1. Motivation and Problem Description

Embedded systems are a global business in a diverse environment. Companies and research institutions build up competences in communications, automation, robotics, or medical and security applications. All of them generate new useful information, thereby contributing to the global knowledge about embedded systems. Accessing this information in time is key for any business, research, or educational activity to stay competitive.

The Internet provides means for communication that ease publishing information (e.g., web sites). It also provides means that ease accessing such information (e.g., news sections on web sites). Global search engines provide simple ways to search for relevant information online; yet, they have drawbacks such as the information in the search databases is updated infrequently and search engines usually do not concentrate on a particular domain.

Domain portals (such as electronic newspapers) overcome these limitations. Such portals normally employ journalists, who select, edit, and then repost relevant news on the portal web-site. Due to the increase in news sources (e.g., companies, research centers) and news per source, journalists suffer from information overload and can consider only a fraction of news. This may decrease portal quality and the result is a biased view towards the market.

News engines try to address this problem. Google News, AltaVista News, Yahoo! News, Daypop, AlltheWeb News [24], and other systems collect data from domain portals and individual web sites. In contrast to domain portals, news engines automatically collect data and need no human support.
However, news engines usually do not concentrate on one specific domain and they seldom consider individual web pages as news sources. The EmBase Project provides a feasible approach that overcomes these limitations.

The EmBase Project was created to bridge the gap between the information scattered across web sites, the general need of accurate and up-to-date information, and a focus on Austria and the surrounding prospering regions.

It had to achieve three main goals:

- No human editing is required.
- Automatic classification of news into one of several categories.
- Information sources will include not only domain portals but also relevant individual web sites.

We built a news engine that automatically checks the set of web pages for changes on a daily basis. It monitors the news from more than 150 key players in the field of embedded systems, classifies each news item into the right category and presents the first 200 characters along with a link to the source web site on the portal. For classifying, the news engine uses the following categories: “business”, “technology”, “hardware”, “software”, “networking”, “embedded”, “events and conferences”, “research and publications”, and “education”. Powerful search functions integrated into the portal ease retrieving specific news or important information.

While implementing the news engine, we faced several challenges. As each web site has a different style and format, automatically retrieving individual news items is a non trivial task. Classifying news into the right category without human editing, meta-data about the web site or known categories from the publisher web site required applying current state-of-the-art classification algorithms. Finally, the classification even became more difficult, because the portal has to classify news written in English or German language.

This chapter presents the EmBase software platform developed within the EmBase Project. It describes the overall system and details about our approach to the previously mentioned challenges.

2. EmBase Software Platform

In order to comply with the requirements of high performance, efficiency, and scalability the system was designed with a layered distributed architecture. This section describes the software modules of the EmBase Web Portal, presenting first the system architecture and then each module with the specifications and the approach used for implementation.

2.1. System Architecture

The system architecture [10] defines the organization, behavior, and structural relationships between the components that together create the software system as a whole. Layered architectures support the separation of concerns, and decomposition of a large system into subtasks. Distributed architectures support performance and scalability requirements, and are inherent in client-server model, typical for Internet applications.

The EmBase system implements a client-server architecture organized in three layers. Figure 1 shows how it distributes the different functionality blocks across the three layers: presentation layer, application logic layer, and data layer.
In the EmBase system, the presentation layer is a web interface, the application logic is implemented by the retriever, analyzer, and publisher modules, and the data layer is represented by a common SQL storage solution. The modules of the application logic layer are organized in style of a data-flow architecture, as they execute a series of transformations on the set of input data. As the system has to process and store all the news items published on more than 150 web sites, this data is centralized in a repository (SQL Database) and all three modules are clients who access and update the shared data.

2.2. The Retriever Module

The retriever is the first module in the data-flow architecture. It is responsible for retrieving information from web sites, extracting the text of each news item, identifying duplicates, and then passing on the text to the next module - the analyzer.

Specifications

The retriever module handles the web monitoring and data fetching. It monitors the web pages of interest for EmBase (represented by a list of URLs) and fetches the items of news, which have been published recently. As the EmBase web portal has to publish the up-to-date news, the fetcher must poll the indicated URLs on a daily basis, and has to identify the changes since the last review of the URL list. The retriever has to differentiate between minor corrections of old items such as edited spelling mistakes and new items. In both cases, the system has to identify the section of the web page representing a news item and extract it. In the case of minor changes, the retriever has to update the record in the database representing the corrected item, without changing the date of publication. In the case of a new published item, the retriever has to create a new record, store all the data plus the current publishing date.

The retriever module has to cope with different types of publishing. The URL database contains about 150 addresses of web sites, each one having a different style and structure. At each individual web site, the retriever must correctly identify all news items. For example, there are web sites which publish all news items on the same page, because their content is not that long. Other web sites maintain an index page with a short description/title and a link to the entire content of the news item.
It is common, that companies and research institutions publish additional documents such as product information documents or scientific reports accompanying the news items. The publishers usually use different document formats for different activities (e.g., product releases in Adobe PDF, scientific reports in Postscript, or organizational tables as Microsoft Word documents). The retriever module must extract the text from different formats such as HTML/XML, PDF, plain text, Postscript, or Word documents.

The receiver retains only the text of the news items and accompanying documents and passes it to the analyzer module. The communication interface is provided by a table in the database, which contains for each news item the title, body, URL, name of source, creation date, and additional notes.

Implementation

The retriever module uses a framework built by Online-Market Watch. This framework provides basic functionality for processing and extracting text information from web sites (see Figure 2 for the architecture).

The framework includes an engine, which fetches the sites, extracts the items of news, converts them to plain text, and stores them in the database. The framework can be extended by coding additional web handlers for matching and extracting specific parts of HTML files. Each site that is monitored has a corresponding web handler, or wrapper, which defines the delimiters of the zones of interest.

Web handlers are usually composed of two functions: the handler-index and the handler-item. The handler-index function identifies the set of sub pages (web pages containing single news items) to be fetched from a given target web page; the second function specifies which sections are extracted from individual web pages.

As example, we show the web handler for Infineon Technologies (see Figure 3 for a snapshot): http://www.infineon.com/cgi/ecrm.dll/jsp/showfrontend.do?lang=EN&channel_oid=-11421

The index page has a table with the title of the news item and link to the entire document. A fragment of its HTML source code is:

```html
<tr valign="top">
    <td><br></td><td width="80" class="smalltext">2004-Apr-23</td>
    <td><br></td><td><br></td><td class="smalltext">
```
Each news item is published via the `showfrontend` program with the parameter: content_type=NEWS
The handler-index function will define a regular expression to match this specific descriptor:
showfrontend.+content_type=NEWS. A fragment of the HTML source code of the web page corresponding to a news item is:

```
<div class="largeheadline">News</div>
<img alt="" src="/ecrm/images_lib/ecrm/relaunch/spacer.gif" width="1" height="15"><br>
<div class="contheadline">Infineon Technologies Plans Expansion of Manufacturing Capacity - Will Start Equipping 300mm Module in Virginia Plant</div>
<div class="normaltext">2004-04-23</div>
<img src="/ecrm/images_lib/ecrm/relaunch/spacer.gif" width="1" height="15"><br>
```

Munich, Germany / Richmond, Virginia – April 23, 2004 – Infineon Technologies AG (FSE/NYSE: IFX) today announced a capacity expansion...
at its Virginia subsidiary semiconductor plant, Infineon Technologies Richmond, LP. The expansion will begin with an initial equipment move in to start production of advanced DRAM chips on 300mm wafers beginning in early 2005. The US dollar 1 billion expansion project

For each news item, the retriever collects the header, body, and the source name. In the example above the document contains a title, a publishing date, and some text. We considered the title being the header, and the remaining text being the body. These fragments are identified by specific comments in the HTML document (e.g., enclosed by <!-- and -->) or by different <div> or <span> classes. We selected the second approach, so we fetch the fragments that correspond to certain style class for the <div> tag (e.g., "contheadline", "normaltext").

In order to differentiate between similar news items (e.g., corrected spelling mistakes or layout changes), the retriever compares new items with old items. As a threshold value, if more than 90 percent of its words are already in another item from the database with similar frequencies, then the news item is a duplicate (i.e., republished news).

2.3. The Analyzer Module

The analyzer module is the second module in the data-flow architecture. It is responsible for classifying the news items into different categories (currently seven are available) and providing this information to the publisher module.

Specifications

The analyzer module indexes plain text documents, analyzes them, and assigns to each document one or more categories and the corresponding relevancy scores. Its input is a list of news items represented as text fields and stored in the centralized database.

The set of categories is fixed from the beginning and does not change during run-time: the number of categories, their name and meaning (any way of defining what the category refers to) are known. A formal set of rules to encode the exact definitions of categories is not available, but the project members have the knowledge to give examples of documents for each category. Currently, one document belongs to one category to prevent duplicates in the publisher module, but in the future this might change.

The analyzer splits the given text documents into words (groups of words, or abstract features) and builds an internal model for each category (e.g., identifies the keywords for each category). This is used to classify text provided by the retriever module. The set of categories is predefined and the rules for classifications are determined automatically.

The documents within the same category can be listed by chronological order or by relevance, so the fact that a document belongs to a category is indicated by a ranking score. Additionally, all news items are indexed to offer search facilities on the presentation layer. Users can view recent news, or perform full text searches for particular items.

The analyzer stores its output in the database. Each news item is updated with a category, score information, and the index represented in a document-words matrix.
Implementation

The main task is to assign categories to text documents. Text categorization is a relatively new field in computer science and there are two known approaches. Knowledge engineering builds an expert system from a set of rules - one rule per category - which manually encode the expert knowledge on how to classify documents. Machine learning text categorization uses a process of learning a model for each category, based on the characteristics discovered in a set of manually labeled training examples [23].

The requirements of the analyzer module imply an automatic process and machine learning. The project members can provide training examples for each category, but not a set of classification rules. Furthermore, the set of categories may evolve in time. Based on these arguments, the analyzer module uses machine learning (implementing a process described in [27]). The algorithm induces a classifier by generalizing the training examples, and it is able to assign labels to new documents. The advantage is that machine learning text categorization does not only build a classifier, but also a builder of classifiers. If the original set of categories is updated, or if the classifier is ported to a completely different domain, the algorithms are still valid and can be re-used. Furthermore, it is easier to manually classify a set of documents, than to define the characteristics of categories.

The analyzer approach uses inductive learning [12], because information is inferred from available data. Similar examples are grouped into classes, and the system tries to formulate rules, if it is possible to predict the class of new documents. The process uses supervised learning as the classification is supervised by the knowledge of categories and by a set of pre-classified training examples, provided by a teacher. The training set must include positive and negative examples for each category, but if all documents have labels, positive examples for one category can be used as negative examples for another. However, in case of related categories, this can decrease the accuracy.

The process of classifying consists of two phases: the learning phase when the classification knowledge is obtained, and the classification phase, when new documents are classified based on this knowledge.

The learning process is summarized in the block diagram from Figure 4.

![Figure 4. Analyzer learning process.](image-url)

First, the documents in the training set are pre-processed and represented in a vector-space model [25, 15]. The term set represents the words/terms encountered in the training example, which are
considered to be relevant for classification. The analyzer splits the text into words, and these are the terms used for indexing. More complicated representations identify the terms with phrases, but experimentally, it was determined that phrase representation does not significantly improve the task of classification [12, 27].

In the analyzer module, all classification algorithms base on the occurrences of words and their distribution in the training set. The order of words or different meanings for polysemous words are not considered. Two documents containing the same words but in a different order, are treated equally, no matter the actual meaning of them.

After building the vector-space model, the analyzer executes inductive learning algorithms on the training set and obtains draft classification knowledge. This must be evaluated on a test set and improved on a validation set. These sets must differ from the training set, because algorithms may re-classify the training documents well, but may misclassify new documents.

The testing and validation documents are pre-processed and represented in a vector space model, too. But the preprocessing differs from the one used for the training set. Only the words encountered in the training set, the words from the so called reduced term set, are considered for classification. The new words that appear in test/validation documents are ignored.

Effectiveness is the ability of a classifier to assign the right label to a new document. The analyzer tests the effectiveness of the draft classifier on the test set, by comparing the predicted categories with the known categories (the manually assigned ones). Based on their matches, the effectiveness (e.g., accuracy) is measured and if it is not good enough, some parameters of the classifier may be tuned/optimized on the validation set [35]. Common internal parameters of classifiers are: the dimension of the term set, the thresholding policy (Scut, Rcut, Pcut), the weighting scheme (atc, ltc, ntc from SMART notation), the number of categories scores taken into account for ranking classifiers, the value of \( k \) for \( k \)-NN classifier (i.e., the number of nearest neighbors considered) (see experimental results in [35]). Afterwards, the classifier is tested again, and this process continues until the desired effectiveness of the classification algorithm is obtained. In the end, the learning process is performed on the entire set of pre-classified documents (train, test, and validation documents). This further improves the effectiveness, because the classifier has been trained on more data.

The process described above represents the learning phase and is performed off-line. At the beginning, the system learns the knowledge classification, and then goes on-line once new documents are received.

Figure 5 summarizes the classification process for new documents. The analyzer pre-processes new documents the same way as test documents and executes more classification algorithms, each one giving a list of predicted categories with the confidence scores. Then, the analyzer combines these lists in order to obtain a final and better prediction [27, 18, 36].

The analyzer engine uses three classes of classification algorithms listed in [27, 12, 23, 15]:

**Example based:** \( k \)-Nearest Neighbor (\( k \)-NN). It classifies a given document \( d_j \) under category \( c_i \), if the \( k \) training documents most similar to this document \( d_j \) are also in category \( c_i \).

**Linear classification:** Rocchio. It analyzes the training set all at once (batch on-line classifier), and builds a classifier (i.e., profile) for each category. This classifier is represented in the same vector-space model as all the other documents. The Rocchio algorithm computes how close the
test document is to the centroid of the positive training examples, and how far it is from the centroid of the negative training examples.

**Probabilistic models:** Naive Bayes classifies a document by estimating the probability that a document \( d_j \) belongs to category \( c_i \) from \( P(c_i) \) (the probability that a randomly picked document has the category \( c_i \)) and \( P(w_k|c_i) \) (the conditional probability of a word given a category). Naive Bayes assumes that any words are statistically independent of each other (i.e., appear independent of their context).

Preprocessing

The analyzer cannot interpret a plain text document, but needs a structured representation of its content. In the vector-space model a document \( d_j \) is represented as a vector of term weights: \( d_j = \{ w_{1j}, \ldots, w_{|T|j} \} \), where:

- \( T = \) the set of terms/features that occur at least once in at least one document of the training set.
- \( w_{kj} = \) the weight of term \( t_k \), relatively to document \( d_j \), meaning how significant is this term for the document.

During the preprocessing phase, the indexer receives news items in plain text format and provides a vector of terms. It is composed of the following functions (Figure 6):

The tokenizer and lexical filter split raw text from the input into an array of tokens/words. This separation process has impact on the performance, because the more terms generated, the more calculations are to be done at the next classification levels. On the other hand, a classifier with a small term set has less accuracy. The EmBase analyzer keeps only alphabetic characters.

The case converter converts all terms into their lowercase version, except some specific technical terms such as "IEEE", "WINDOWS", "CPU". Upper case words are not processed by the stemming algorithm.
The stemming sub-module is responsible for stripping the suffixes from lower case terms in the term set: each word is reduced to its basic morphologic form (e.g., advanced -> advance, terms -> term, etc.). The result is a more compact term set than before, that sometimes improves the overall accuracy by a few percents. The analyzer uses more algorithms with different results, all derived from the initial Porter algorithm published in [22].

The stop words sub-module filters the common words that occur frequently in all documents. They usually have no actual influence on the context, but are needed from a human perspective to form meaningful sentences. This module uses a list of about 524 words compiled from various stop-words lists available online (e.g., http://www.dcs.gla.ac.uk/idom/ir_resources/linguistic_utils/stop_words). These are the most common words in English vocabulary (e.g., about, after, be, because, every, get, here, how, last, must, now, out, and so on).

As various methods for computing term relevancy exists, the counter sub-module computes the count and frequency for each term in the term set.

When all training set documents are indexed, the analyzer identifies the term set. The number of words compiled from the training set (the cardinality of term set) may be too high and will decrease the speed of the classification algorithms. All the algorithms currently implemented in the analyzer module can scale up to high dimensionality of vectors, but this is not the case for other learning algorithms (e.g., LLSF) [27]. The dimensionality of vectors is reduced by selecting only a fraction of words. This increases efficiency and sometimes accuracy, as it reduces over-fitting. There are many functions with different complexity to rank the words in order to select the most relevant ones. Theoretical studies [37] proved that the easiest function has very good results. The analyzer module implemented the document frequency filter by keeping only the words, which appear in a higher number of training documents (only 10 percent terms are retained). The system is designed to be easily extended by adding new classes of algorithms, thus the dimensionality reduction module is independent and may be or not enabled. Currently, the analyzer uses the entire set of terms.

Computing Weights

Computing weights is a key element for the classification process. The analyzer computes the relevance of each word for a document and for each category. If the weights were computed in a wrong way, then even if the classifier were the best possible, it would have a very low effectiveness.

A word is more relevant for a document, if it is more frequent within that document and more infrequent in other documents. Also, a word is more relevant for a category, if it is more relevant for most documents from that category and less relevant for the documents from other categories.

The analyzer implements the term frequency - inverse document frequency (TF-IDF) weighting scheme. It combined the term frequency (i.e., the number of times a term occurs in a document) and document frequency (i.e., the number of training documents in which a term occurs) in different ways [26, 30] and the result shows similar effectiveness for the classifiers. Finally, the chosen function was LTC from the SMART (i.e., a popular information retrieval system developed by Gerard Salton of Cornell University) notation.

One important aspect is that for the training set, each word is checked against the current document and all documents in the training set. For the documents in the test set, we do not care about the document frequency of words within the test set. We consider only words in the training set with their corresponding document frequency. For a test document (or validation, or new document),
we compute TF-IDF for each word based on the frequency of the word in that document, and the
document frequency computed for the training set.

Training Set
One of the challenges of the analyzer module was that the EmBase web portal monitors bilingual
news: English and German. This is usually solved by having all training documents in both languages,
and an expert assigns categories only to documents from one language. However, the EmBase training
set was formed only from English documents. This required an automatic tool that translated all
documents in German.

When a new document has to be classified, the system does not know its language, so it cannot
select one of the training sets. We used the analyzer to detect the language during the classifying
process. The analyzer tries to classify the new document using independently the knowledge from
each training set. This actually means using two classifiers. Each one will provide for the respective
document a list of categories with confidence scores. For example, an English document will get high
scores from the classifier built on the English training set, and low scores from the classifier built on
the German training set. Thus, the language is indicated by the list with the highest scores, and the
analyzer considers only the results of this classifier.

2.4. The Publisher Module
The publisher module exports information in HTML format, and gives it to the web interface and/or
sends it by email.

The module extracts from the database the list of categories and the corresponding documents, and
sorts them by relevance and date. It also performs the search requests from users and the relevance
ranking of the query results.

For each category, the web portal displays the three most recent and relevant news items. An addi-
tional page shows all items from that category. Items that have been published within the last three
days are marked as NEW.

For each news item, the publisher module displays the following information (see Figure 7 for a
snapshot of the portal):

1. Title
2. A short descriptive text, from the beginning of the news item (at most 200 characters).
3. Time related information: the exact date of publishing on the EmBase portal. Generally this is
   the same as the date of publishing on the web site of the source institution, because the EmBase
   system monitors daily all these web-sites, and tracks the changes. However, one day it may
   happen that some source servers are inaccessible; therefore the EmBase portal cannot fetch the
   news items that day. Yet, it will fetch them one of the following days, when the server is on-line.
   In this case, the news items are published on the EmBase portal with the date of fetching. We
   selected this approach, because items are sorted by date, and we want these new items to be
   displayed on the top of the list, as they have not been displayed before on the portal.
4. The name of the source site, which has published the news item.
5. Link to the original document, on the source web site.
3. Evaluation

3.1. Test Collections

To evaluate [20, 21, 35] the classification algorithms of the analyzer module, we used three collections of documents, from different domains of application: Reuters collection from economy, Usenet collection which contains the forum chats on Internet, and EETimes collection which contains technical documents.

The Reuters collection is distributed in different versions, and the one used for testing here is Reuters-21578, Distribution 1.0. The copyright resides with Reuters Ltd. and Carnegie Group and is distributed for free for research purposes. The collection contains 22 files, in standard generalized markup language (SGML) format. The tags indicate different splits on the collection, the date, categories (topics, places, people, org, exchanges, companies, and unknown), the title, body, type, and author. The stories have a mean length of 90.6 words with standard deviation 91.6 [37]. Some documents have 14 categories but the mean is of 1.24 categories per document [37]. The analyzer module uses for evaluation the mod Apte split, and additionally all documents without at least one category were removed. The results for the evaluation are:

- Training set: 7063 documents
- Test set: 2740 documents
- Number of categories: 92 (one document can belong to multiple categories)

The Usenet collection contains around 20,000 documents, organized into 20 different newsgroups, each one corresponding to a category. The analyzer sorted the news items by date, removed duplicated and some headers, and split the documents into training and test set. The results are:

- Training set: 11348 documents
- Test set: 7550 documents
- Number of categories: 20

The Reuters and Usenet collections are popular collections for text categorization used in research for the evaluation of classification algorithms. However, the categories from these collections are unrelated with the domain of application of EmBase. Thus, a new collection was compiled from technical news published on www.eetimes.com, www.embedded.com, www.semiconductorsbusiness.com, etc. The distribution of words depends on the collection, and the effectiveness of algorithms, too. Therefore, it is important to have a technical collection for evaluation. The EETimes collection has the following structure:

- Training set: 13914 documents
- Test set: 4005 documents
- Number of clustered categories: 7 (the categories are combined from different sources): business, electronic design, embedded, networking, systems and software, technology, semiconductors.

All these three collections had to be processed and inserted into the analyzer’s database. The Reuters collection has 20 SGML files where the news items are distributed. The Usenet collection has each
news item in a separate file, and these are organized on a directory structure depending of the categories. The EETimes collection has the news published on the web, by different sources, with different structure and way of identifying the category (some had the category names as images). Thus, for each type of category, the analyzer needed a different parser to extract the fields for the database.

3.2. Evaluation of Effectiveness Measurements

The effectiveness of classifiers is usually measured by precision and recall [15]. Precision measures how many categories are correct, from the categories found by the classifier. Recall measures how many categories are found from the ones that are correct:

\[
\text{Precision} = \frac{\text{number of categories found and correct}}{\text{number of categories found}}
\]

\[
\text{Recall} = \frac{\text{number of categories found and correct}}{\text{number of categories correct}}
\]

Precision and recall are estimated by counting the correlations between the classifiers decisions and the real categories. After the classifier has been applied on the entire test set, the contingency table (Table 1) is built, for a specific category \(c_i\):

<table>
<thead>
<tr>
<th>Category (c_i)</th>
<th>Real (assigned by domain expert)?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Predicted by the classifier?</td>
<td>Yes</td>
</tr>
<tr>
<td>No</td>
<td>(FN_i)</td>
</tr>
</tbody>
</table>

Table 1. The contingency table for a category.

- \(TP_i\) = true positives, is the number of test documents correctly assigned to \(c_i\).
- \(TN_i\) = true negatives, is the number of test documents correctly unassigned to \(c_i\).
- \(FP_i\) = false positives or errors of commission, is the number of test documents incorrectly assigned to \(c_i\).
- \(FN_i\) = false negatives or errors of omission, is the number of test documents missed from the classification under \(c_i\).

The analyzer computes precision \((\pi_i)\) and recall \((\rho_i)\) as:

\[
\pi_i = \frac{TP_i}{TP_i + FP_i}
\]

\[
\rho_i = \frac{TP_i}{TP_i + FN_i}
\]

The global precision and recall, for all the categories, can be estimated by two different ways of averaging the individual results for each category [35]. In micro-averaging, the elements in the contingency tables for all categories are summed together obtaining a global contingency; precision and recall are obtained from this final table. In macro-averaging, the global precision and recall are obtained by averaging the precision and recall computed for each category. The analyzer measures both
metrics. For example, the contingency table for the combined $k$-NN and Rocchio algorithms, for the EETimes collection, as computed by the analyzer, is as follows:

Document count: 4005

<table>
<thead>
<tr>
<th>Category</th>
<th>TP</th>
<th>FN</th>
<th>FP</th>
<th>TN</th>
<th>P</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>business</td>
<td>579</td>
<td>172</td>
<td>68</td>
<td>3185</td>
<td>0.895</td>
<td>0.771</td>
</tr>
<tr>
<td>electronicdesign</td>
<td>333</td>
<td>46</td>
<td>40</td>
<td>3585</td>
<td>0.893</td>
<td>0.879</td>
</tr>
<tr>
<td>embedded</td>
<td>161</td>
<td>133</td>
<td>20</td>
<td>3690</td>
<td>0.890</td>
<td>0.548</td>
</tr>
<tr>
<td>networking</td>
<td>870</td>
<td>9</td>
<td>135</td>
<td>2990</td>
<td>0.866</td>
<td>0.990</td>
</tr>
<tr>
<td>semiconductors</td>
<td>610</td>
<td>167</td>
<td>203</td>
<td>3247</td>
<td>0.750</td>
<td>0.648</td>
</tr>
<tr>
<td>technology</td>
<td>476</td>
<td>224</td>
<td>18</td>
<td>3286</td>
<td>0.964</td>
<td>0.680</td>
</tr>
<tr>
<td>TOTAL</td>
<td>3434</td>
<td>971</td>
<td>570</td>
<td>23053</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MicroAvg Precision: 0.85764 # Recall: 0.77957 # F1 Function: 0.81674
MacroAvg Precision: 0.86879 # Recall: 0.75715 # F1 Function: 0.80914

The total number of false negatives and false positives is different, because some documents have more categories. Also, the precision and recall differ for each category. For example most of documents classified by the analyzer in 'technology' are classified correctly (i.e., high precision), and most documents from 'networking' are found by the analyzer as belonging to 'networking' (i.e., high recall). The summary of effectiveness for $k$-NN, Rocchio algorithms, and their combination is presented in the following tables:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>EETimes Stemming</th>
<th>Reuters Stemming</th>
<th>Usenet Stemming</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$-NN, $k=45$</td>
<td>0.801</td>
<td>0.698</td>
<td>0.778</td>
</tr>
<tr>
<td>Rocchio</td>
<td>0.794</td>
<td>0.773</td>
<td>0.761</td>
</tr>
<tr>
<td>Combined $k$-NN and Rocchio</td>
<td>0.817</td>
<td>0.729</td>
<td>0.796</td>
</tr>
</tbody>
</table>

Table 2. Number of true positives.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>EETimes Stemming</th>
<th>Reuters Stemming</th>
<th>Usenet Stemming</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$-NN, $k=45$</td>
<td>0.842</td>
<td>0.864</td>
<td>0.778</td>
</tr>
<tr>
<td>Rocchio</td>
<td>0.834</td>
<td>0.853</td>
<td>0.761</td>
</tr>
<tr>
<td>Combined $k$-NN and Rocchio</td>
<td>0.857</td>
<td>0.893</td>
<td>0.787</td>
</tr>
</tbody>
</table>

Table 4. Accuracy (measured as True Positives/Total number of test documents).

From these tables we can conclude that:
1. The effectiveness of the combined algorithms is usually higher.
2. The results depend on the test collections: the human preferences when manually assigning labels, the distribution of training examples for each category, the spelling of the words, and so on.
3. Stemming usually improves effectiveness; however, this does not apply to technical documents. The reason may be that some technical words are common English words, and with stemming we decrease their semantic value.
4. For collection with single-label documents (a document has only one category, as in Usenet collection), the precision, recall, and accuracy have the same value.

Such measurements were done for all internal parameters of the algorithms, or for different preprocessing steps (e.g., dimensionality reduction), to support the selection of the best variant. Compared with the results from \( k \)-NN and Rocchio, the results from Naive Bayes are poor, for example the effectiveness on EETimes collection (without stemming) is 0.62.

### 3.3. Strategies to Optimize Efficiency

All the tests were performed on the following configuration: Dual Athlon MP/2.2GHz, 4GB RAM, 360GB HDD RAID, running Linux 2.4.21, Perl 5.8.0, and PostgreSQL 7.2.3.

Dimensionality reduction increases the speed, as it reduces the number of words and significantly the number of index records. Stemming was proved to decrease the speed of some algorithms; it reduces the number of words and index records, too, but because of reductions to morphological basis of words, there are more matches between the words in test documents, and words in train documents. For example, the results for the \( k \)-NN algorithm on the Reuters collection are:

<table>
<thead>
<tr>
<th></th>
<th>Without Stemming</th>
<th>With Stemming</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training documents</td>
<td>7063</td>
<td></td>
</tr>
<tr>
<td>Test documents</td>
<td>2740</td>
<td></td>
</tr>
<tr>
<td>Training words</td>
<td>22367</td>
<td>15883</td>
</tr>
<tr>
<td>Training index records</td>
<td>337354</td>
<td>314186</td>
</tr>
<tr>
<td>Classification time for entire test set in seconds</td>
<td>903</td>
<td>1005</td>
</tr>
</tbody>
</table>

**Table 5.** Classification time for \( k \)-NN with and without stemming on Reuters.

Furthermore the data structures were improved to increase efficiency, using hashes of arrays. The classification times obtained in the end are presented in Table 6. The algorithms were executed on the entire test set. We measured the total classification time for the test set and the average classification time per document. We also added a custom metric: the total number of operations (comparison of terms from test set and training set) for the test set and the average number of operations per document.

The classification time for \( k \)-NN is much higher than for the other algorithms, which are linear, but we can notice the huge number of operations that \( k \)-NN has to perform. The average classification time per document is not higher that 1 second for \( k \)-NN, which is reasonable enough for the requirements of the system.
<table>
<thead>
<tr>
<th></th>
<th>EETimes</th>
<th>Reuters</th>
<th>Usenet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test documents</td>
<td>4005</td>
<td>2740</td>
<td>7550</td>
</tr>
<tr>
<td>Rocchio</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Classification time per test set</td>
<td>13 s</td>
<td>16 s</td>
<td>21 s</td>
</tr>
<tr>
<td>Average classification time per document</td>
<td>3.25 ms/doc</td>
<td>5.84 ms/doc</td>
<td>2.78 ms/doc</td>
</tr>
<tr>
<td>Total number of operations per test set</td>
<td>4.517.640 op</td>
<td>4.252.480 op</td>
<td>7.429.200 op</td>
</tr>
<tr>
<td>Average number of operations per document</td>
<td>1.128 op/doc</td>
<td>1.552 op/doc</td>
<td>984 op/doc</td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Classification time per test set</td>
<td>4043 s</td>
<td>484 s</td>
<td>1606 s</td>
</tr>
<tr>
<td>Average classification time per document</td>
<td>1.009 s/doc</td>
<td>0.177 s/doc</td>
<td>0.213 s/doc</td>
</tr>
<tr>
<td>Total number of operations per test set</td>
<td>870.271.922 op</td>
<td>61.433.540 op</td>
<td>179.312.500 op</td>
</tr>
<tr>
<td>Average number of operations per document</td>
<td>215.254 op/doc</td>
<td>22.421 op/doc</td>
<td>23.750 op/doc</td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Classification time per test set</td>
<td>21 s</td>
<td>45 s</td>
<td>42 s</td>
</tr>
<tr>
<td>Average classification time per document</td>
<td>5.25 ms/doc</td>
<td>16.40 ms/doc</td>
<td>5.56 ms/doc</td>
</tr>
<tr>
<td>Total number of operations per test set</td>
<td>4.769.955 op</td>
<td>13.316.400 op</td>
<td>10.894.650 op</td>
</tr>
<tr>
<td>Average number of operations per document</td>
<td>1.191 op/doc</td>
<td>4.860 op/doc</td>
<td>1.443 op/doc</td>
</tr>
</tbody>
</table>

Table 6. Summary of classification time comparative for different algorithms

4. Conclusions

The EmBase portal provides a state-of-the-art information portal for embedded systems. It retrieves, classifies, and publishes news from other domain portals and individual web sites. Its target audience are companies, research centers, and educational institutions.

To implement this portal, several steps were necessary: technology selection, coding algorithms, building evaluation sets. The portal uses three algorithms in parallel (k-NN, Rocchio, and Naive Bayes) and creates a combined score from the results of each algorithm. We used known test collections for evaluating the portal such as Reuters\(^1\) and Usenet\(^2\), freely available for scientific research. The training for the final running system was performed on a collection of more than 15,000 documents, selected by the EmBase members, from the EETimes collection.

The results of the web portal show that we met our requirements in both, classification quality as well as user functionality. Depending on the training set, the accuracy of the classification is between 77-90 percent, which goes along with other experiments [27]. The portal provides two important features for its visitors: search capabilities and news presentation. Visitors can perform full-text searches in the document database. Consequently, they can use the portal for researching topics in the area of embedded systems. The portal also presents latest news in a stylish way. Visitors can quickly preview the last three news items from each category. Also, the presentation includes special marks added to the recent news to help visitors quickly identify hot new items.

The analyzer module uses machine learning and provides a feasible solution for classifying documents in the area of embedded systems. This is important as the number of electronic documents is increasing, and manual classification is not possible, because of the total number and the frequency

\(^{1}\) http://www.daviddlewis.com/resources/testcollections/reuters21578/index.html

\(^{2}\) http://kdd.ics.uci.edu/databases/20newsgroups/20newsgroups.html
in which new documents are available. The module can be re-used to, for instance, classify lists of publications or the books in a library.

As future work, we will continually improve the document training set by adding manually classified documents. In the long run, this will create an extensive training set for the area of embedded systems. Furthermore, we will add more companies to the watch list to enlarge the knowledge base.
EmBase News as Web Portal

Figure 7. Screen-shot of the EmBase News portal.