Index* and *TopicCount*: we multiply all values in one ranking list with the same factor so that the highest value in each list is 100 (or any other fixed value). The values of the ranking lists are now comparable. For each category, we now add the multiplied values of $CatRank_{ind}$ and $CatRank_{top}$.

Another method is to normalize both ranking lists like vectors in a vector space (we have to compute both factors f_{ind} and f_{top} so that both ranking lists $rankind$ and $rank_{top}$ have the same length).

We yield the combined ranking list as follows:

$$CatRank_{com} = \begin{pmatrix} cat_1 & cat_2 & \dots & cat_{CatNum} \\ rank_1 & rank_2 & \dots & rank_{CatNum} \end{pmatrix}$$

with

$$rank_i = rank_{ind}(cat_i) * f_{ind} + rank_{top}(cat_i) * f_{top}$$

where f_{ind} and f_{top} are the factors mentioned above.

In the previous example this combination results in assigning the right category 'grain' to the document. Our tests have shown that the combination is a clear improvement of SimpleCat.

In a similar way we can combine $CatRank_{ind}$ and $CatRank_{top}$ for *TopicDistance*. Indeed, we first have to 'invert' $CatRank_{top}$ because in this case the category with the lowest value (that means the lowest 'distance' to the document in question) is the best.

Multiplying the ranking lists in the mentioned way has a consequence: the factors f_{ind} and f_{top} depend on the individual ranking of the new document. Therefore, given a distinct category cat_i, the ranks $rank_i$ for different documents are not directly comparable. This is the reason why we use *Rcut*. Because we were specially interested in finding out the effect of combining ranking lists on tracing out the best category, this was not a serious restraint, although we want to optimize the classification in further steps.

3.3 Evaluation of SimpleCat by Yang-Schema

For evaluation of our approach we use the schema proposed by Yang (Yang:99). For each category *cat* we built a table with four cells, where

- a counts the documents correctly assigned to *cat*
- b counts the documents incorrectly assigned to *cat*
- c counts the documents incorrectly rejected from *cat*
- d counts the documents correctly rejected from *cat*.

With these values we compute standard performance measures for each category:

- Recall: $rc = \frac{a}{a+c}$ if $a + c > 0$, otherwise undefined
- Precision: $pc = \frac{a}{a+b}$ if $a + b > 0$, otherwise undefined
- Fallout: $fo = \frac{b}{b+d}$ if $b + d > 0$, otherwise undefined
- Accuracy: $acc = \frac{a+d}{a+b+c+d}$
- Error: $err = \frac{b+c}{a+b+c+d}$
- F_1-measure: $f_1 = \frac{2*rc*pc}{rc+pc}$

To obtain global performance average, we compute all measures across categories. There are two methods to do this: macro-averaging means to compute first the measures per category and then to average across all categories. Micro-averaging is done by first building a global table with the cells a,b,c,d , where the values are the sum of the cells of all categories, and then to compute the performance measures.

Our optimization criterion is the micro-averaged F_1-measure.

4 Tests and results

4.1 Tests on Reuters-21578 corpus

4.1.1 Description of the corpus

For our experiments we used the Reuters-21578 corpus described in Yang/Liu:99. This is a refined version of Reuters Version 3 prepared by Apte. We obtain it from the original Reuters corpus by eliminating unlabelled documents and selecting only those categories which have at least one document in the training set and the test set. This results in a set of 10788 documents: 7770 training documents and 3018 test documents (Yang/Liu:99 have 10788 documents, 7769 doc's in training set, 3019 doc's in test set; we did not find the cause of the difference). The documents of the training set and the test set belong to 90 categories; each document has 1.3 categories on average. That makes it possible to use such a simple thresholding strategy as $Rcut(n = 1)$ and after all to yield acceptable results.

4.1.2 Results of single ranking methods

The examination of a single method makes it possible to examine the effects of parameter tuning on the single ranking lists.

IndexList-ranking. The following parameters were examined most extensively: Filtering thresholds to determine the index-stop-words (section 3.1), incrementation value of the category ranks (section 3.2.2) and normalization of all words to lower case.

We used different parameters to determine stop-words. Parameters in question were relative frequency of a word w in the training set $relCorpusTF(w)$, the $TFICF(w, cat)$ and the $ICF(w)$.

Words with a high $relCorpusTF(w)$ are e.g. articles and prepositions – that means usually words of the closed word classes. The disadvantage of $relCorpusTF(w)$ is that it contains no information about the distribution of w in different categories and therefore has no direct influence on the discriminating power of w. This was confirmed by our tests.

Another examined parameter is $TFICF(w, cat)$ where TF is the relative term frequency within cat (see above). Consequently, the $TFICF$-value indicates the distribution of w within cat. On the other side, we use the $TFICF(w, cat)$ in the $IndexList$-ranking (if parameter inc is set to 'i') so that, apart from this, words with a very low $TFICF$ do hardly have any influence on the ranking of a category in $CatRank_(ind)$ (that means filtering by $TFICF$ and ranking by $TFICF$ can be said to be redundant).

Finally, the $ICF(w)$ of a word w turned out to be the best threshold for filtering index-stop-words. This result agrees with our intuition that a category-external view ($ICF(w)$ is category-independent) is profitable for index word detection. Words whose ICF is lower than ICF-threshold (that means with a high category ambiguity) are declared as index-stop-words.

$IndexList$-ranking is performed by adding specific values to the ranks of the categories if a word w of a new document doc is in the $IndexList$ of the category in question (section 3.2.2). We will briefly discuss the different parameter settings.

If $inc ='1'$ we only consider the appearance of w in $IndexList(cat)$. In this case, the words are not weighted but only counted, e.g. to determine $CatRank(cat_i)$ we run through $DocVec$ and verify, if $w_j \in DocVec$ is contained in $IndexList(cat_i)$. If it is, we increment $CatRank(cat_i)$ by the frequency of w_j in the document; or in other words: for each category cat the rank is simply the number of index words of cat in the new document. This was done to examine the effect of weighting the words on the ranking of the categories.

After this, we weighted each index word w with absolute frequency, relative frequency, $ICF(w)$, $TFICF(w, cat)$ and $sTFICF(w, cat)$. $sTFICF(w, cat)$ was found to be the best weighting value. F_1-measure was 68,62%.

TopicCount-Ranking. The results of this simple method show that the selection of topic words with their topicality is a promising approach. After stop-word- removal (via ICF-threshold) we only varied the length of topic vectors (parameter $TopVecLength$).

Only to count topic words in a new document means not to weight each single topic word. The decision for one or another category only depends on selection criteria for topic words (topicality, $TopVecLength$, section 3.1). We tested the effects of $TopVecLength$ on micro-averaged F_1-measure and found a maximum at $TopVecLength = 325: F_1 = 71,68\%$. Longer vectors didn't improve the results.

TopicDistance-Ranking. *TopicDistance* differs from *TopicCount* in weighting each topic word with its relative frequency in the categories respectively in the new document. Moreover, the document vector is limitable (section 3.2.3).

The micro-averaged F_1-measure increases with longer TopicVectors. We varied $TopVecLength$ from 0,...,1000 and found the maximum at $TopVecLength = 975 : F_1 = 73,42\%$.

4.1.3 Results of combined ranking

Best result. Combined ranking means the combination of the two lists $CatRank_{ind}$ and $CatRank_{top}$ (section 3.2.4). The combination *Index+TopicDistance* brought our best results: microaveraged $F_1 = 78,84\%$.

Parameter settings:

- Combination *Index+TopicDistance*
- Increment-value for $IndexList$-ranking is sTFICF (smart version of TFICF, see section 3.2.2)
- $TopVecLength = 125$

- Thresholding strategy: $Rcut(n = 1)$

Note, that parameter optimization is a quite difficult task because the two ranking methods can't be optimized independently. The combination of two ranking lists makes the parameters of the single ranking methods (*Index* and *TopicDistance*) dependent on each other.

Discussion/Comparison. SimpleCat is a very simple approach to the categorization task. In this state of research we wanted to know if the used measures and values contain relevant information about the categories of the corpus. We took the Reuters corpus because many approaches were tested on this corpus. In particular, Yang:99 and Yang/Liu:99 describe several classifiers which are evaluated on Reuters corpus. The comparison with our results shows that the differences (in particular to NaiveBayes) are not crucial. Table 1 shows the performance results of several methods which were tested by Yang/Liu:99 (SVM = Support Vector Machine, KNN = k-nearest neighbor, LSF = Linear Least Squares Fit, NNet = Neural Network, NB = Naive Bayes). The last line shows the results of SimpleCat (SC). You find the complete output of test result of SimpleCat in appendix A.

method	miR	miP	miF1	maF1	error
SVM	.8120	.9137	.8599	.5251	.00365
KNN	.8339	.8807	.8567	.5242	.00385
LSF	.8507	.8489	.8498	.5008	.00414
NNet	.7842	.8785	.8287	.3765	.00447
NB	.7688	.8245	.7956	.3886	.00544
SC	.7118	.8834	.7884	.4323	.0053

Table 1: Comparison of Results on Reuters Corpus

Table 2 shows the top twenty topic words of four categories.

The results of SimpleCat are encouraging because our main interest is the effect of linguistic pre-processing of the topic words on the categorization result. Besides, we see several possibilities for improvements, e.g. optimization of combination of ranking lists, optimization of thresholding strategy and further parameter tuning. In section **??** we will describe our further purposes.

Especially our linguistic purposes led us to another corpus.

4.2 Tests on BPA corpus

4.2.1 Description of the corpus

This corpus contains news wire stories from several news agencies which arrived the Bundespresseamt (BPA) in one week (July, 16. - 22., 1995). We only considered German

no.	earn	acq	coffee	orange
1	vs	shares	coffee	orange
2	cts	stock	quotas	juice
3	net	share	ico	frozen
4	shr	offer	brazil	fcoj
5	loss	group	producers	brazil
6	qtr	acquisition	meeting	agriculture
7	profit	co	talks	citrus
8	revs	stake	bags	estimate
9	share	common	quota	florida
10	note:	buy	producer	duties
11	record	merger	organization	final
12	4th	sale	brazilian	crop
13	dividend	acquire	delegates	oranges
14	quarter	shareholders	colombia	duty
15	prior	sell	president	gallons
16	div	securities	consumers	season
17	shrs	investment	exporters	brazilian
18	avg	outstanding	brazil's	october
19	stock	terms	negotiations	usda
20	earnings	bank	executive	based

Table 2: Reuters Corpus: top 20 topic words of 4 categories

texts and yielded 8817 documents. The categories are persons and institutions of the German government, e.g. the foreign ministry, or departments of BPA. An incoming news story can be distributed to 96 possible categories; 74 of them are contained in our corpus ($C_{BPA} = \{c_1, \ldots, c_{74}\}$). If the human distributor declares a news as unimportant or irrelevant it will not be distributed to any of the categories. That means, we have to filter unimportant documents; the category $NoCat$ will contain those documents. In this corpus we have 2886 documents which belong to $NoCat$.

In the Reuters corpus each document has 1.3 categories on average (section 4.1.1). The number of categories per document in BPA corpus is 8,98 on average. Figure 4 shows on the x-axis the number x of assigned categories, on the y-axis the number of documents which belong to x categories.

Many documents belong to 10 or even more categories. About 100 documents have 57 (!) categories. Besides, the categories of BPA corpus seem to be highly overlapping, e.g. cat_i is the chief of a department, cat_j is a special section of the department etc. Moreover, an examination of documents and their assigned categories showed that the assignment not only depends on the semantic content of the document but also on organizational hierarchies in the government and in BPA or perhaps on special interests of the receivers.

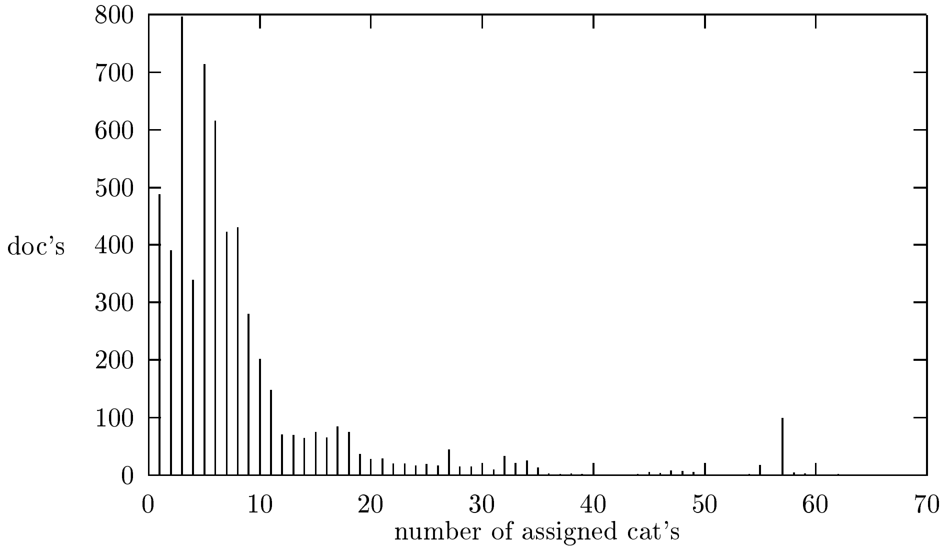

Figure 4: Number of doc's in relation to the number of assigned cat's (without mapping)

4.2.2 A new category mapping

We looked for such categories which we thought to build a reasonable set of new categories. We found that the set of ministries should be a good choice, because

- we supposed a connection between the documents sent to a special ministry and their semantic content.
- a number of 16 ministries is easily surveyed.
- the number of categories per document is clearly lower than in the original distribution (fig. 5; after the mapping each document has 1,69 cat's on average)
- each category contains a lot of documents which should be an advantage for our probabilistic approach.
- the set of ministries is less overlapping than the original categories of BPA corpus.

This new distribution is obtained as follows: for each document of the corpus we map its set of assigned categories on the set of assigned ministries. The destination category set is $C_{min} = \{M_1, \ldots, M_{16}, NoMin, NoCat\}$, where $M_1, \ldots, M_{16}$ are the ministries, $NoMin$ contains documents which are not sent to any ministry and $NoCat$ those documents which are not assigned to any category.

If this distribution leads to proper results, it would be possible to ease the task of a human distributor who has to decide where to send the incoming news.

In further steps, we could try to find reasonable distributions for the other categories in C_{BPA}, e.g. special departments of BPA.

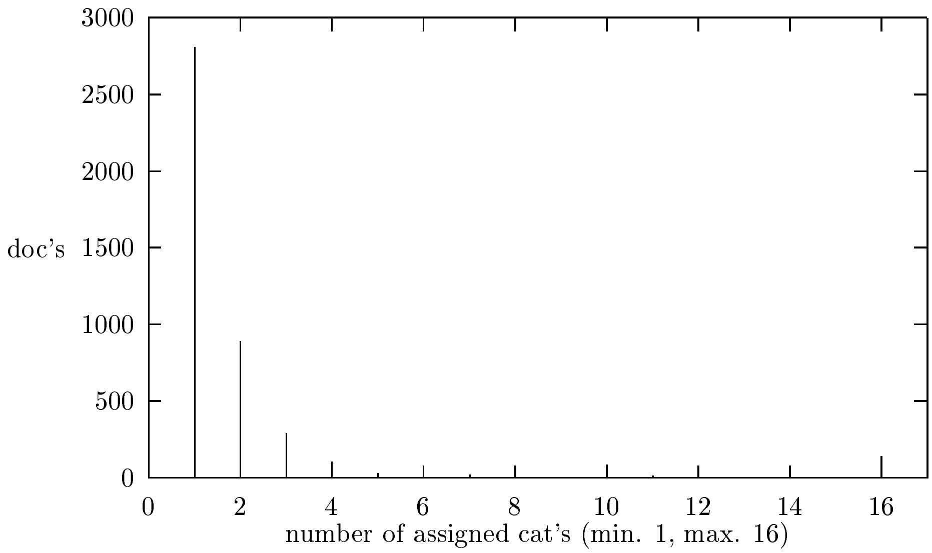

Figure 5: Number of doc's in relation to the number of assigned cat's (mapping 'ministries')

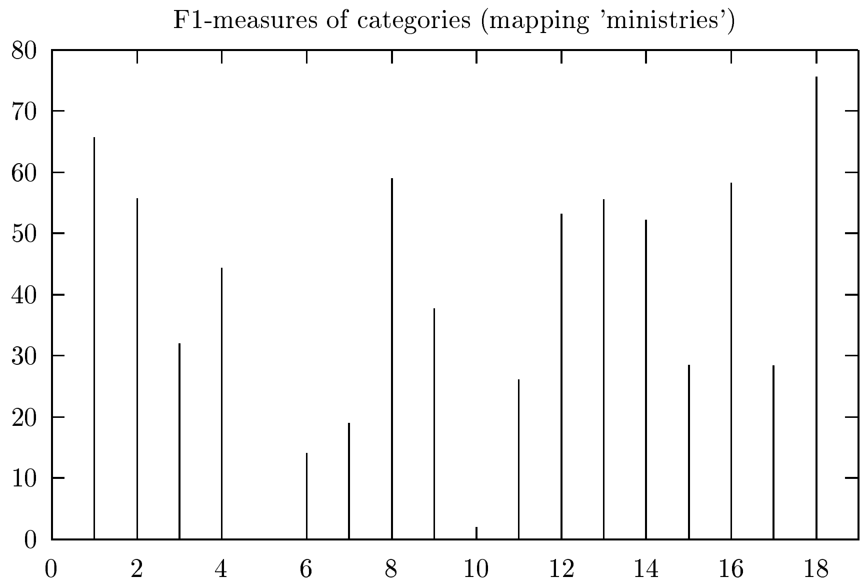

Figure 6: F1-measures of categories (mapping 'ministries')

4.2.3 Results

Several tests were performed with the category set C_{min}. Fig.6 shows F_1-measures per category $(1 = M_1, \ldots, 16 = M_{16}, 17 = NoMin, 18 = NoCat)$.

Against our assumption, the results don't supply a homogenous distribution. The F_1-

measures of categories M_5 (ministry of agriculture) and M_{10} (postal ministry) are irrelevant. The reason for this result is not necessary a bad choice of category set. It may be that the mentioned ministries do hardly become the best category in the ranking list; because we are using $Rcut(n = 1)$ we only select the best category to be assigned to the new document. M_5 and M_{10} usually occur in combination with other categories; probably in this case the other categories dominate M_5 and M_{10}. Consequently, the next step is to find adequate thresholding strategies for binary classification.

Document filtering. Furthermore, we observe that $NoCat$ has the highest F_1-measure. This led us to the assumption that $NoCat$ can be used for document filtering. Document filtering means categorization with two non-overlapping categories: Cat and $NoCat$. Those documents which get the category $NoCat$ will not be considered for further categorization process; they are supposed to be irrelevant. If Cat is assigned to a document, this document will be further categorized. In a first test, 93% of the decisions were right without additional parameter tuning.

Best topic words. In spite of low F_1-measures for the ministries, we want to take a look at the topic vectors of some ministries. Intuitively, they seem to be a good profile for the categories. The table below shows the top 20 of topic words for four ministries (in application we use about 100 or more topic words per category).

As mentioned above, the difficulty with BPA corpus is that not only the content of a message is decisive for its receiver; other criteria in question: competences, (personal?) interests, etc. In addition, the categorization (done by a human classifier) seems to be handled rather liberal - more receivers are better than forgetting an important one. Therefore, the 'boundaries' between categories are more fuzzy than in Reuters corpus.

Before this background, we think that the topic vectors (table 3) are a propper representation of the categories. Note, that BPA corpus is a collection of news stories of one week. Therefore, the topic words represent the actual topics of a category.

Interpretation of index words and topic words. As mentioned above, we suppose that $IndexList$-ranking corresponds to a category-external view while $TopicVector$-ranking corresponds to a category-internal view. That means that the words in $IndexList(cat)$ should be as discriminating as possible (surface-oriented) and that $TopicVector(cat)$ should contain the most typical words of the category cat (content-oriented).

As a first test for our assumption, we examined the effect of normalizing all characters of the texts to lower case (LC). We performed these tests for $IndexList$-ranking and for $TopicVector$-ranking (table 4). The effect should be more evident with German texts. Thus, we performed it on BPA corpus.

These results confirm our assumptions; the decrease of F_1-measure in *Index* is significant. The effect on *TopicCount* is moderate but our aim is to perform further linguistic processing on the corpora for topic detection.

no.	M_2 : economics	M_4 : labour	M_{12} : health	M_{13} : justice
1	vw	bauarbeiter	zahnaerzte	kurden
2	dasa	kurden	staatskommissar	auslieferung
3	minus	hbv	kzvn	verdaechtigen
4	basf	versicherungen	medizin	untersuchungshaft
5	daimler-benz	simsek	forscher	enderle
6	ford	schlechtwettergeld	hiller	genc
7	enderle	asylrecht	schirbort	verhaftet
8	kirch	betriebsrat	krankenhaeusern	turnoff
9	versicherungen	baustellen	jung	simsek
10	jokisch	ueberbrueckungsgeld	ignaz	kaution
11	geschaeftsjahr	abitur	behinderte	schneiders
12	ferdinand	oecd	beitragsstabilitaet	gerichts
13	hoechst	schmalz-jacobsen	abberufung	basf
14	lohr	auszubildenden	wahlmoeglichkeiten	inhaftierten
15	betriebsrat	baugewerbes	erreger	angeklagte
16	hbv	becksteins	lundberg	leeson
17	ruf	bregger	kassenzahnaerztlichen	asylrecht
18	gold	ingenieur	pflegesaetze	william
19	fixing	behinderte	mediziner	bestrafung
20	japanischen	tarifparteien	spender	instanz

Table 3: BPA Corpus: top 20 topic words of 4 categories

	no LC	LC
Index	55,28%	51,08%
TopicCount	43,78%	44,70%

Table 4: BPA Corpus: Effect of normalization to lower case

4.3 Tests on Medical Corpus

We used another corpus to show the impact of linguistic processing. Our medical corpus contains about 7500 abstracts of different medical disciplines. Each of the 38 disciplines is seen as a category for our classifier. There is a german and an english version of this corpus; thus, we are able to compare the results of each version. There are two formats available: ASCII-texts and XML-annotated texts.

4.3.1 Tests with ASCII-texts

First, we executed tests on ASCII-texts to find out the impact of the language (viz. the amount of inflection) on the results. We found, that english topic words lead to better results than german topic words (fig. 7). This backs our assumption that a linguistic processing for german texts could improve the results for topic words classification (method: *TopicCount*). We suppose that a reduction of inflection via linguistic processing (e.g. by only taking lemmas) increases the 'semantic density' of topic vectors.

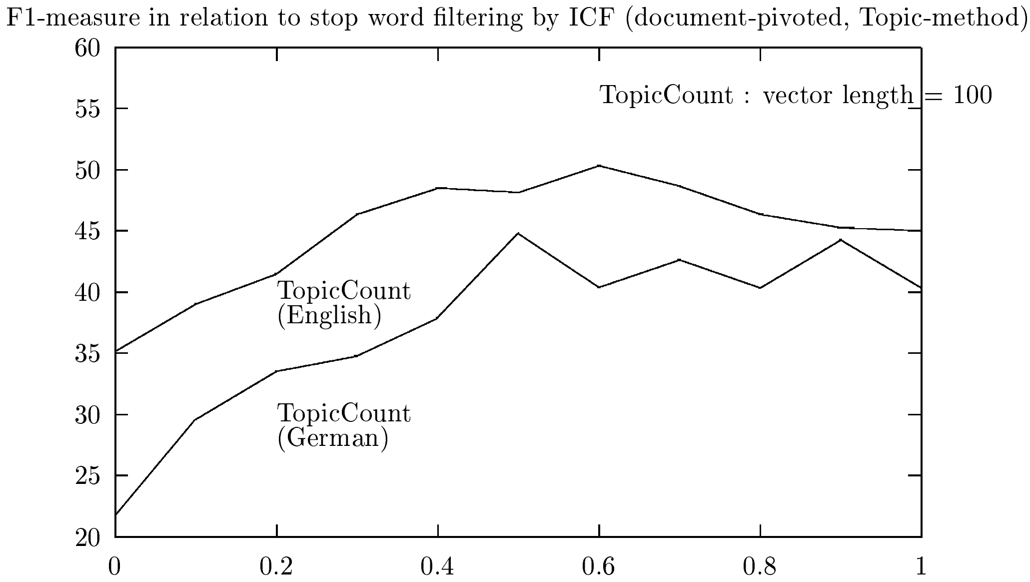

Figure 7: Topic-method: english vs. german texts

A comparison of index word classification (*Index*-method) shows an inverse result. German index words lead to better results than english index words. A reason could be the higher discrimination value of german index words because of higher inflection of the German language (fig. 8).

The following list shows top-20 index- and topic-words of one category ('heart'), extracted from german and english version (ASCII-version) of our corpus (without any linguistic processing but normalization to lower case for topic words).

German version

Kategorie : herz

Nr.	Indexwords	Topicwords
1	Kardiomyopathie	kardiomyopathie
2	ARVD	herzkrankheit
3	hsp60	dilatativer
4	Enteroviren	herzinsuffizienz
5	Herzkrankheit	koronarer
6	dilatativer	koronare
7	Atherosklerose	atherosklerose
8	koronare	herzerkrankung
9	1-adrenergen	koronaren
10	Herzmuskelerkrankung	myokard
11	VEGF121	t-lymphozyten
12	Riesenzellmyokarditis	entz\"undlichen
13	Coxsackie-Viren	ptca
14	Herzinsuffizienz	kardiomyopathien
15	PTCA	nichtinvasiven
16	Kardiomyopathien	bildgebung
17	Aortenerkrankungen	kardiovaskul\"are

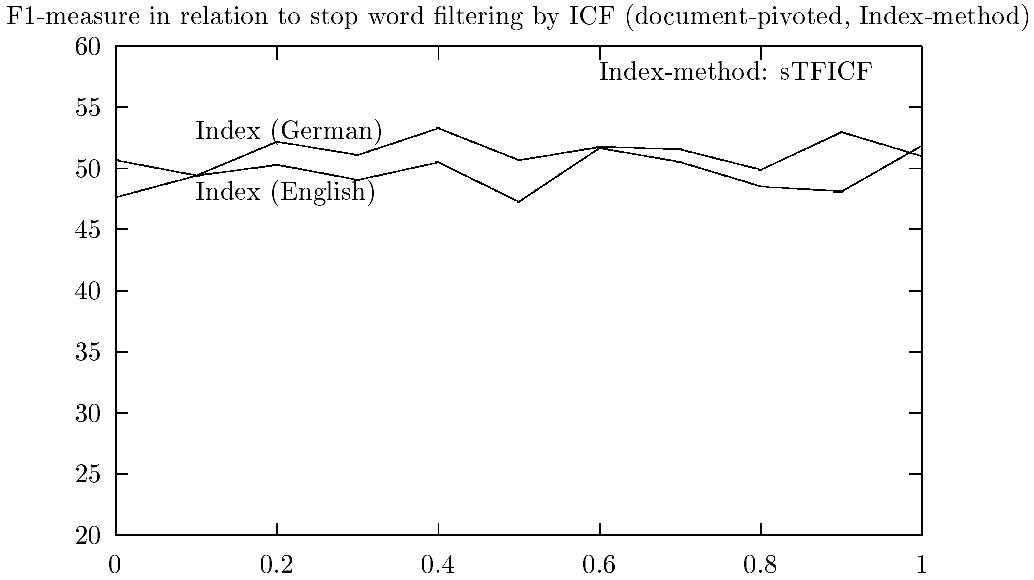

Figure 8: Index-method: english vs. german texts

18	VEGF165	enteroviren
19	koronarer	enthalten
20	Herzerkrankung	proteine

English version

Kategorie : herz

Nr.	Indexwords	Topicwords
1	myocarditis	myocarditis
2	coronary	atherosclerosis
3	ANT	cardiomyopathy
4	ARVD	myocardium
5	TMR	angina
6	VEGF121	ventricular
7	atherosclerosis	t
8	cardiomyopathy	dilated
9	VEGF	cytokines
10	restenosis	angiogenesis
11	ventricular	macrophages
12	VEGF165	revascularization
13	angina	noninvasive
14	phosphocreatine/ATP	familial
15	Myocarditis	signal
16	angiogenesis	viral
17	CAD	ventricle
18	myocardium	plaques
19	myocardial	inheritance
20	fibrofatty	restenosis

4.3.2 Tests with XML-annotated texts

The texts are XML-annotated; they contain the tokens of the texts, the corresponding lemmas and the parts of speech. Further informations are: chunks, grammatical and semantic relations. First of all, we only use tokens, lemmas and parts of speech for our tests.

A first question is: which parts of speech will contain relevant information for classification? And: where are the differences between index words and topic words?

In the tests with Reuters Corpus and BPA Corpus we used ICF-measure to filter stop words; that means, the discriminating power of a word decides about its usage in a word list (index words or topic words). Now we examine the impact of stop word filtering by part-of-speech.

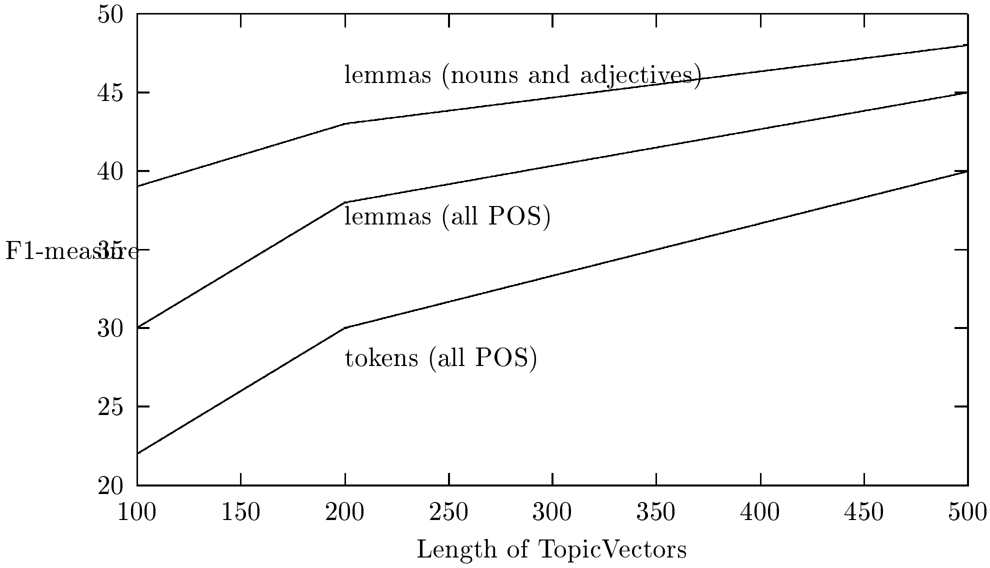

Figure 9: F1-measure in relation to TopicVector-Length (stop word filtering by POS; method: TopicCount)

Fig. 9 shows that filtering by part-of-speech seems to be a worth while attempt. But this result only shows that stop-word-filtering could be done *either* by statistical filtering *or* by 'linguistic' filtering. What we want to find out is that linguistic processing brings about an *additional* improvement of classification results.

Thus, the next step will be an optimization of statistical filtering for *Index* and *TopicCount* method. After that, we will execute an additional filtering by part-of-speech.

Optimization of statistical stop word filtering To optimize stop word filtering we extract the tokens of the texts (all POS) and filter stop words by ICF. After that, we extract lemmas (only nouns and adjectives) and filter by ICF. We expect a clear improvement of results. More: if we have found an optimized ICF-threshold x for tokens-extraction, we suppose that linguistic preprocessing (viz. extraction of lemmas/nouns&adjectives) will improve the result for the same ICF-threshold x.

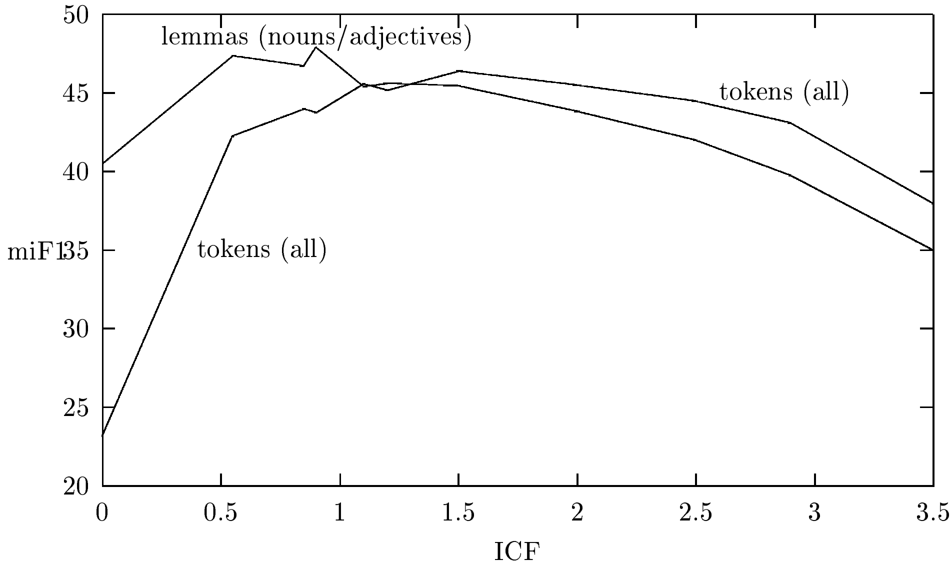

Figure 10: miF1 in relation to stop word filtering by ICF (topic-vector-length 100)

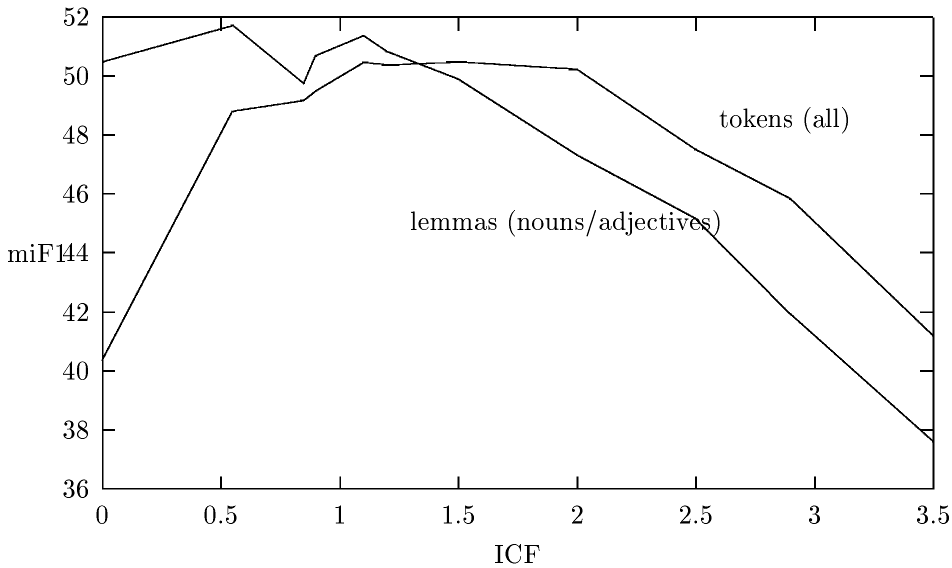

Figure 11: miF1 in relation to stop word filtering by ICF (topic-vector-length 500)

As we can see (fig. 10 and 11), the best ICF-threshold for token-extraction (all POS) is $ICF = 1.5$ (for vector length = 100 *and* 500). Against our expectation, extracting lemmas of adjectives and nouns does not clearly improve results for ICF-threshold=1.5. On the other hand, we get the best results with a combination of ICF-filtering and linguistic preprocessing (lemmas of adjectives and nouns). For topic-vector- length 100 we get the best result with an ICF-threshold of 0.9: miF1=47.89% (compared to best result without any linguistic processing: miF1=%46.39). For a vector-length of 500 the best miF1 is 51.71% with an ICF-threshold of 0.55 (fig. 11).

Discussion. We attain best results with a combination of statistical stop-word-filtering via ICF and 'linguistic' filtering (viz. extracting lemmas of nouns and adjectives). A look at fig. 10 shows that best miF1 is attained at ICF=1.5 for statistical filtering; with combined filtering we attain best miF1 already at ICF=0.9. Similar situation in fig. 11: best miF1 with statistical filtering at ICF=1.5; with combined filtering best miF1 is higher and attained already at ICF=0.55. That means, a weaker statistical filtering combined with a linguistic filtering supplies better results than an optimized statistical filtering.

Why does additional linguistic filtering (lemmas of nouns and adjectives) only bring a moderate improvement of results? First, we filter stop words via ICF; that means, we declare those words as stop words which have a high category ambiguity. Then, we only take lemmas of those words in our topic lexicon which are either nouns or adjectives.

Which words have a high category ambiguity? Words of closed word classes (articles, conjunctions, adverbs etc.) have a very high category ambiguity; so they will occur in almost each category. That means that nouns and adjectives (open word classes) will be the category-specific words. Consequently, the class of words with a low category ambiguity (high ICF-value) and the open word classes (in our case: nouns and adjectives) are highly overlapping classes; so, the effect of additional linguistic filtering is moderate. In further tests we want to back this assumption.

Effect of linguistic filtering on *Index-method*. The following tests were executed:

- extracting all tokens ⇒ miF1=51,35%
- all lemmas ⇒ miF1=53,55%
- tokens of nouns and adjectives ⇒ miF1=55,65%
- lemmas of nouns and adjectives ⇒ miF1=54,63%

We obtain best result with tokens of nouns and adjectives; worst result with taking all tokens without any filtering. In spite of no filtering, the results with *Index-method* are much better than tests with *TopicCount-method*. The reason is the different ranking of Index- and Topic-words. Topic words are ranked by *topicality* ($DocF * TF$), where Index words are ranked by *TFICF*. Consequently, without any filtering topic-vectors will contain many words of closed word classes (e.g. articles, conjunctions) because these words are words with both a very high document frequency and a high term frequency. On the other hand, $TFICF$-value of those words is not very high because of their high category ambiguity (their occur in almost all categories); so their ICF is low ($ICF = 0$ if a word occurs in all categories). Then, in spite of no statistical filtering the values of such ambiguous words are low in index-lists; that means their contribution to the result is almost irrelevant.

So, we expect a low effect of statistical filtering on results with *Index-method* (fig. 12).

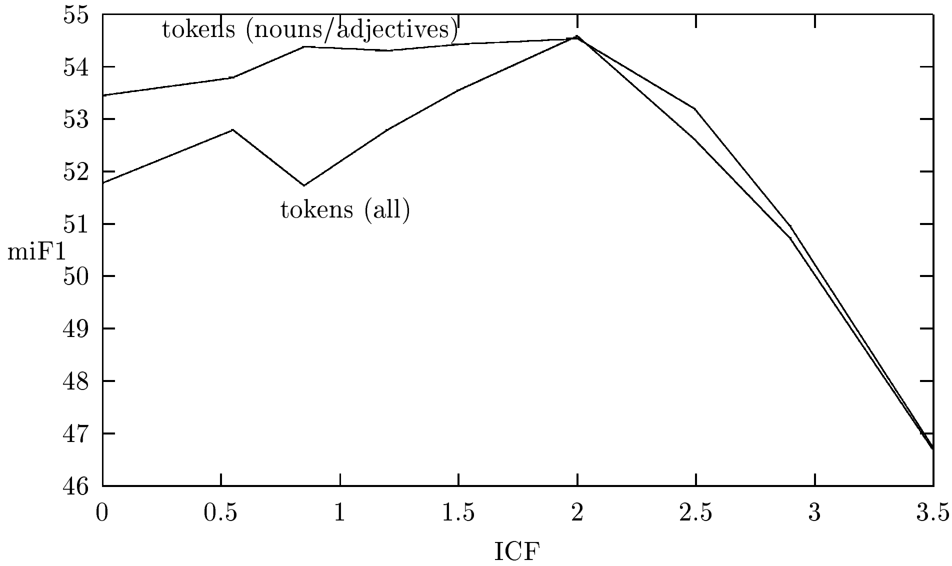

Figure 12: miF1 in relation to stop word filtering via ICF (index-method)

4.3.3 Combination of Index- and TopicCount-method

A combination of *Index-method* and *TopicCount-method* brings $miF1 = 59,02$
The parameters were set as follows:

- method: *Index+TopicCount*
- topic-vector-length: 500
- settings of *Index-method*: extraction of tokens (nouns and adjectives); stop-word-filtering via ICF=1.5
- settings of *Topic-method*: extraction of lemmas (nouns and adjectives); stop-word-filtering via ICF=0.55

5 References

Lehman, J.F.: Toward the essential nature of statistical knowledge in sense resolution. In: *AAAI* 1, 1999. pp.734-741

Neumann, G.: 1997

Neumann, G., S. Schmeier: 1999

Yang, Y.: An evaluation of statistical approaches to text categorization. In: *Information Retrieval* 1-2(1), 1999. pp.69-90

Yang, Y., X. Liu: A re-examination of text categorization methods. In: *Proceedings of SIGIR-99, 22nd ACM International Conference on Research and Development in Information Retrieval.* Berkeley: 1999.

A Complete output of performance test

```
Cat=Category WordNum=Number of words rc=recall pc=precision fo=fallout acc=accuracy err=error F1=F1 measure

 a= num of docs in which this cat was one of the correct and was assigned so
 b= num of docs in which this cat was not one of the correct but was assigned so
 c= num of docs in which this cat was one of the correct and wasn't assigned so
 d= num of docs in which this cat was not one of the correct and wasn't assigned so


Cat: _cocoa          WordNum: 14503 rc: 77.78% pc: 93.33% fo:  0.03% acc: 99.83% err:  0.17% F1: 84.85%  DEBUG: a=14 b=1 c=4 d=2999
Cat: _interest       WordNum: 57699 rc: 60.31% pc: 87.78% fo:  0.38% acc: 97.91% err:  2.09% F1: 71.49%  DEBUG: a=79 b=11 c=52 d=2876
Cat: _tin            WordNum:  5792 rc: 58.33% pc:100.00% fo:  0.00% acc: 99.83% err:  0.17% F1: 73.68%  DEBUG: a=7 b=0 c=5 d=3006
Cat: _strategic-metal WordNum:  2700 rc:  0.00% pc:  0.00% fo:  0.03% acc: 99.60% err:  0.40% F1:  undef  DEBUG: a=0 b=1 c=11 d=3006
Cat: _wheat          WordNum: 40790 rc: 25.35% pc: 85.71% fo:  0.10% acc: 98.14% err:  1.86% F1: 39.13%  DEBUG: a=18 b=3 c=53 d=2944
Cat: _pet-chem       WordNum:  3579 rc: 33.33% pc:100.00% fo:  0.00% acc: 99.73% err:  0.27% F1: 50.00%  DEBUG: a=4 b=0 c=8 d=3006
Cat: _copra-cake     WordNum:   591 rc:  0.00% pc:  undef fo:  0.00% acc: 99.97% err:  0.03% F1:  undef  DEBUG: a=0 b=0 c=1 d=3017
Cat: _oat            WordNum:  1590 rc: 16.67% pc:100.00% fo:  0.00% acc: 99.83% err:  0.17% F1: 28.57%  DEBUG: a=1 b=0 c=5 d=3012
Cat: _rapeseed       WordNum:  2812 rc: 44.44% pc:100.00% fo:  0.00% acc: 99.83% err:  0.17% F1: 61.54%  DEBUG: a=4 b=0 c=5 d=3009
Cat: _carcass        WordNum: 10265 rc: 27.78% pc: 83.33% fo:  0.03% acc: 99.54% err:  0.46% F1: 41.67%  DEBUG: a=5 b=1 c=13 d=2999
Cat: _coconut        WordNum:   434 rc: 25.00% pc:100.00% fo:  0.00% acc: 99.90% err:  0.10% F1: 40.00%  DEBUG: a=1 b=0 c=3 d=3014
Cat: _earn           WordNum:278201 rc: 97.06% pc: 97.50% fo:  1.40% acc: 98.05% err:  1.95% F1: 97.28%  DEBUG: a=1055 b=27 c=32 d=1904
Cat: _meal-feed      WordNum:  5713 rc: 10.53% pc: 50.00% fo:  0.07% acc: 99.37% err:  0.63% F1: 17.39%  DEBUG: a=2 b=2 c=17 d=2997
Cat: _wpi            WordNum:  3533 rc: 70.00% pc:100.00% fo:  0.00% acc: 99.90% err:  0.10% F1: 82.35%  DEBUG: a=7 b=0 c=3 d=3008
Cat: _sorghum        WordNum:  7114 rc:  0.00% pc:  0.00% fo:  0.13% acc: 99.54% err:  0.46% F1:  undef  DEBUG: a=0 b=4 c=10 d=3004
Cat: _sun-oil        WordNum:   586 rc:  0.00% pc:  undef fo:  0.00% acc: 99.93% err:  0.07% F1:  undef  DEBUG: a=0 b=0 c=2 d=3016
Cat: _rice           WordNum:  7252 rc: 12.50% pc:100.00% fo:  0.00% acc: 99.30% err:  0.70% F1: 22.22%  DEBUG: a=3 b=0 c=21 d=2994
Cat: _soy-meal       WordNum:  3324 rc:  7.69% pc:100.00% fo:  0.00% acc: 99.60% err:  0.40% F1: 14.29%  DEBUG: a=1 b=0 c=12 d=3005
Cat: _palmkernel     WordNum:   605 rc:  0.00% pc:  0.00% fo:  0.03% acc: 99.93% err:  0.07% F1:  undef  DEBUG: a=0 b=1 c=1 d=3016
Cat: _income         WordNum:  2087 rc: 57.14% pc: 66.67% fo:  0.07% acc: 99.83% err:  0.17% F1: 61.54%  DEBUG: a=4 b=2 c=3 d=3009
Cat: _sun-meal       WordNum:   135 rc:100.00% pc:100.00% fo:  0.00% acc:100.00% err:  0.00% F1:100.00%  DEBUG: a=1 b=0 c=0 d=3017
Cat: _jet            WordNum:   447 rc:  0.00% pc:  0.00% fo:  0.07% acc: 99.90% err:  0.10% F1:  undef  DEBUG: a=0 b=2 c=1 d=3015
Cat: _hog            WordNum:  2367 rc: 33.33% pc:100.00% fo:  0.00% acc: 99.87% err:  0.13% F1: 50.00%  DEBUG: a=2 b=0 c=4 d=3012
Cat: _trade          WordNum: 96204 rc: 66.38% pc: 61.11% fo:  1.69% acc: 97.08% err:  2.92% F1: 63.64%  DEBUG: a=77 b=49 c=39 d=2853
Cat: _alum           WordNum:  5670 rc: 43.48% pc:100.00% fo:  0.00% acc: 99.57% err:  0.43% F1: 60.61%  DEBUG: a=10 b=0 c=13 d=2995
Cat: _sugar          WordNum: 28486 rc: 66.67% pc: 88.89% fo:  0.10% acc: 99.50% err:  0.50% F1: 76.19%  DEBUG: a=24 b=3 c=12 d=2979
Cat: _gnp            WordNum: 31606 rc: 74.29% pc: 70.27% fo:  0.37% acc: 99.34% err:  0.66% F1: 72.22%  DEBUG: a=26 b=11 c=9 d=2972
Cat: _groundnut      WordNum:  1297 rc:  0.00% pc:  0.00% fo:  0.03% acc: 99.80% err:  0.20% F1:  undef  DEBUG: a=0 b=1 c=5 d=3012
Cat: _soy-oil        WordNum:  3298 rc:  0.00% pc:  0.00% fo:  0.03% acc: 99.60% err:  0.40% F1:  undef  DEBUG: a=0 b=1 c=11 d=3006
Cat: _zinc           WordNum:  3461 rc: 23.08% pc:100.00% fo:  0.00% acc: 99.67% err:  0.33% F1: 37.50%  DEBUG: a=3 b=0 c=10 d=3005
Cat: _l-cattle       WordNum:  1898 rc: 50.00% pc:100.00% fo:  0.00% acc: 99.97% err:  0.03% F1: 66.67%  DEBUG: a=1 b=0 c=1 d=3016
Cat: _ipi            WordNum:  8864 rc: 58.33% pc:100.00% fo:  0.00% acc: 99.83% err:  0.17% F1: 73.68%  DEBUG: a=7 b=0 c=5 d=3006
Cat: _lead           WordNum:  2448 rc:  0.00% pc:  0.00% fo:  0.03% acc: 99.50% err:  0.50% F1:  undef  DEBUG: a=0 b=1 c=14 d=3003
Cat: _barley         WordNum:  5037 rc: 42.86% pc: 85.71% fo:  0.03% acc: 99.70% err:  0.30% F1: 57.14%  DEBUG: a=6 b=1 c=8 d=3003
Cat: _heat           WordNum:  2681 rc: 60.00% pc: 60.00% fo:  0.07% acc: 99.87% err:  0.13% F1: 60.00%  DEBUG: a=3 b=2 c=2 d=3011
Cat: _gold           WordNum: 16409 rc: 76.67% pc: 71.88% fo:  0.30% acc: 99.47% err:  0.53% F1: 74.19%  DEBUG: a=23 b=9 c=7 d=2979
Cat: _nickel         WordNum:  1268 rc:  0.00% pc:  undef fo:  0.00% acc: 99.97% err:  0.03% F1:  undef  DEBUG: a=0 b=0 c=1 d=3017
Cat: _jobs           WordNum:  9261 rc: 52.38% pc:100.00% fo:  0.00% acc: 99.67% err:  0.33% F1: 68.75%  DEBUG: a=11 b=0 c=10 d=2997
Cat: _platinum       WordNum:   864 rc:  0.00% pc:  undef fo:  0.00% acc: 99.77% err:  0.23% F1:  undef  DEBUG: a=0 b=0 c=7 d=3011
Cat: _reserves       WordNum: 10162 rc: 50.00% pc:100.00% fo:  0.00% acc: 99.70% err:  0.30% F1: 66.67%  DEBUG: a=9 b=0 c=9 d=3000
Cat: _propane        WordNum:   756 rc: 66.67% pc:100.00% fo:  0.00% acc: 99.97% err:  0.03% F1: 80.00%  DEBUG: a=2 b=0 c=1 d=3015
Cat: _lin-oil        WordNum:   255 rc:  0.00% pc:  0.00% fo:  0.10% acc: 99.87% err:  0.13% F1:  undef  DEBUG: a=0 b=3 c=1 d=3014
Cat: _dlr            WordNum: 25447 rc: 34.09% pc: 57.69% fo:  0.37% acc: 98.67% err:  1.33% F1: 42.86%  DEBUG: a=15 b=11 c=29 d=2963
Cat: _sunseed        WordNum:  4416 rc: 20.00% pc: 33.33% fo:  0.07% acc: 99.80% err:  0.20% F1: 25.00%  DEBUG: a=1 b=2 c=4 d=3011
Cat: _potato         WordNum:   410 rc: 66.67% pc: 66.67% fo:  0.03% acc: 99.93% err:  0.07% F1: 66.67%  DEBUG: a=2 b=1 c=1 d=3014
Cat: _coffee         WordNum: 27489 rc: 82.14% pc: 92.00% fo:  0.07% acc: 99.77% err:  0.23% F1: 86.79%  DEBUG: a=23 b=2 c=5 d=2988
Cat: _cotton         WordNum:  8167 rc: 14.29% pc: 60.00% fo:  0.07% acc: 99.34% err:  0.66% F1: 23.08%  DEBUG: a=3 b=2 c=18 d=2995
Cat: _coconut-oil    WordNum:  1455 rc:  0.00% pc:  undef fo:  0.00% acc: 99.90% err:  0.10% F1:  undef  DEBUG: a=0 b=0 c=3 d=3015
Cat: _veg-oil        WordNum: 16312 rc: 35.14% pc: 86.67% fo:  0.07% acc: 99.14% err:  0.86% F1: 50.00%  DEBUG: a=13 b=2 c=24 d=2979
Cat: _groundnut-oil WordNum:   325 rc:  0.00% pc:  undef fo:  0.00% acc: 99.97% err:  0.03% F1:  undef  DEBUG: a=0 b=0 c=1 d=3017
Cat: _rubber         WordNum: 10092 rc: 66.67% pc:100.00% fo:  0.00% acc: 99.87% err:  0.13% F1: 80.00%  DEBUG: a=8 b=0 c=4 d=3006
Cat: _livestock      WordNum: 14405 rc: 25.00% pc: 54.55% fo:  0.17% acc: 99.24% err:  0.76% F1: 34.29%  DEBUG: a=6 b=5 c=18 d=2989
Cat: _crude          WordNum: 90581 rc: 70.37% pc: 85.26% fo:  0.81% acc: 97.38% err:  2.62% F1: 77.10%  DEBUG: a=133 b=23 c=56 d=2806
Cat: _ship           WordNum: 36576 rc: 59.55% pc: 88.33% fo:  0.24% acc: 98.58% err:  1.42% F1: 71.14%  DEBUG: a=53 b=7 c=36 d=2922
Cat: _soybean        WordNum: 19911 rc:  9.09% pc: 75.00% fo:  0.03% acc: 98.97% err:  1.03% F1: 16.22%  DEBUG: a=3 b=1 c=30 d=2984
Cat: _oilseed        WordNum: 26133 rc: 19.15% pc: 69.23% fo:  0.13% acc: 98.61% err:  1.39% F1: 30.00%  DEBUG: a=9 b=4 c=38 d=2967
Cat: _rye            WordNum:   461 rc:  0.00% pc:  undef fo:  0.00% acc: 99.97% err:  0.03% F1:  undef  DEBUG: a=0 b=0 c=1 d=3017
Cat: _naphtha        WordNum:   231 rc:  0.00% pc:  undef fo:  0.00% acc: 99.87% err:  0.13% F1:  undef  DEBUG: a=0 b=0 c=4 d=3014
Cat: _housing        WordNum:  2401 rc: 50.00% pc:100.00% fo:  0.00% acc: 99.93% err:  0.07% F1: 66.67%  DEBUG: a=2 b=0 c=2 d=3014
Cat: _lumber         WordNum:  2178 rc: 16.67% pc:100.00% fo:  0.00% acc: 99.83% err:  0.17% F1: 28.57%  DEBUG: a=1 b=0 c=5 d=3012
```

```
Cat: _fuel         WordNum:  2265 rc: 50.00% pc: 71.43% fo:  0.07% acc: 99.77% err:  0.23% F1: 58.82%  DEBUG: a=5 b=2 c=5 d=3006
Cat: _copper       WordNum:  7684 rc: 50.00% pc: 69.23% fo:  0.13% acc: 99.57% err:  0.43% F1: 58.06%  DEBUG: a=9 b=4 c=9 d=2996
Cat: _dmk          WordNum:   957 rc:  0.00% pc:  undef fo:  0.00% acc: 99.87% err:  0.13% F1:  undef  DEBUG: a=0 b=0 c=4 d=3014
Cat: _nat-gas      WordNum: 18990 rc: 56.67% pc: 77.27% fo:  0.17% acc: 99.40% err:  0.60% F1: 65.38%  DEBUG: a=17 b=5 c=13 d=2983
Cat: _cpi          WordNum: 13191 rc: 42.86% pc: 92.31% fo:  0.03% acc: 99.44% err:  0.56% F1: 58.54%  DEBUG: a=12 b=1 c=16 d=2989
Cat: _grain        WordNum: 83413 rc: 37.58% pc: 78.87% fo:  0.52% acc: 96.42% err:  3.58% F1: 50.91%  DEBUG: a=56 b=15 c=93 d=2854
Cat: _bop          WordNum: 19564 rc: 46.67% pc: 53.85% fo:  0.40% acc: 99.07% err:  0.93% F1: 50.00%  DEBUG: a=14 b=12 c=16 d=2976
Cat: _silver       WordNum:  3924 rc:  0.00% pc:  0.00% fo:  0.10% acc: 99.64% err:  0.36% F1:  undef  DEBUG: a=0 b=3 c=8 d=3007
Cat: _lei          WordNum:  1552 rc:100.00% pc:100.00% fo:  0.00% acc:100.00% err:  0.00% F1:100.00%  DEBUG: a=3 b=0 c=0 d=3015
Cat: _money-fx     WordNum:104727 rc: 62.01% pc: 75.51% fo:  1.27% acc: 96.55% err:  3.45% F1: 68.10%  DEBUG: a=111 b=36 c=68 d=2803
Cat: _palm-oil     WordNum:  5521 rc: 30.00% pc: 75.00% fo:  0.03% acc: 99.73% err:  0.27% F1: 42.86%  DEBUG: a=3 b=1 c=7 d=3007
Cat: _rape-oil     WordNum:   477 rc:  0.00% pc:  0.00% fo:  0.07% acc: 99.83% err:  0.17% F1:  undef  DEBUG: a=0 b=2 c=3 d=3013
Cat: _acq          WordNum:222083 rc: 94.85% pc: 93.42% fo:  2.09% acc: 97.18% err:  2.82% F1: 94.13%  DEBUG: a=682 b=48 c=37 d=2251
Cat: _orange       WordNum:  2315 rc: 72.73% pc:100.00% fo:  0.00% acc: 99.90% err:  0.10% F1: 84.21%  DEBUG: a=8 b=0 c=3 d=3007
Cat: _retail       WordNum:  6024 rc: 50.00% pc: 50.00% fo:  0.03% acc: 99.93% err:  0.07% F1: 50.00%  DEBUG: a=1 b=1 c=1 d=3015
Cat: _cpu          WordNum:   424 rc:100.00% pc: 50.00% fo:  0.03% acc: 99.97% err:  0.03% F1: 66.67%  DEBUG: a=1 b=1 c=0 d=3016
Cat: _cotton-oil   WordNum:    83 rc:  0.00% pc:  0.00% fo:  0.03% acc: 99.90% err:  0.10% F1:  undef  DEBUG: a=0 b=1 c=2 d=3015
Cat: _dfl          WordNum:  1433 rc:  0.00% pc:  undef fo:  0.00% acc: 99.97% err:  0.03% F1:  undef  DEBUG: a=0 b=0 c=1 d=3017
Cat: _yen          WordNum: 11148 rc:  0.00% pc:  0.00% fo:  0.03% acc: 99.50% err:  0.50% F1:  undef  DEBUG: a=0 b=1 c=14 d=3003
Cat: _tea          WordNum:  3509 rc: 25.00% pc: 50.00% fo:  0.03% acc: 99.87% err:  0.13% F1: 33.33%  DEBUG: a=1 b=1 c=3 d=3013
Cat: _iron-steel   WordNum:  8562 rc: 50.00% pc: 87.50% fo:  0.03% acc: 99.73% err:  0.27% F1: 63.64%  DEBUG: a=7 b=1 c=7 d=3003
Cat: _palladium    WordNum:   181 rc:100.00% pc:100.00% fo:  0.00% acc:100.00% err:  0.00% F1:100.00%  DEBUG: a=1 b=0 c=0 d=3017
Cat: _nzdlr        WordNum:  1039 rc:  0.00% pc:  0.00% fo:  0.03% acc: 99.90% err:  0.10% F1:  undef  DEBUG: a=0 b=1 c=2 d=3015
Cat: _gas          WordNum:  7314 rc: 58.82% pc:100.00% fo:  0.00% acc: 99.77% err:  0.23% F1: 74.07%  DEBUG: a=10 b=0 c=7 d=3001
Cat: _castor-oil   WordNum:    83 rc:  0.00% pc:  undef fo:  0.00% acc: 99.97% err:  0.03% F1:  undef  DEBUG: a=0 b=0 c=1 d=3017
Cat: _rand         WordNum:   819 rc:100.00% pc:100.00% fo:  0.00% acc:100.00% err:  0.00% F1:100.00%  DEBUG: a=1 b=0 c=0 d=3017
Cat: _corn         WordNum: 37662 rc: 28.57% pc: 66.67% fo:  0.27% acc: 98.41% err:  1.59% F1: 40.00%  DEBUG: a=16 b=8 c=40 d=2954
Cat: _money-supply WordNum: 21026 rc: 76.47% pc: 70.27% fo:  0.37% acc: 99.37% err:  0.63% F1: 73.24%  DEBUG: a=26 b=11 c=8 d=2973
Cat: _instal-debt  WordNum:   640 rc:100.00% pc: 50.00% fo:  0.03% acc: 99.97% err:  0.03% F1: 66.67%  DEBUG: a=1 b=1 c=0 d=3016
Cat: _nkr          WordNum:    81 rc:  0.00% pc:  undef fo:  0.00% acc: 99.93% err:  0.07% F1:  undef  DEBUG: a=0 b=0 c=2 d=3016

Macro-average:                    rc: 37.66% pc: 59.80% fo:  0.14% acc: 99.47% err:  0.53% F1: 43.23%
Micro-average:                    rc: 71.18% pc: 88.34% fo:  0.13% acc: 99.47% err:  0.53% F1: 78.84%  DEBUG: a=2667 b=352 c=1080 d=267521
```