Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student:	István Módos
Study programme:	Open Informatics
Specialisation:	Computer and Information Science
Title of Bachelor Project:	Intelligent Data Warehouse with Data from the Maritime Domain

Guidelines:

1. Study the basics of document categorization an annotation.
2. Create an intelligent data warehouse collecting data and documents relevant to the problem of contemporary maritime piracy.
3. Conduct basic categorization and analysis of collected data and enrich the data warehouse with the results of the analysis.
4. Evaluate the correctness of the categorization module on an independent domain.
5. Create a user interface to access to the data warehouse.

Bibliography/Sources:

Jakob, M. et al.: Adversarial Modeling and Reasoning in the Maritime Domain (Year 2 Report). 2010

Others will be provided by the supervisor.

Bachelor Project Supervisor: Ing. Ondřej Vaněk

Valid until: the end of the winter semester of academic year 2012/2013

[illegible]

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
L.S.
4

[illegible]

prof. Ing. Vladimír Mařík, DrSc.
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 9, 2012

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student:	István M ó d o s
Studijní program:	Otevřená informatika (bakalářský)
Obor:	Informatika a počítačové vědy
Název tématu:	Inteligentní datový sklad dat z námořní domény

Pokyny pro vypracování:

1. Prostudujte základní techniky pro kategorizaci a anotaci dokumentu.
2. Vytvořte inteligentní datový sklad shromažďující data a dokumenty relevantní k problému současného námořního pirátství.
3. Proveďte základní kategorizaci a analýzu shromážděných dat a obohaťte datový sklad výsledky této analýzy.
4. Vyhodnoťte správnost kategorizačního modulu na nezávislé doméně.
5. Vytvořte uživatelské rozhraní pro přístup k datovému skladu.

Seznam odborné literatury:

M. Jakob et al.: Adversarial Modeling and Reasoning in the Maritime Domain (Year 2 Report). 2010

Další literaturu dodá vedoucí práce.

Vedoucí bakalářské práce: Ing. Ondřej Vaněk

Platnost zadání: do konce zimního semestru 2012/2013

L.S.

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická

[illegible]

prof. Ing. Vladimír Mařík, DrSc.
vedoucí katedry

[illegible]

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 9. 1. 2012

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics

Bachelor's Project

Intelligent Data Warehouse with Data from the Maritime Domain

István Módos

Supervisor: Ing. Ondřej Vaněk

Study Programme: Open Informatics

Specialisation: Computer and Information Science

May 24, 2012

Declaration

Aknowledgements

I would like to thank my parents for their constant support through my studies and my supervisor for his guidance and advices.

Abstract

This work deals with design and implementation of intelligent data warehouse for domain of maritime piracy. This warehouse provides automatic categorization of text documents based on their topics, allows to search for documents using queries, it automatically downloads news from predefined sources and provides analysis of submitted piracy incidents in form of charts and map of incidents. For creating the categorization module, we studied the task of text classification. We evaluated three classification methods (Naive Bayes, k-nearest neighbours and Support Vector Machines) on a dataset from multi-agent systems domain.

Abstrakt

Tato práce se zabývá implementací inteligentního datového skladu dostupného z webového prohlížeče pro data z domény námořního pirátství. Sklad provádí automatickou kategorizaci textových dokumentů na základě jejich tématu, umožňuje vyhledávání dokumentů dle uživatelských dotazů, stahuje zprávy z předdefinovaných zdrojů a nabízí také analýzu vložených pirátských útoků ve formě grafů a mapy útoků. Pro vytvoření modulu pro kategorizaci dokumentů bylo nutné nastudovat problematiku klasifikace textu a provést rešerši několika používaných metod (Naivní Bayesův klasifikátor, algoritmus k nejbližších sousedů a Support Vector Machines). Tyto metody byly otestovány na množině dokumentů z domény multi-agentních systémů.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In early days, collection of information meant searching through hundreds of documents and finding relevant information by hand. But now, when performance of computers is sufficient, more effort is spent on automation of these tasks by computer programs and leave only the most important decision to human. This is possible due to the progress made in fields of machine learning and data mining.

Organizations involved in preventing maritime piracy gather documents from different sources and they need a mechanism to categorize and extract relevant information from them. Some resources are in form of reports (PDF, etc.), newsfeeds or structured files containing geographic data about piracy incidents. Our goal is to implement a platform-independent data warehouse, which can handle these information, i.e. store them, categorize them to predefined categories and provide them to user either by search queries or as visualization on map and charts.

Existing services are suitable only as web folders for synchronization and sharing, others can offer some additional functionality, such as document editing and file viewing, but none of these services provide automatic categorization and domain-specific features (at least not for free or small amount of money). There are also some database management systems, which incorporate data mining algorithms, but these systems are expensive, thus making them unsuitable for non-enterprise users.

First, we analyse the specifications for the warehouse, upon which we design a warehouse architecture. Then we introduce the task of text classification needed for automatic data categorization. Introduced classification methods will be evaluated on a dataset from domain of multi-agent systems. The next section will provide a description of the implementation with used libraries. Last section summaries this work and offers few ideas for future work.

Chapter 2

Related work

There are many services for document management, file sharing and data analysis, here we provide a short list of selected public solutions:

1. Dropbox[23]
    - mainly used for file sharing and synchronization
    - accessible from web browser or from standalone client (which is cross-platform)
    - file revisions
    - manual file categorization using folders
    - no file searching, automatic categorization and data analysis
2. Google Docs[25]
    - document sharing and collaborative editing
    - accessible form web browser
    - conversion of files to text
    - file searching, reviewing and revisions
    - manual file categorization using folders
    - no automatic file categorization and data analysis
3. OpenKM[4]
    - open-source document management system
    - system can be deployed at own server
    - accessible from web browser or from standalone client (which is cross-platform)
    - conversion of files to text
    - file searching and revisions
    - manual file categorization using folders and user defined categories
    - no automatic file categorization and data analysis

4. Pentaho[5]
    - business intelligence suite for data analysis
    - enterprise and open-source editions (however the open-source edition has a very small set of features)
    - accessible from web browser
    - provides complex data analysis - charts, maps, dashboards, tables etc.
    - data for analysis may be stored in database - user provides tables and columns to be analysed
    - analysis results can be exported in CSV, PDF, XLS etc.
    - support for scripting
    - no file searching and automatic categorization

There are also solutions for large enterprises, for example Autonomy Corporation[9] provides “Intelligent Data Operating Layer” technology. They offer extraction of information from structured and unstructured data, document searching, categorization, summarization, scene detection etc.

Chapter 3

Design

3.1 Specifications and use cases

Here we provide a list of use cases and specifications for the warehouse:

- user access to the warehouse should be platform-independent and without the need of installing additional application
- submission of piracy incidents in web form or from KML files
- uploading files to the warehouse
- automatic categorization of files into predefined categories with optional user intervention
- downloading files from the warehouse
- searching for files using queries
- analysis of piracy incidents
- automatic acquisition of documents from maritime piracy websites

3.2 Task analysis

One of our main requirement is categorization of incoming data into categories, which may overlap (i.e. one file can be assigned to more categories). We will denote the categorization process as *tagging* and categories will be called *tags* (or *labels*, used interchangeably). Tagging should be two-step process: automatic and manual. The manual step is mainly for auditing, i.e. correction and specifying additional tags by the user. The question is, why to use automatic tagging when we already rely on user interaction? The answer is that the number of tags could be large and it would be very tedious work for users to go through all of them and choose the relevant ones. With automatic tagging, we will suggest only few tags and user will only add the most obvious ones which were omitted in the first step (we are making an assumption that most documents have small number of tags). We will limit the automatic step to files from which we can extract some text (e.g. PDF, HTML etc.).

When files are categorized, a user can easily search for files by the name of the category. Another option is searching for files using specific queries. The warehouse would return a sorted list of files with descending relevancy to the given query. This is particularly useful for text files where user would search for files containing some specific keywords. Searching is done with *text indexer*, which stores the incoming files in optimized data structure for fast retrieval of relevant files. Combination of text and tags search can narrow down the number of retrieved files, e.g. query "agents" would retrieve documents about national security and robotics, but by specifying the category, the user can filter documents from unwanted category.

In maritime piracy we could devise a less dangerous routes using knowledge about location of incidents and their evolution through time. These information can be visualized in charts or on an interactive world map. The problem is acquiring this information. One option is to employ techniques from *text mining* on incoming files, however this is outside the scope of this work. We will limit ourselves to parsing incidents from files with a fixed syntax.

The warehouse will also periodically download and categorize news from web pages. This will be performed in background by *data tasks* without any intervention from users or warehouse administrator.

Security is not our primary concern, thus the warehouse will not provide any means to protect private informations, all imported documents can be viewed by everyone. This solution is suitable for internal use in private companies[1].

3.3 Architecture

After defining and analysing specifications, we design the warehouse architecture. Warehouse architecture with different modules can be seen in Figure 3.1; it is based on the client-server model[32]. Reasons for using this model are:

- centralized data storage - server can work with all available data and perform analysis on them.
- user application (i.e. client) doesn't do "the hard work", every computation and data fetching is done on the server side. Client can be low-end computer with internet connection and web browser.
- support for concurrent work of multiple clients.
- the warehouse can be accessed from anywhere, specification for platform-independence is satisfied.

Here we provide a brief description of every module in the architecture:

Client - provides interaction with the warehouse to user in form of a web page. User can upload and download files, search for files using queries and analyse incidents on map or charts.

Server - serving client requests and communication with database. Data tasks collect documents from web sources.

[1]International Maritime Organization, The Maritime Security Centre – Horn of Africa etc.

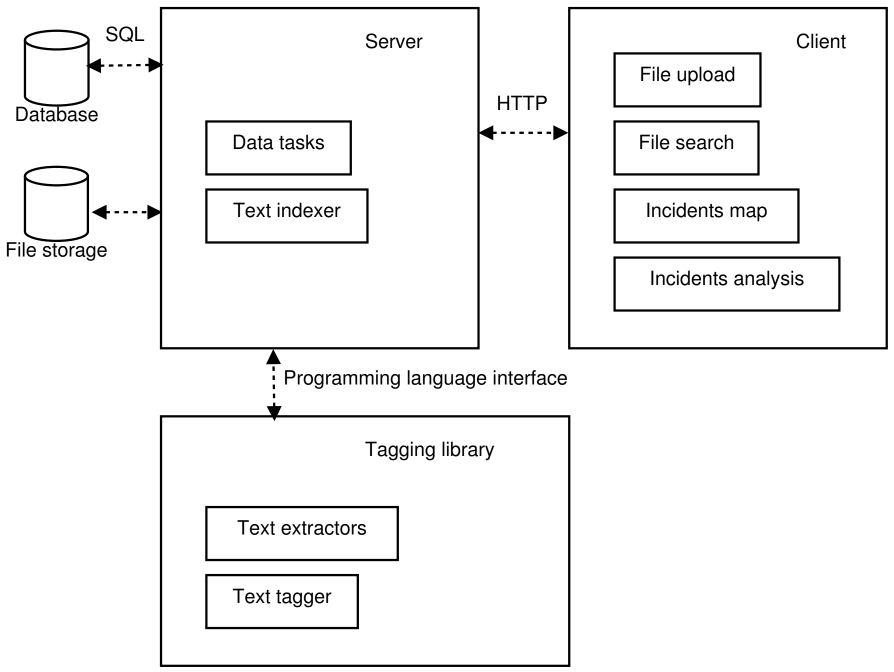


Figure 3.1: Architecture

Database - contains information about incidents, existing tags and temporary session information. For these tasks, any *relational database* with SQL queries will suffice.

File storage - in our previous work we stored files in the database with their *metadata* (name, MIME type, assigned tags etc.). With usage of indexer, we can store files directly on server's filesystem and metadata inside the indexer[2]. This enables user to perform search queries on metadata too.

Tagging library - text extractors are used to extract text from files of various formats, whereas text tagger assigns appropriate tags to extracted text. Section 4 describes text tagging in deeper detail.

3.4 Process design

This section provides a high-level overview of each function for client web page. We omitted error reporting in our flowcharts.

3.4.1 Uploading file

Files can be uploaded from user's computer or by providing publicly accessible URL. As a response, server sends back suggested tags. User can select additional tags or remove incorrectly suggested tags.

[2]this could lead to inconsistencies if files are deleted from filesystem by a careless administrator.

3.4.2 Downloading files

Since files can be uploaded to the warehouse, there should be a mechanism to download them as well. User can create complex search queries and server will return a list of relevant files to that query.

3.4.3 Visualization of incidents

The idea of the incident map is to provide a visual where incidents occur. User can filter incidents by type and year and by clicking on the incident marker, detailed information about incident is shown.

3.4.4 Incident charts

Further analysis can be made by looking at charts. Charts provide an evolution of incidents in time (i.e. per month, year etc.). As well as in map, user can filter incidents by type and year. For computer data processing, CSV files can be downloaded for each chart type.

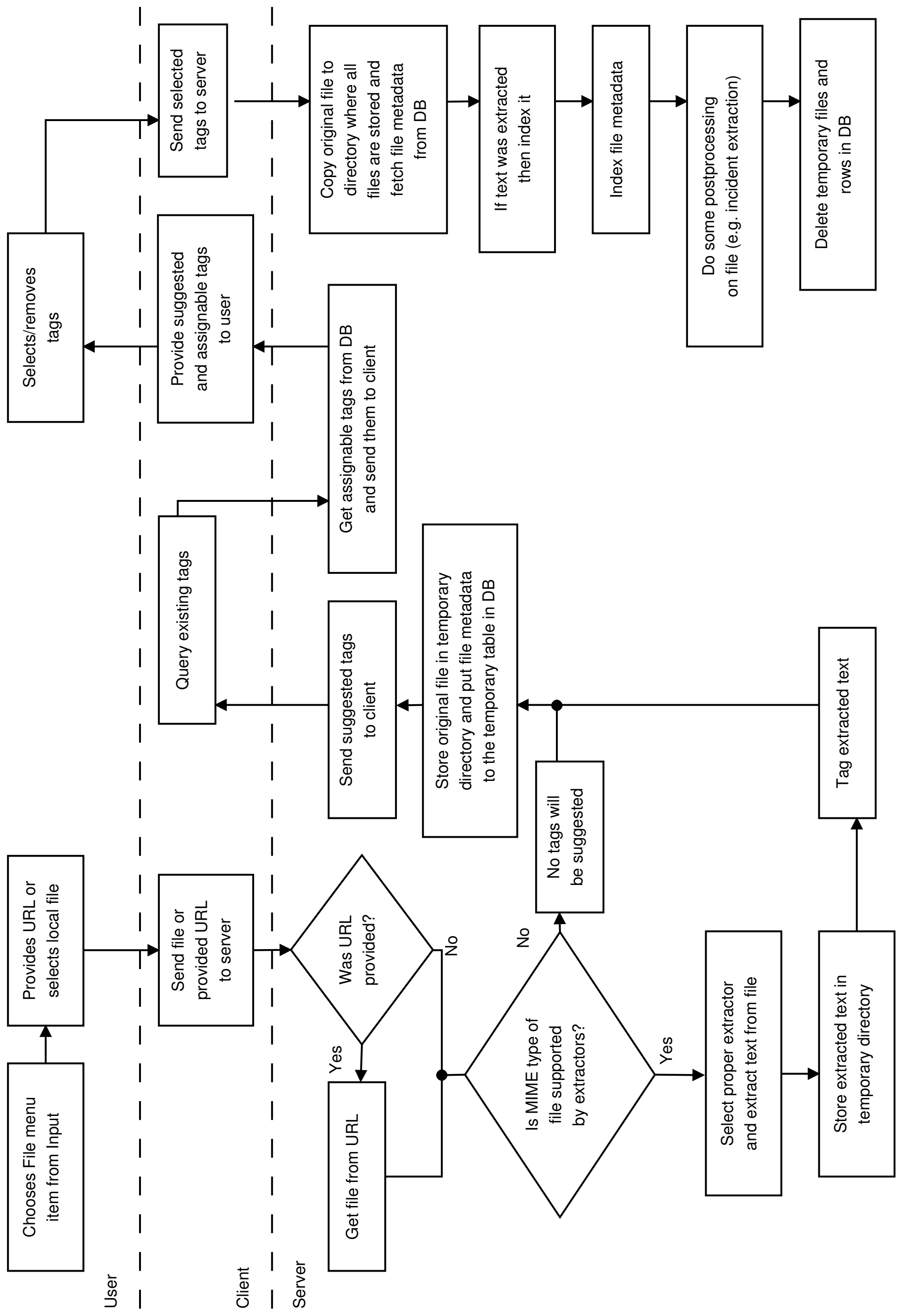

User
Client
Server
Chooses File menu item from Input
Provides URL or selects local file
Send file or provided URL to server
Was URL provided?
Yes
No
Get file from URL
Is MIME type of file supported by extractors?
Yes
No
Select proper extractor and extract text from file
Store extracted text in temporary directory
Tag extracted text
No tags will be suggested
Store original file in temporary directory and put file metadata to the temporary table in DB
Send suggested tags to client
Query existing tags
Get assignable tags from DB and send them to client
Provide suggested and assignable tags to user
Selects/removes tags
Send selected tags to server
Copy original file to directory where all files are stored and fetch file metadata from DB
If text was extracted then index it
Index file metadata
Do some postprocessing on file (e.g. incident extraction)
Delete temporary files and rows in DB

Figure 3.2: Uploading files

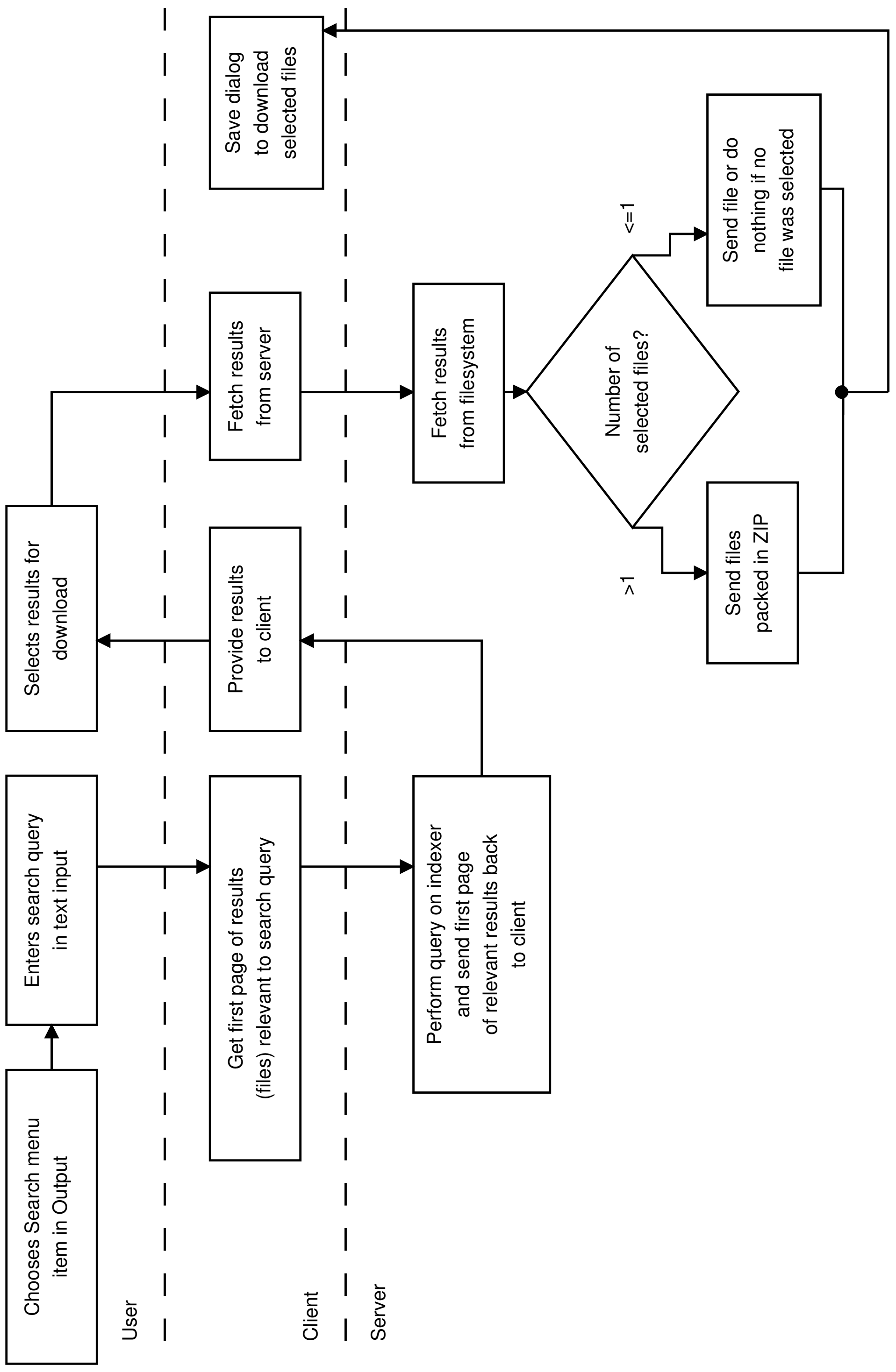

User
Client
Server
Chooses Search menu item in Output
Enters search query in text input
Get first page of results (files) relevant to search query
Perform query on indexer and send first page of relevant results back to client
Provide results to client
Selects results for download
Fetch results from server
Fetch results from filesystem
Number of selected files?
<=1
>1
Send file or do nothing if no file was selected
Send files packed in ZIP
Save dialog to download selected files

Figure 3.3: Searching and downloading files

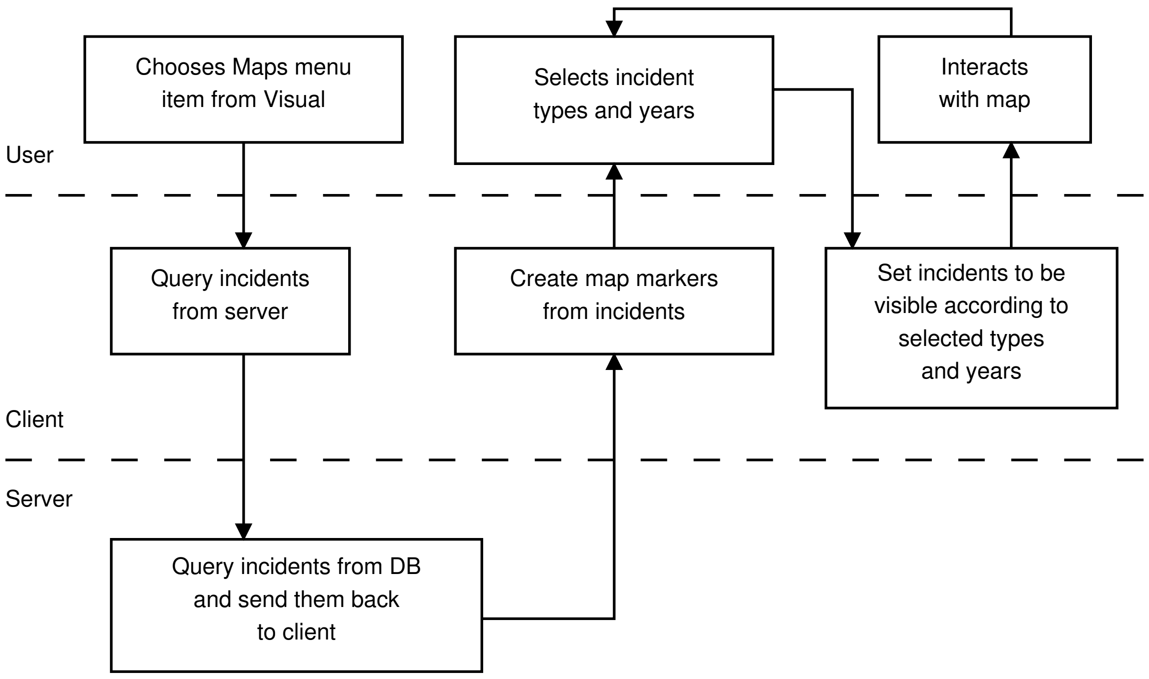

Chooses Maps menu item from Visual
Selects incident types and years
Interacts with map
User
Query incidents from server
Create map markers from incidents
Set incidents to be visible according to selected types and years
Client
Server
Query incidents from DB and send them back to client

Figure 3.4: Visualization of incidents on map

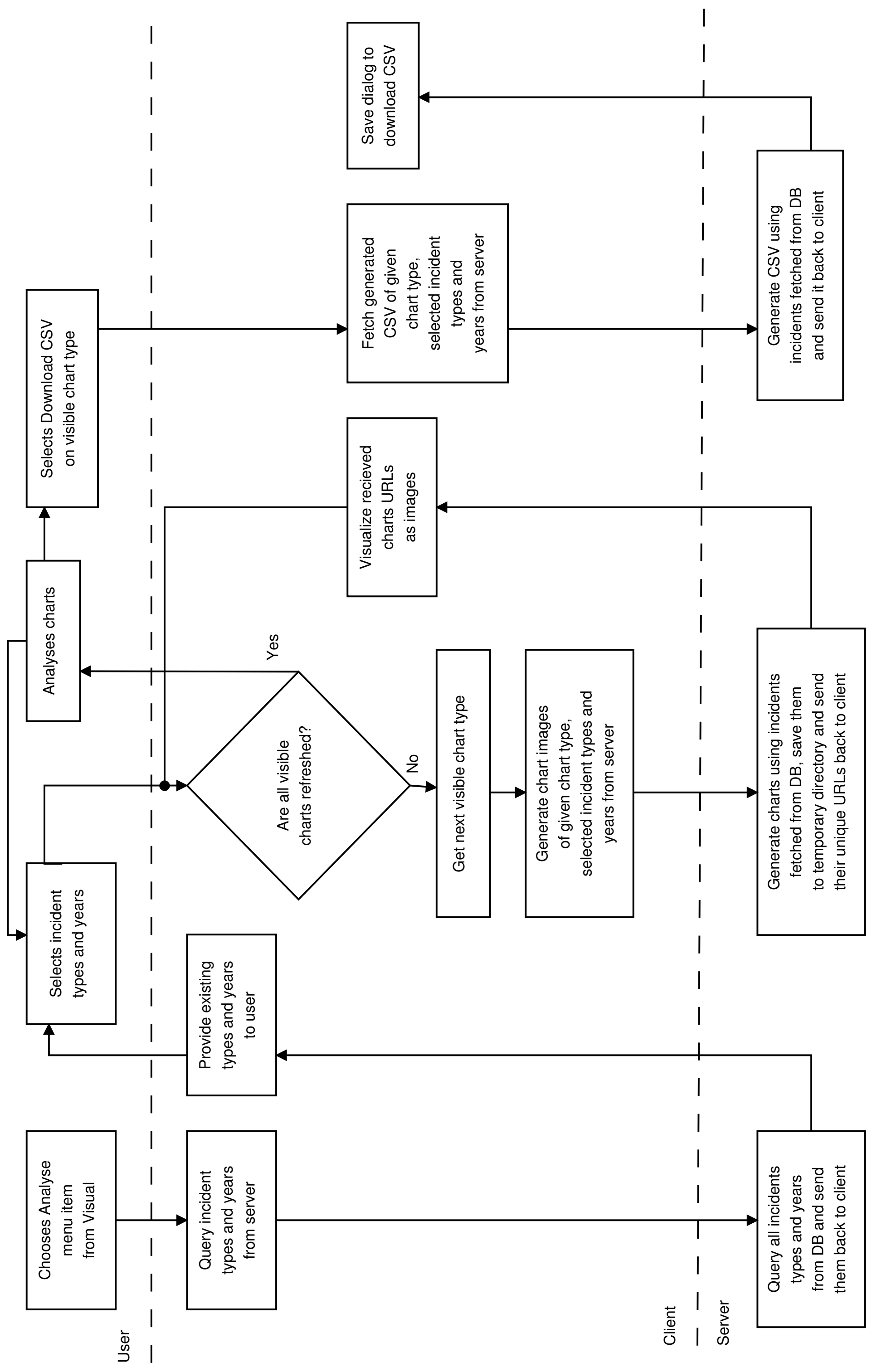

User
Client
Server
Chooses Analyse menu item from Visual
Query incident types and years from server
Query all incidents types and years from DB and send them back to client
Provide existing types and years to user
Selects incident types and years
Are all visible charts refreshed?
Yes
No
Get next visible chart type
Generate chart images of given chart type, selected incident types and years from server
Generate charts using incidents fetched from DB, save them to temporary directory and send their unique URLs back to client
Visualize recieved charts URLs as images
Analyses charts
Selects Download CSV on visible chart type
Fetch generated CSV of given chart type, selected incident types and years from server
Generate CSV using incidents fetched from DB and send it back to client
Save dialog to download CSV

Figure 3.5: Incident charts

Chapter 4

Text classification

In this section we provide an overview of the text classification task and give brief descriptions for methods used in evaluation. Gained knowledge will be used in implementation of text tagger for the warehouse.

4.1 Task definition

Definition follows [38], which also contain a nice overview of text classification. Let $D = \{d_1, \ldots, d_{|D|}\}$ be a domain of documents and $C = \{c_1, \ldots, c_{|C|}\}$ a set of categories. Our task is to find a *classifier* $\Phi : D \times C \to \{true, false\}$, whose output is a decision for input arguments $(d_i, c_k) \in D \times C$ whether to assign a category c_k to document d_i ($true$ for yes, $false$ for no). Classifier function is an approximation of the unknown *target function* $\breve{\Phi} : D \times C \to \{true, false\}$, whose output decisions are always correct.

In machine learning approach, we use a general methods from the field of ML to find the function Φ. We are given an *initial corpus* of documents $\Omega \subset D$, for which outputs of $\breve{\Phi}(d_i), d_i \in \Omega$ are known. The set Ω is split in two sets, TV and Te, where $TV \cap Te = \emptyset$. TV is called a *training* set, which is used to build a classifier. Te is called a *testing* set, on which the built classifier is tested and evaluated by some measures (discussion of used measures in evaluation will be presented in Section 5.1.2).

We define functions $\Lambda : D \to C_s$ as $\Lambda(d_i) = \{c_k \mid \Phi(d_i, c_k) = true, c_k \in C\}$ and $\breve{\Lambda} : D \to C_s$ as $\breve{\Lambda}(d_i) = \{c_k \mid \breve{\Phi}(d_i, c_k) = true, c_k \in C\}$, where C_s is a set of all possible subsets of C. In *single-label classification*, both $|\Lambda(d_i)| = 1$ and $\left|\breve{\Lambda}(d_i)\right| = 1$ must hold for every $d_i \in D$. On the other hand, in *multi-label classification* the cardinality of $\Lambda(d_i)$ and $\breve{\Lambda}(d_i)$ is not constrained.

Text classification is a task with wide range of applications, including patent classification, news classification, spam filtering, sentiment detection etc.

4.2 Document representation

First, we need to define a set of *terms* $T = \{t_1, \ldots, t_{|T|}\}$. Terms can be words, bi-grams (i.e. a pair of consecutive words) or anything else. For our purpose, we will use words (T then can be interpreted as a vocabulary).

In *Vector space model*[1], document $d_i \in D$ is represented as a vector $\mathbf{dr}(d_i) \in \mathbb{R}^{|T|}$ of *term weights* $w_j(d_i)$, i.e.

$$\mathbf{dr}(d_i) = (w_1(d_i), w_2(d_i), \ldots, w_{|T|}(d_i))$$

4.2.1 Term weights

Representations differs in meaning of term weights. In general, term weight is computed as $w_j(d_i) = \mathrm{tlw}(d_i, t_j)\,\mathrm{tgw}(t_j)$, where tlw, tgw are called *term local weight* and *term global weight* respectively. We present here a few variants which are commonly used in text classification and information retrieval [12].

- Term local weights

  term frequency $\mathrm{tlwtf}(d_i, t_j) =$ number of occurrences of term t_j in document d_i

  binary $\mathrm{tlwbin}(d_i, t_j) = \begin{cases} 1 & \mathrm{tlwtf}(d_i, t_j) > 0 \\ 0 & \text{otherwise} \end{cases}$

  logarithmic $\mathrm{tlwlog}(d_i, t_j) = \log(1 + \mathrm{tlwtf}(d_i, t_j))$

- Term global weights

  none $\mathrm{tgwno}(t_j) = 1$

  inverse document frequency $\mathrm{tgwidf}(t_j) = \log \frac{|TV|}{\#TV(t_j)}$
  where $\#TV(t_j)$ is a number of documents in training set in which the term t_j occurred.

  mutual information can be used as a weighting scheme [15] or as a *term selection* method [41][2]. Global weight can be computed as

  $$\mathrm{tgwmi}(t_j) = H(X) - H(X|Y_j) \tag{4.1}$$

  where $H(X)$ is the entropy of the random variable X whose outputs are categories of all training documents. $H(X|Y_j)$ is a conditional entropy where Y_j is a random variable which tells whether the training document contains term t_j.

To abstract from document length, we can use *normalization* of weights, i.e.

$$w_j(d_i) = \frac{\mathrm{tlw}(d_i, t_j)\,\mathrm{tgw}(t_j)}{\sqrt{\sum_{k=1}^{|T|}(\mathrm{tlw}(d_i, t_k)\,\mathrm{tgw}(t_k))^2}}$$

4.2.2 Term selection

Because T tends to be large, techniques to reduce its dimensionality are employed, which may lead to faster classification and more accurate prediction. Commonly used methods are word lower-casing, removing stopwords and stemming (T will contain only stems of words).

Another technique is *term scoring*. Every term is scored by means of some function $\mathrm{score} : T \to \mathbb{R}$ and only the n terms with highest scores are preserved. In evaluation we will use $\mathrm{tgwmi}(t)$ as a scoring function.

[1]Introduced in SMART Information Retrieval System

[2]Please note that in this article the mutual information is presented as information gain.

4.3 Methods

For evaluation we chose three methods: Naive Bayes ([30], [34], [12]), k-nearest neighbours ([12]) and Support Vector Machines ([28], [10]). Presented methods are used as single-label classifiers, but in Section 4.4 we will extend them for needs of multi-label classification.

4.3.1 Naive Bayes

Naive Bayes classification is probabilistic method based on idea of the well-known Bayes theorem

$$\Pr(c_k \mid d_i) = \frac{\Pr(c_k)\Pr(d_i \mid c_k)}{\Pr(d_i)} \tag{4.2}$$

where

$$\Pr(d_i) = \sum_{s=1}^{|C|} \Pr(d_i \mid c_s)\Pr(c_s) \tag{4.3}$$

Now we have to find estimates of $\Pr(c_k)$ and $\Pr(d_i \mid c_k)$ (we will denote them by hat).
Probability of the given category can be easily estimated from training set as

$$\widehat{\Pr}(c_k) = \frac{\left|\{d \mid \breve{\Phi}(d, c_k) = true,\ d \in TV\}\right|}{|TV|} \tag{4.4}$$

If we make an assumption, that occurrences of terms are independent, then we can estimate $\Pr(d_i \mid c_k)$ using multinomial distribution with parameters θ_{kj}

$$\widehat{\Pr}(d_i \mid c_k) = \left(\sum_{t\in T} \mathrm{tlwtf}(d_i, t)\right)! \prod_{j=1}^{|T|} \frac{\theta_{kj}^{\mathrm{tlwtf}(d_i,t_j)}}{\mathrm{tlwtf}(d_i, t_j)!} \tag{4.5}$$

Now we have to find $|C||T|$ parameters θ_{kj}. Using Maximum likelihood estimate we can arrive to estimates

$$\theta_{kj} = \frac{\sum_{d\in TV_k} \mathrm{tlwtf}(d, t_j)}{\sum_{t\in T}\sum_{d\in TV_k} \mathrm{tlwtf}(d, t)} \tag{4.6}$$

where $TV_k = \{d \mid d \in TV, \breve{\Phi}(d, c_k) = true\}$. To avoid multiplication by zero in Eq. 4.5 for zero-frequency terms in classes we use *Laplace smoothing*[12].

With these derived results, we can define our classifier as

$$\Phi(d_i, c_k) = \begin{cases} true & c_k \in \arg\max_{c_r\in C} \widehat{\Pr}(c_r \mid d_i) \\ false & \text{otherwise} \end{cases} \tag{4.7}$$

The reason why to maximize $\widehat{\Pr}(c_r \mid d_i)$ comes from minimization of the *Bayesian risk* [16].

Disadvantage is the independence assumption, which is often violated by real data.

Advantages of Naive Bayes method is fast classifier modelling, no parameter selection and simplicity of the implementation. Naive Bayes is successfully employed in *spam-filtering*[3].

[3]`http://spamassassin.apache.org/` project uses Naive Bayes in combination with different techniques.

4.3.2 k-nearest neighbours

Let us define a *distance function* $\mathrm{dist} : \mathbb{R}^{|T|} \times \mathbb{R}^{|T|} \to \mathbb{R}$. For given testing document d_i we construct a set $N_k \subset TV$ of k elements, whose distances to d_i (i.e. value of $\mathrm{dist}(\mathbf{dr}(d_i), \mathbf{dr}(d_j)), d_j \in TV$) are lowest. Classifier will be then defined as

$$\Phi(d_i, c_s) = \begin{cases} true & c_s \in \arg\max_{c_r \in C} \left| \{d \mid \breve{\Phi}(d, c_r) = true, d \in N_k\} \right| \\ false & \text{otherwise} \end{cases} \tag{4.8}$$

k-NN can suffer from overfitting, thus some mean to find k which minimizes error on testing data has to be employed. Other disadvantage can be somewhat slower classification phase, since obtaining the set N_k can be costly (to improve the classification speed *space-partitioning structures* like *k-d tree* can be used instead of linear search).

Advantages of k-NN are simple implementation and capability to classify non-linearly separable data. Performance of k-NN is usually among the top classifiers (see [28],[40]).

4.3.3 Support Vector Machines

SVM is a binary classification method first proposed for text classification task in [28]. First, we briefly describe how SVM works in its general, *soft-margin* form, then we apply it to our task.

Let $(\mathbf{x}_i, y_i), \mathbf{x}_i \in \mathbb{R}^n, i = 1, \dots, m$ be a training set of data with their corresponding labels $y_i \in \{-1, 1\}$. SVM finds the separating hyperplane of this dataset with the widest margin (distance from the hyperplane to nearest data from both classes) by solving the following optimization task with unknowns $\mathbf{w}, b, \xi_i$

$$\min \frac{1}{2} ||\mathbf{w}||^2 + C \sum_{i=1}^{m} \xi_i \tag{4.9}$$

$$s.t. \quad \forall i : \; y_i(\mathbf{w}\mathbf{x}_i + b) \geq 1 - \xi_i \tag{4.10}$$

$$\forall i : \; \xi_i \geq 0 \tag{4.11}$$

where $(\mathbf{w}, b)$ are parameters of the hyperplane, C is a *cost* parameter and ξ_i are *slack* variables.

The formulation in Eq. 4.9 is transformed into dual task with variables $\alpha_i, i = 1, \dots, m$ (α_i are *Lagrange multipliers*)

$$\max \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \mathbf{x}_j \tag{4.12}$$

$$s.t. \quad \sum_{i=1}^{m} \alpha_i y_i = 0 \tag{4.13}$$

$$\forall i : \; 0 \leq \alpha_i \leq C \tag{4.14}$$

Vectors $\mathbf{x}_i$ for which their corresponding Lagrange multipliers are greater than zero are called *support vectors.*

Variables from primal task Eq. 4.9 are found as

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i \tag{4.15}$$

$$b = y_k(1 - \xi_k) - \mathbf{w}^T \mathbf{x}_k \tag{4.16}$$

where k is an index of any support vector, i.e. $\alpha_k > 0$ must hold. Classification of the new data $\mathbf{x}$ will then be

$$\operatorname{sign}(\mathbf{w}^T \mathbf{x} + b)$$

For data, which are not linearly separable, a kernel trick can be used by replacing dot products with kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$, which computes a dot product of $\mathbf{x}_i, \mathbf{x}_j$ in higher dimensional space.

For assigning a category for a testing document d_i, additional work needs to be done, since SVM is a binary classification method. We solve this by *one-against-one* method, where we build a $\frac{|C|(|C|-1)}{2}$ binary classifiers on each pair of categories. For categories c_r, c_s with condition $r < s$ we define a set $TV_{rs} = \{d \mid \breve{\Phi}(d, c_r) = true \vee \breve{\Phi}(d, c_s) = true, d \in TV\}$. We associate a category c_r with class -1 and category c_s with class 1. Binary SVM classifier is then built on TV_{rs} and its decision is defined as

$$\Phi_{rs}(d_i) = \begin{cases} c_r & \operatorname{sign}(\mathbf{w}_{rs}^T \mathbf{dr}(d_i) + b_{rs}) < 0 \\ c_s & \text{otherwise} \end{cases} \tag{4.17}$$

where $(\mathbf{w}_{rs}, b_{rs})$ are parameters of the hyperplane in classifier Φ_{rs}. Final decision for document d_i and category c_k is then

$$\Phi(d_i, c_k) = \begin{cases} true & c_k \in \arg\max_{c \in C} |\{(r,s) \mid \Phi_{rs}(d_i) = c, 1 \le r < s \le |C|\}| \\ false & \text{otherwise} \end{cases} \tag{4.18}$$

SVM has probably the slowest training phase from all presented methods (however, classification phase for linear kernel is very fast as it requires only computing dot product on sparse vectors). Soft-margin SVM has also a cost parameter to tune.

4.4 Multi-label classification

Different approaches to solve the multi-label classification task are presented in [39]. For our purpose the *binary relevance* method is used. For each classification method we construct $|C|$ binary classifiers $\Phi_{c_k} : D \times \{0, 1\} \to \{true, false\}$. Each classifier Φ_{c_k} is built on transformed dataset D_{c_k}, where documents assigned with category c_k are associated with class 1 (note that c_k doesn't have to be the only assigned category for those documents) and documents without this category are associated with class 0. Classification of the new document is then defined as

$$\Phi(d_i, c_k) = \Phi_{c_k}(d_i, 1) \tag{4.19}$$

In this section we provided an overview of text classification task. We introduced different term weighting techniques and presented three methods used in classification. Next section will evaluate these methods with different term weight combinations on multi-label dataset from domain of multi-agent systems.

Chapter 5

Evaluation

In this section we evaluate classification methods presented in Section 4 on multi-label dataset. The second part of this section deals with time measurements of the warehouse.

5.1 Classification methods

5.1.1 Dataset

As dataset for evaluation, we used a collection of scientific papers published on four conferences from multi-agent systems domain: International Conference on Autonomous Agents and Multi-agent Systems, AAMAS 2008 - 2011. We will refer to this collection as *aamas*. Every document in this collection contains a field with document topics, which are used as labels. We didn't use the whole text of documents, we removed the part before the Introduction section. Reason for this is to remove fields with document topics, because without removing them, classifiers tend to overfit (they base their decision only on occurrence of those topics). We also removed categories with low document frequency. Number of documents for each category in dataset is presented in Table 5.1. Average number of labels per document is 1.9546.

Category	Number of documents
economics	149
algorithms	411
experimentation	251
design	163
human factors	68
performance	103
theory	276

Table 5.1: Number of documents per category in *aamas*

5.1.2 Measures

Let c_k be the examined category, then we define (see [29],[38],[39])

$$TP_k = \left|\{d_j \mid \Phi(d_j, c_k) = true \wedge \breve{\Phi}(d_j, c_k) = true, d_j \in Te\}\right| \tag{5.1}$$

$$TN_k = \left|\{d_j \mid \Phi(d_j, c_k) = false \wedge \breve{\Phi}(d_j, c_k) = false, d_j \in Te\}\right| \tag{5.2}$$

$$FP_k = \left|\{d_j \mid \Phi(d_j, c_k) = true \wedge \breve{\Phi}(d_j, c_k) = false, d_j \in Te\}\right| \tag{5.3}$$

$$FN_k = \left|\{d_j \mid \Phi(d_j, c_k) = false \wedge \breve{\Phi}(d_j, c_k) = true, d_j \in Te\}\right| \tag{5.4}$$

$$P_k = \frac{TP_k}{TP_k + FP_k} \qquad R_k = \frac{TP_k}{TP_k + FN_k} \qquad F_k = \frac{2P_kR_k}{P_k + R_k} \tag{5.5}$$

where P_k, R_k and F_k are called *precision*, *recall* and *F-measure* respectively. Using these local measures for individual categories we can compute three global measures F^{mi},F^{ma} and HL

$$P^{mi} = \frac{\sum_{i=1}^{|C|} TP_i}{\sum_{i=1}^{|C|}(TP_i + FP_i)} \qquad R^{mi} = \frac{\sum_{i=1}^{|C|} TP_i}{\sum_{i=1}^{|C|}(TP_i + FN_i)} \tag{5.6}$$

$$F^{mi} = \frac{2P^{mi}R^{mi}}{P^{mi} + R^{mi}} \qquad F^{ma} = \frac{\sum_{i=1}^{|C|} F_i}{|C|} \tag{5.7}$$

$$HL = \frac{1}{|Te|}\sum_{j=1}^{|Te|} \frac{\left|\{c_r \mid \Phi(d_j, c_r) \neq \breve{\Phi}(d_j, c_r), c_r \in C\}\right|}{|C|} \tag{5.8}$$

where F^{mi}, F^{ma} and HL are called *micro-averaged* F-measure, *macro-averaged* F-measure and *Hamming loss* respectively. Macro-averaged measure gives equal weight to each category and can be easily affected by small category, on which the classifier is giving incorrect decisions. On the other hand, micro-averaged measure gives equal weight to each document and it represent classifier decisions on most frequent categories.

5.1.3 Results

We evaluated dataset on three classification methods presented in Section 4. Before transforming documents to representation, we lower-cased all words, removed stopwords and numbers (stemming turned out to lower the accuracy in our preliminary tests, therefore we omitted it). 66% of all documents were used as a training set, the rest were used as a testing set. Evaluation was implemented in Octave[37] with our own implementations of Naive Bayes and k-NN, for SVM we used LibSVM [11] package. Octave is an interpreted language for numerical computations mostly compatible with Matlab[33].

Accuracy of SVM and k-NN is influenced by both document representation and classifier parameters (cost in case of SVM, number of nearest neighbours in case of k-NN). For this reason we tested both methods with all combinations of local and global term weighting (as presented in Section 4.2.1) and in case of k-NN we also empirically found optimal k for each combination (we simply used k on which the given combination reached its maximal F^{mi}

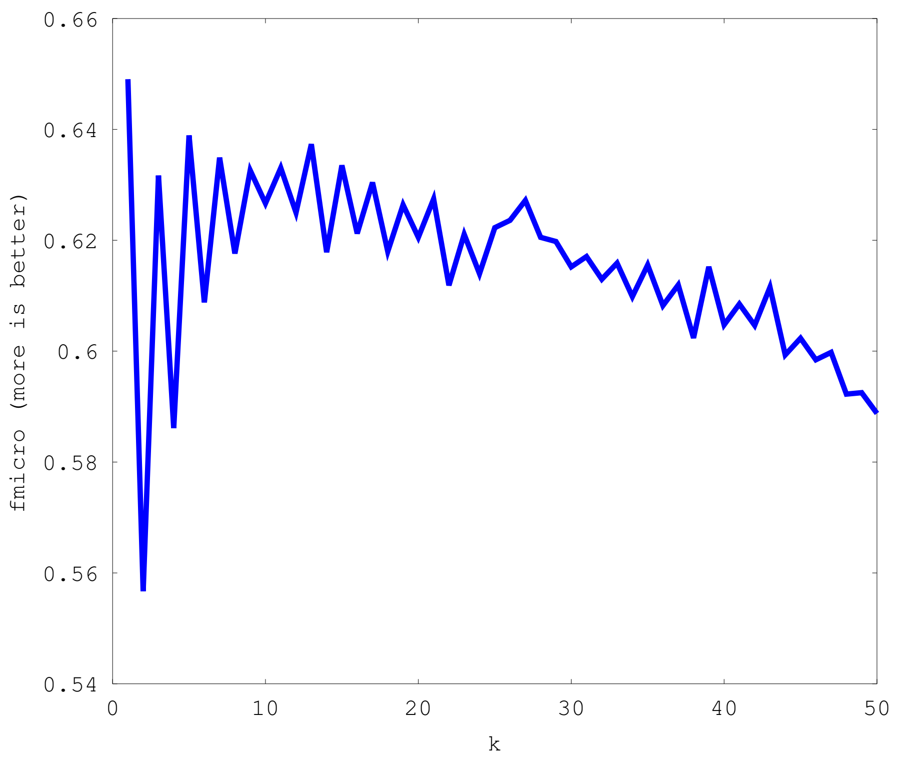


Figure 5.1: Best F^{mi} values for different k, bin-idf weighting

measure), Figure 5.1 shows an example of such graph. For SVM we used a default cost as defined by LibSVM for every combination. Found parameters are in Table 5.3. Graphs for SVM weighting combinations are in Figure 5.2 and Figure 5.3, k-NN weighting combinations are in Figure 5.4 and Figure 5.5. All representations of SVM and k-NN are normalized. We used linear kernel for SVM and negative dot product as distance function for k-NN.

Final results (only using the best performing representations - log-mi for SVM and bin-idf or k-NN) are shown in Figure 5.6, Figure 5.7, Figure 5.8 and Table 5.8.

Abbreviation	Local weight	Global weight
bin-no	binary	none
tf-no	term frequency	none
log-no	logarithmic	none
bin-idf	binary	inverse document frequency
tf-idf	term frequency	inverse document frequency
log-idf	logarithmic	inverse document frequency
bin-mi	binary	mutual information
tf-mi	term frequency	mutual information
log-mi	logarithmic	mutual information

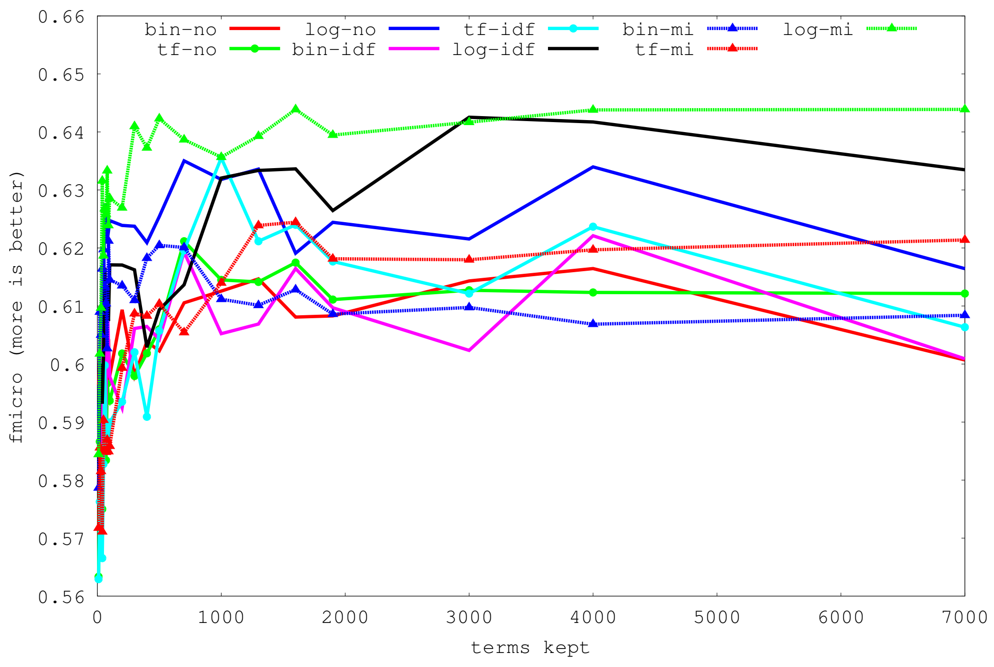
Table 5.2: Abbreviations for term weights (see Section4.2.1)

Method	Term weights	Parameter value
k-NN	bin-no	$k = 9$
k-NN	tf-no	$k = 1$
k-NN	log-no	$k = 3$
k-NN	bin-idf	$k = 1$
k-NN	tf-idf	$k = 3$
k-NN	log-idf	$k = 13$
k-NN	bin-mi	$k = 13$
k-NN	tf-mi	$k = 5$
k-NN	log-mi	$k = 7$
SVM	all	$C = 1$

Table 5.3: Optimal parameters for weighting combinations

	F^{mi}			F^{ma}			HL		
terms kept	NB	SVM	k-NN	NB	SVM	k-NN	NB	SVM	k-NN
10	0.602	0.584	0.525	0.498	0.429	0.461	0.222	0.215	0.266
20	0.630	0.602	0.509	0.546	0.444	0.445	0.223	0.207	0.272
30	0.643	0.610	0.517	0.586	0.453	0.456	0.224	0.200	0.266
40	0.648	0.632	0.525	0.593	0.491	0.456	0.226	0.190	0.265
50	0.657	0.619	0.555	0.604	0.485	0.478	0.222	0.195	0.247
60	0.651	0.627	0.541	0.602	0.491	0.467	0.228	0.194	0.257
70	0.653	0.626	0.546	0.603	0.491	0.465	0.229	0.194	0.257
80	0.656	0.633	0.547	0.610	0.497	0.475	0.228	0.190	0.258
90	0.650	0.624	0.565	0.600	0.489	0.500	0.232	0.196	0.244
100	0.649	0.629	0.569	0.602	0.494	0.511	0.232	0.195	0.243
200	0.638	0.627	0.604	0.600	0.489	0.546	0.237	0.195	0.226
300	0.639	0.641	0.611	0.590	0.502	0.541	0.230	0.188	0.219
400	0.644	0.637	0.609	0.599	0.494	0.534	0.230	0.190	0.219
500	0.647	0.642	0.587	0.604	0.506	0.503	0.224	0.186	0.228
700	0.644	0.639	0.599	0.605	0.514	0.529	0.228	0.188	0.218
1000	0.649	0.636	0.618	0.611	0.512	0.538	0.222	0.190	0.203
1300	0.651	0.639	0.632	0.611	0.516	0.551	0.221	0.187	0.197
1600	0.651	0.644	0.647	0.605	0.519	0.558	0.218	0.186	0.189
1900	0.660	0.639	0.649	0.617	0.517	0.563	0.210	0.188	0.184
3000	0.664	0.642	0.632	0.617	0.518	0.529	0.202	0.187	0.194
4000	0.667	0.644	0.641	0.617	0.519	0.553	0.197	0.185	0.184
7000	0.660	0.644	0.629	0.606	0.520	0.536	0.198	0.186	0.185

Table 5.4: Comparison of classification methods



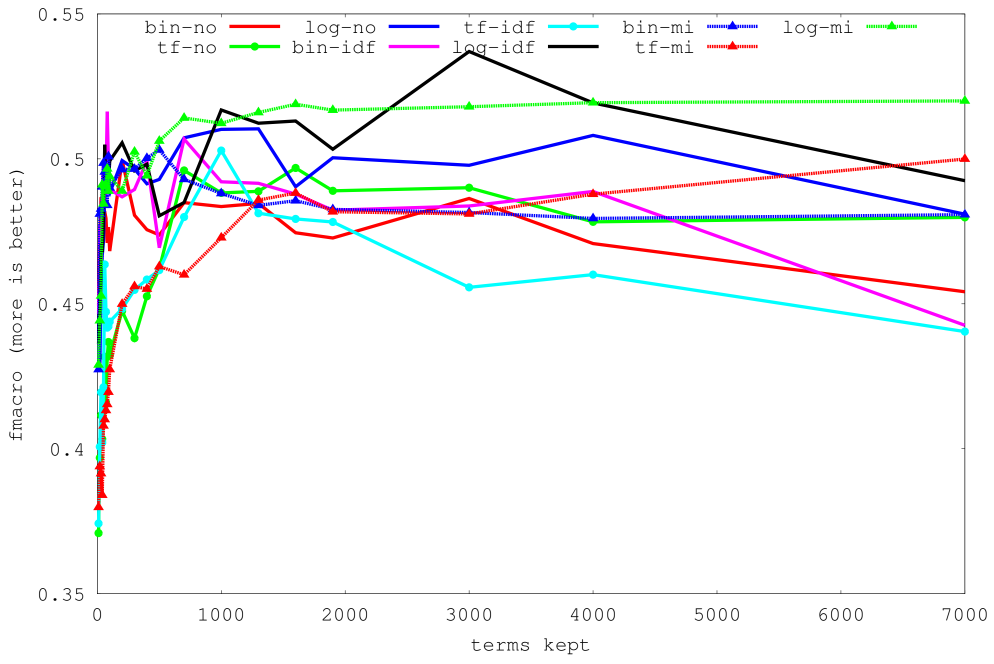


Figure 5.2: Measures for SVM weighting combinations, F^{mi} and F^{ma}

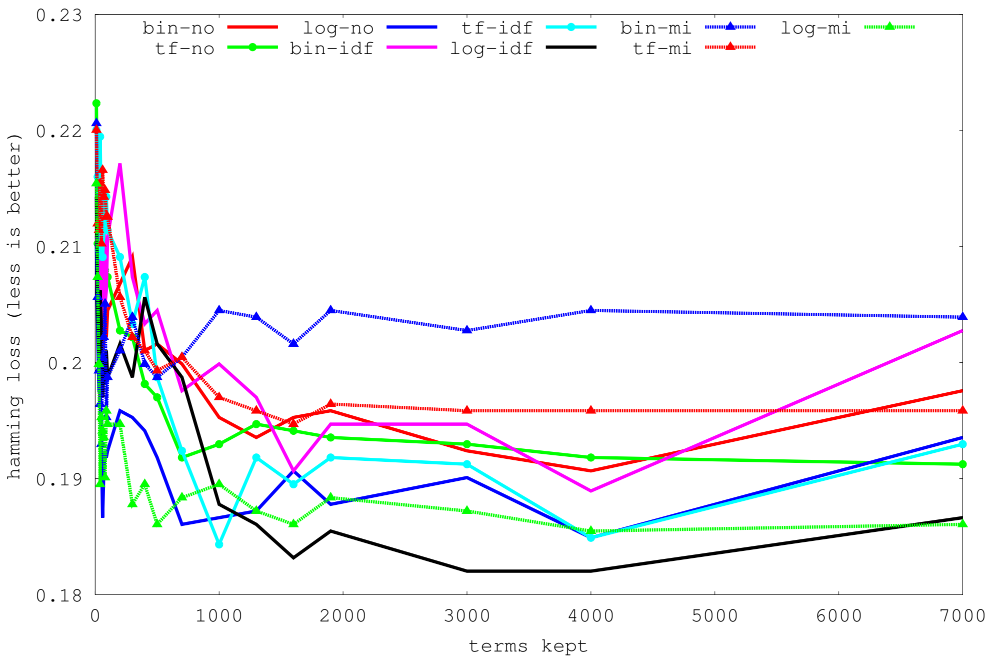


Figure 5.3: Measures for SVM weighting combinations, Hamming Loss

5.1.4 Discussion

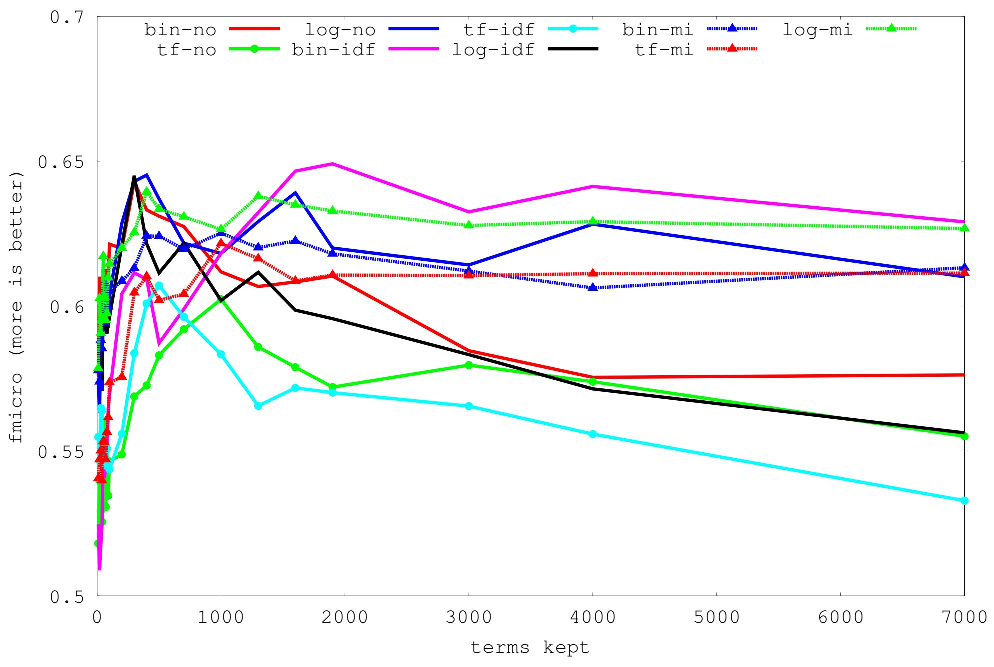
Looking at results in Figure 5.2 and Figure 5.3 we conclude that choice of representation for SVM is between log-idf and log-mi. Log-idf can achieve better values in F^{ma} and Hamming loss, however they are achieved at specific number of kept terms. Log-mi is less dependant on number of kept terms, classifier decisions are not significantly influenced by parameter choice. For these reasons, log-mi is a better choice.

On the other hand, choice of representation for k-NN is easily made, because bin-idf achieve better values on all measures (we could achieve a similar results using log-no with small number of kept terms, however this would negatively affect Hamming loss).

Comparing all methods we get an interesting conclusion that NB has better results in F^{mi} and F^{ma} than remaining methods, however by looking at Figure 5.8 we can see that in most cases, NB has higher FP ratio than SVM and k-NN meaning that NB more often recognizes an incoming document under incorrect category.

By excluding Naive Bayes, k-NN seems to be a good candidate, because we can achieve better or similar results as SVM, however k-NN has much slower classification phase. However, its disadvantage is memory complexity, k-NN needs to store the whole training set opposed to SVM, where we store only $|C|$ vectors of dimension $|T|$ and $|C|$ numbers (this is true only if we use linear kernel). For these reasons, we will use SVM in the warehouse as a classifier.

The problem with this dataset is close relation between categories, which are sometimes overlapping (e.g. algorithms and theory). Another problem is how the text is transformed to terms. In our implementation, we split words on any non-alphabetic characters, for example the sentence *3 heuristics for A* algorithm* is converted to terms *heuristics,for,a,algorithm.*



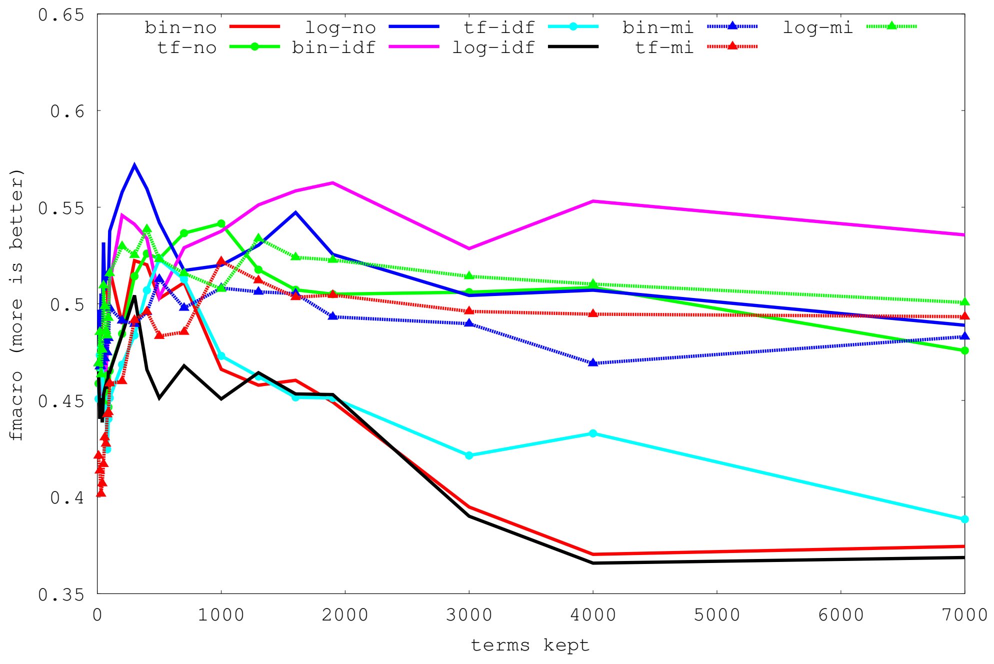


Figure 5.4: Measures for k-NN weighting combinations, F^{mi} and F^{ma}

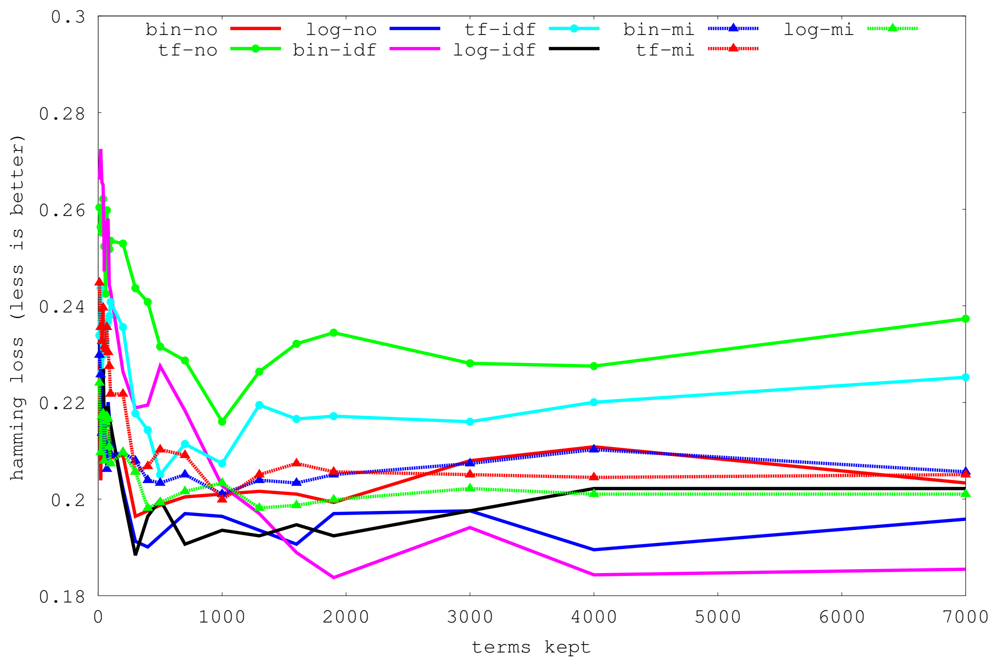


Figure 5.5: Measures for k-NN weighting combinations, Hamming Loss

Next we remove stopwords, so the original sentence is represented by terms *heuristics* and *algorithm*, which effectively removed $A*$, which would be probably a very good indicator for algorithms category. This could be solved by creating a domain-specific vocabulary, in which the tokenizer would look whether to split the examined words. However, the creation of vocabulary can't be automatized, we need a domain expert to write this vocabulary.

5.2 Warehouse

5.2.1 Testing hardware and Software

CPU: Intel Core 2 CPU 4400, 2.00GHz

RAM: 2 GB

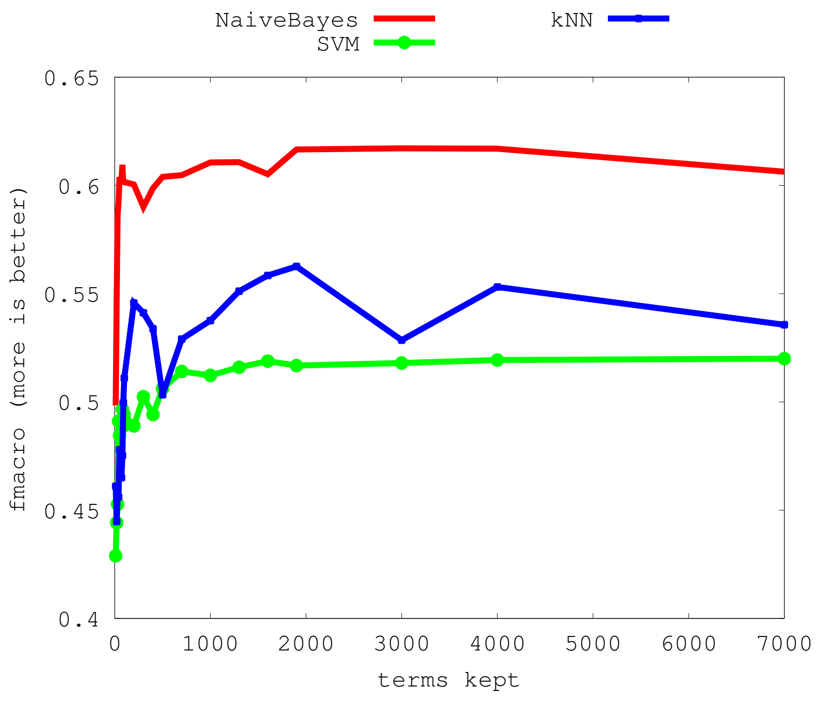
Operating system: Arch Linux, kernel version 3.3.6-1

Java Runtime Environment: OpenJDK 1.6.0_24

Web browser: Firefox 12.0

5.2.2 Server

On the server side, we measured the average time of text extraction, tagging and indexing. As a dataset, we used 265 news reports from OceanusLive in HTML format. All changes made to index are immediately committed to filesystem. Results are shown in Table 5.5.



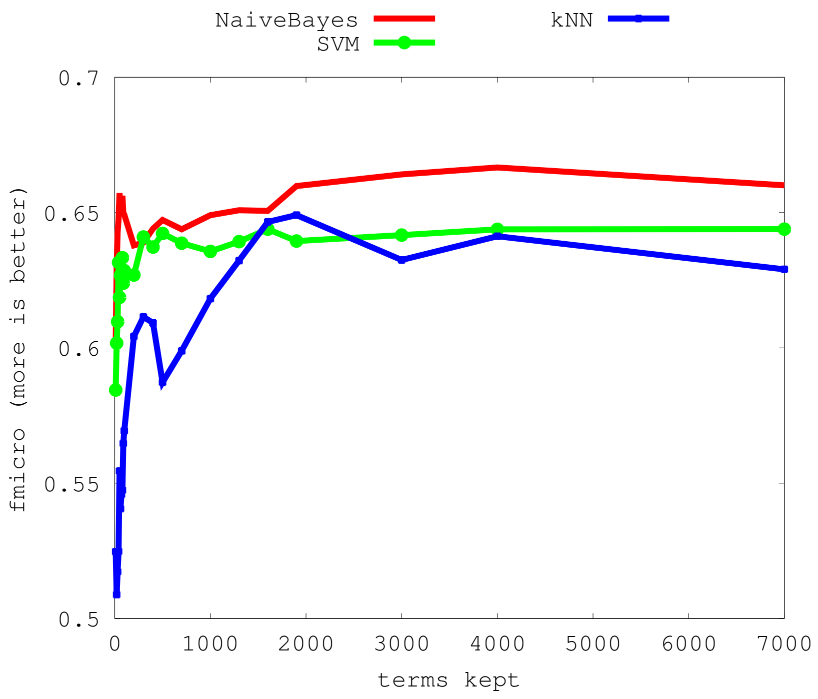


Figure 5.6: Comparison of classification methods using F^{mi} and F^{ma}

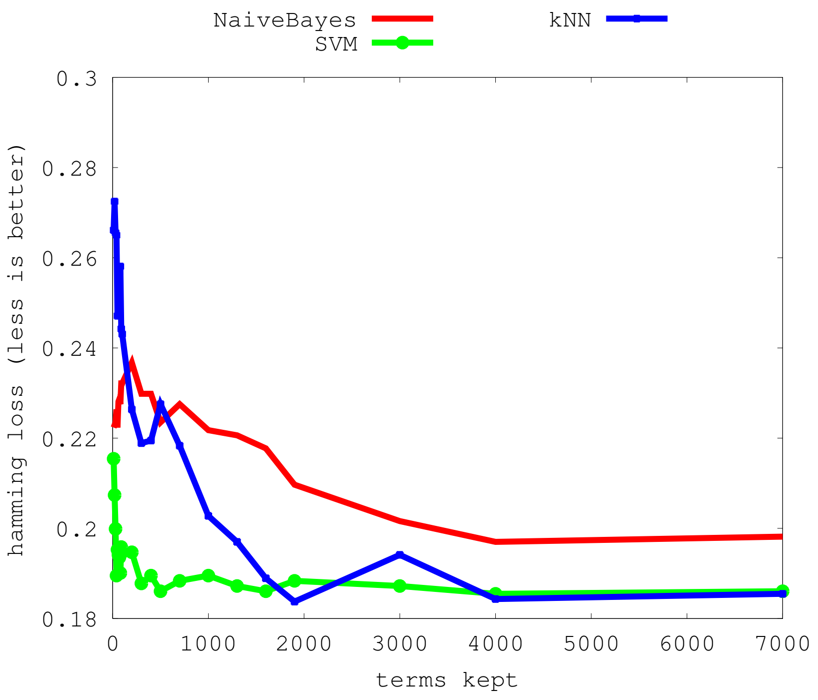


Figure 5.7: Comparison of classification methods using Hamming loss

We also measured an average time of incident extraction from KML file with 434 incidents. We ran this test ten times, the average time of extraction is 664.2 ms.

Avg. extraction time	Avg. tag time	Avg. index time	Sum
5.6226	15.732	136.34	157.69

Table 5.5: Timing for text processing (in milliseconds)

5.2.3 Client

On the client side, we only tested how many incidents can be shown on map without noticeable performance drop (these tests are subjective). About 1000 incidents can be shown without a problem, memory usage of the web browser increased about 10 MB. However, somewhere about 3600 we noticed a minor slowdown and about 4770 incidents the map's response became very slow (although still usable, if the user is patient).

theory			
	NB	SVM	k-NN
TP	55	63	66
TN	128	121	133
FP	28	35	23
FN	37	29	26

algorithms			
	NB	SVM	k-NN
TP	111	111	109
TN	77	75	71
FP	37	39	43
FN	23	23	25

performance			
	NB	SVM	k-NN
TP	16	0	9
TN	185	212	198
FP	27	0	14
FN	20	36	27

experimentation			
	NB	SVM	k-NN
TP	71	56	52
TN	106	140	140
FP	60	26	26
FN	11	26	30

economics			
	NB	SVM	k-NN
TP	45	36	37
TN	171	178	183
FP	17	10	5
FN	15	24	23

design			
	NB	SVM	k-NN
TP	36	15	20
TN	147	174	173
FP	42	15	16
FN	23	44	39

human factors			
	NB	SVM	k-NN
TP	16	9	5
TN	208	219	219
FP	14	3	3
FN	10	17	21

Figure 5.8: Number of TP, TN, FP and FN for each category (2000 terms kept)

Chapter 6

Implementation

Here we describe how each module of the warehouse is implemented. Main language used for implementation is Java[13]. Reasons are

- applications written in Java can be run on any device for which *Java Virtual Machine* exists. Most operating systems already have implementation of JVM.
- large collection of existing libraries.
- we are already familiar with the language.

6.1 Client

Client is build in Google Web Toolkit[26], which is a development toolkit for building browser-based applications in Java. GWT enables us to write web pages without any knowledge of HTML and JavaScript (code written in Java is ultimately transformed to HTML and JavaScript). This is particularly useful, because it enables to share some code packages between client and server (which is also written in Java).

For visualization of incidents on map, we tried three different libraries:

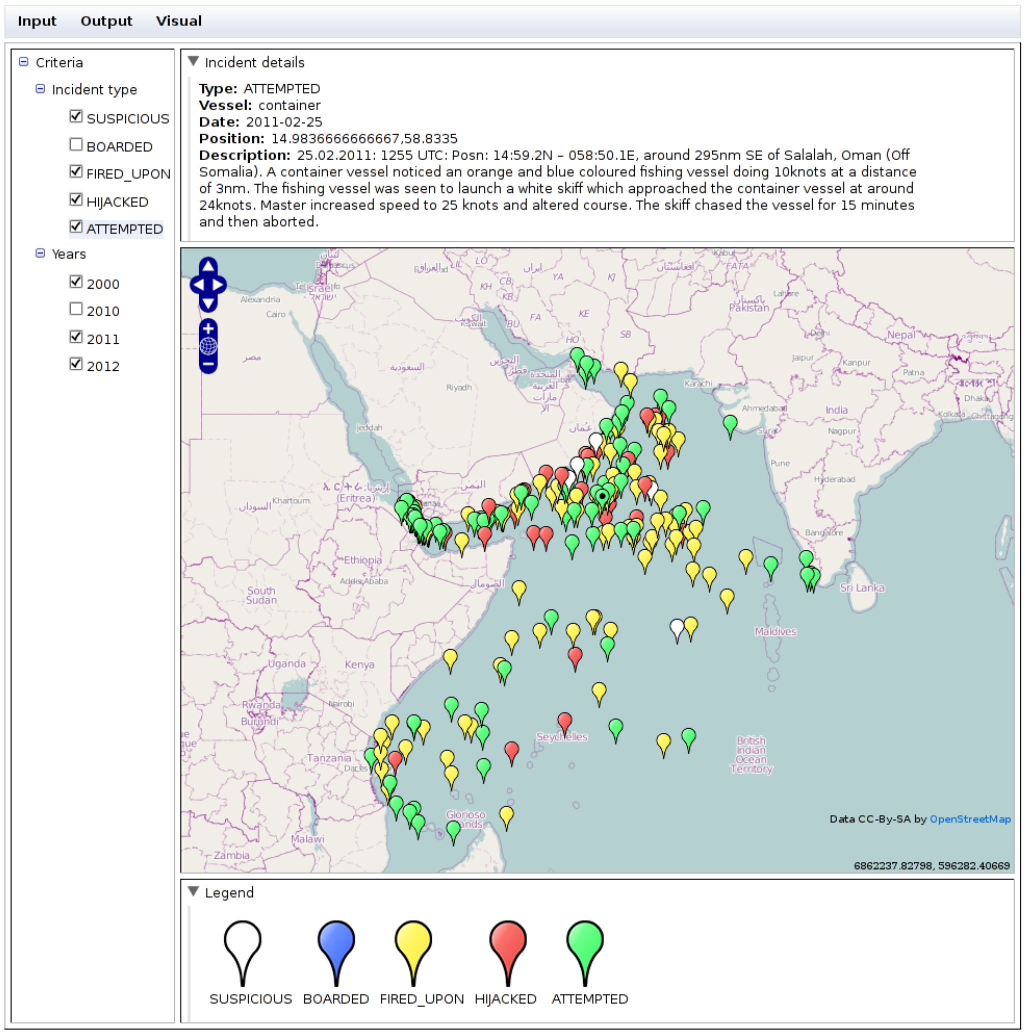
1. Official Google API for GWT[24] - official library from Google. Needs generated *API key*, which is used by Google to trace the usage of their maps and if the usage is too high, they may disable the map service (unless fee is paid). Built upon deprecated Google Maps API[27], no longer in development.
2. Unofficial binding to Google Maps API[6] - No need for *API key*, uses the newer Google Maps API. The implementation is quite buggy, we could not get the functionality we wanted. No longer in development.
3. GWT-OpenLayers[2] - library using map tiles from OpenStreetMap [36] (community project with purpose of creating free world map). Almost no documentation (for the GWT side), development is still active[1].

[1]last commit on 03.05.2012

Even though the GWT-OpenLayers is almost undocumented and the coding was done mostly by “trial and error” method, we could implement almost everything we needed, therefore it is used. However we must note that the implementation is not perfect, for example selecting incidents on map is not very accurate and pushing the selected incident marker on top layer doesn’t seem to always work.

Client also provide a simple web form for incident submission.

Figure 6.1 shows a sample screenshot from the client web page.



⊟ Criteria
- ⊟ Incident type
  - ☑ SUSPICIOUS
  - ☐ BOARDED
  - ☑ FIRED_UPON
  - ☑ HIJACKED
  - ☑ ATTEMPTED
- ⊟ Years
  - ☑ 2000
  - ☐ 2010
  - ☑ 2011
  - ☑ 2012

▼ Incident details

Type: ATTEMPTED
Vessel: container
Date: 2011-02-25
Position: 14.9836666666667,58.8335
Description: 25.02.2011: 1255 UTC: Posn: 14:59.2N – 058:50.1E, around 295nm SE of Salalah, Oman (Off Somalia). A container vessel noticed an orange and blue coloured fishing vessel doing 10knots at a distance of 3nm. The fishing vessel was seen to launch a white skiff which approached the container vessel at around 24knots. Master increased speed to 25 knots and altered course. The skiff chased the vessel for 15 minutes and then aborted.

Figure 6.1: Screenshot from the client - incident map

6.2 Server

Servlet technologies[14] are used to write server application. Servlet applications can be deployed at *web containers* (for testing purpose, Apache Tomcat[20] is used). For each client request (e.g. uploading a file, fetching incident analysis etc.) a web container invokes the appropriate servlet which processes the request and usually sends back some response.

Because the warehouse doesn't use the database extensively (we use it only for storing incidents, existing tags and some temporary information), we could base the choice of database engine on our personal experience. For these reason, we used PostgreSQL[21]. Advantages of PostgreSQL are PL/pgSQL language for writing stored functions, reasonable speed and open-source license. If PostgreSQL would not suffice in the future, it could be replaced with practically any database engine if the code using PL/pgSQL is rewritten (about 10 lines in one Java class). We also used TEXT data type for storing text strings of variable length, which is not part of SQL standard, however most databases seem to support it as well.

The warehouse can extract incidents from KML files. These files must follow syntax described in Appendix C. For parsing KML we use JAK library [35]. JAK is build on top of Java classes generated with JAXB from XML schema and provides an easier API for parsing KML than raw generated classes from JAXB.

Charts for incident analysis are generated on the server side with JFreeChart[31], client only receives URLs for generated images. Our first attempt for implementation was by generating charts on the client side with Google Charts Tools[24], however the result was slower and the implementation more complicated. Advantages were interactive charts (e.g. by hovering mouse over chart user could get some information), which is also possible with JFreeChart, but it would mean a lot of generated traffic between client and server to track the mouse motion. Charts are not cached, cache implementation would increase performance when multiple users are requesting chart analysis (server would generate each chart once for the first request and store it in cache until new incidents are added to the warehouse).

Data tasks are implemented as Java threads, which are started when server application is deployed at web container. To demonstrate the approach, current implementation downloads news from OceanusLive[2].

6.3 Indexing and searching

Indexing and searching is done with Lucene library[17]. User can write quite complex queries, currently supported searchable fields are *text* (only for files from which we can extract text), *name*, *type*, *tags* and *source* of the file. For example, query

text:“support vector machines” AND tags:(classification OR multilabel) AND type:pdf

would search for PDF files containing the phrase *support vector machines* and which are assigned to category *classification* or *multilabel*. Lucene combines *Boolean model* and Vector space model for scoring files. It first narrows down the number of files by using the Boolean model (e.g. from our example, Lucene will omit non-PDF files) and then score each file from this reduced set by VSM. For scoring it uses modified *cosine similarity* (practically normalized dot product of the query and file weights) to provide term boosting. Files are then sorted by scores and presented to the user. More information about Lucene can be found in [22].

[2] `http://www.oceanuslive.org/main/index.aspx`

6.4 Tagging library

There are many text classification Java libraries, for example (all these libraries are free to download from their respective websites):

1. Classifier4J[1] - the simplest, provides Naive Bayes classifier and *Vector Space search* (each category is represented by one document, new document is classified by cosine similarity). It is also capable of creating simple summaries of the text (by preserving sentences with most frequent words). No longer in development.

2. LingPipe[8] - library for natural language processing, classification methods include Naive Bayes, k-NN, *Perceptron* etc. Free version of LingPipe permit its use only for internal software.

3. Mahout[18] - scalable machine learning library for distributed computing, algorithms can be run in parallel on different server clusters. Algorithms for clustering, data mining and classification (Naive Bayes, *Logistic Regression*, etc.).

4. Weka[7] - machine learning library for data mining developed at University of Waikato. Weka comes as a Java library and stand-alone GUI application for experimentation.

From these libraries, Weka has probably the most extensive list of features including many classification and term selection methods (it is the only library containing all methods used in Section 5). Support Vector Machines are directly implemented in Weka and there is also a binding to LibSVM package. Since both implementations are comparable in terms of training phase time and conversion of files to document representation is quite straightforward to implement, we decided to implement our own text classification framework with LibSVM as a classifier, thus dropping the necessity to use GPL as a license for the warehouse source code (Weka is licensed under GPL). We also implemented a command-line interface, which enables to build a classifier on new training data without programming. The warehouse administrator can choose from different term weighting schemes as presented in Section 4.2.1.

As text extractors, two libraries are included, PDFBox[19] for PDF and jsoup[3] for HTML files. Tagging library can be easily extended with additional extractors, if they are needed in the future.

The tagging library needs a trained model to be able to tag files. We built a simple model from *aamas* dataset with 50 documents from maritime piracy domain. From *aamas* we removed design and performance categories (due to low accuracy) and we added a new category for piracy documents. This dataset was evaluated on 4000 kept terms (other evaluation parameters remain the same as in Section 5 with log-mi weighting), the results are in Figure 6.2.

Category	TP	TN	FP	FN
theory	60	130	29	30
algorithms	112	85	32	20
experimentation	60	134	33	22
economics	36	182	9	22
piracy	19	230	0	0
human factors	12	220	3	14

Measure	Value
F^{mi}	0.736
F^{ma}	0.742
HL	0.143

Figure 6.2: Results for built tagger model

Chapter 7

Conclusion

In this work, we studied the problem of creating intelligent data warehouse for maritime piracy domain. We created a client-server application accessible from web browser. We also studied the task of text classification and conducted an evaluation on domain specific dataset with three classification methods (Naive Bayes, Support Vector Machines and k-nearest neighbours). Gained knowledge was used to implement automatic categorization module of text files in our warehouse. Current categorization module makes averagely one incorrect category decision for incoming documents (we are assuming that incoming documents are "similar" to documents, upon which the module was trained). Stored documents can be retrieved using search queries. The warehouse also provide a basic analysis of collected incidents in form of charts and visualization of incidents on a map.

A few things could be implemented to enrich the warehouse. For example, summaries of text files could be shown in search results - this task is called *text summarization*. With this feature, user would immediately see whether the files returned by indexer are relevant to his query.

Currently there is no mechanism for login, therefore the warehouse is suitable for internal use only. It would be better if only registered and logged users could upload files and submit incidents. The warehouse administrator could then delete all files and incidents submitted by user, who has been identified as a spammer.

References

[1] Classifer4J. `http://classifier4j.sourceforge.net/`.

[2] GWT-OpenLayers. `https://bitbucket.org/gwtopenlayers/gwt-openlayers`.

[3] jsoup - Java library for working with HTML. `http://jsoup.org/`.

[4] OpenKM - document managment system. `http://www.openkm.com/en/`.

[5] Pentaho Business Analytics. `http://www.pentaho.com/`.

[6] Unofficial Google Maps API library. `http://code.google.com/p/gwt-google-maps-v3/`.

[7] Weka - data mining software. `http://www.cs.waikato.ac.nz/ml/weka/`.

[8] Alias-I. LingPipe - toolkit for text processing. `http://alias-i.com/lingpipe/`.

[9] Autonomy Corporation. Intelligent Data Operating Layer. `http://www.autonomy.com/`.

[10] Christopher J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[11] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[12] Hinrich Schutze Christopher D. Manning, Prabhakar Raghavan. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[13] Oracle Corporation. Java programming language. `http://openjdk.java.net/projects/jdk7/`.

[14] Oracle Corporation. Java servlet technology overview. `http://www.oracle.com/technetwork/java/javaee/servlet/index.html`.

[15] Franca Debole and Fabrizio Sebastiani. Supervised term weighting for automated text categorization. In Spiros Sirmakessis, editor, *Text Mining and its Applications*, Number 138 in the “Studies in Fuzziness and Soft Computing” series, pages 81–98. Physica-Verlag, Heidelberg, DE, 2004.

[16] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification.* Wiley-Interscience, 2 edition, 2001.

[17] Apache Software Foundation. Lucene - search engine library. `http://lucene.apache.org/`.

[18] Apache Software Foundation. Mahout - machine learning library. `http://mahout.apache.org/`.

[19] Apache Software Foundation. PDFBox - Java tool for working with PDF. `http://pdfbox.apache.org/`.

[20] Apache Software Foundation. Tomcat - servlet container. `http://tomcat.apache.org/`.

[21] PostgreSQL Global Development Group. PostgreSQL - object-relational database system. `http://www.postgresql.org/`.

[22] Erik Hatcher and Otis Gospodnetic. *Lucene in Action.* Manning Publications, 2010.

[23] Dropbox Inc. Dropbox. `https://www.dropbox.com/`.

[24] Google Inc. Google API for GWT. `https://code.google.com/p/gwt-google-apis`.

[25] Google Inc. Google Docs. `https://docs.google.com`.

[26] Google Inc. Google Web Toolkit. `https://developers.google.com/web-toolkit/`.

[27] Google Inc. Official Google Maps API. `https://developers.google.com/maps/`.

[28] Thorsten Joachims. Text categorization with support vector machines: learning with many relevant features. In Claire Nédellec and Céline Rouveirol, editors, *Proceedings of ECML-98, 10th European Conference on Machine Learning*, pages 137–142, Chemnitz, DE, 1998. Springer Verlag, Heidelberg, DE. Published in the “Lecture Notes in Computer Science” series, number 1398.

[29] David D. Lewis. Evaluating text categorization. In *In Proceedings of Speech and Natural Language Workshop*, pages 312–318. Morgan Kaufmann, 1991.

[30] David D. Lewis. Naive (bayes) at forty: The independence assumption in information retrieval. In Claire Nédellec and Céline Rouveirol, editors, *Proceedings of ECML-98, 10th European Conference on Machine Learning*, pages 4–15, Chemnitz, DE, 1998. Springer Verlag, Heidelberg, DE. Published in the “Lecture Notes in Computer Science” series, number 1398.

[31] Object Refinery Limited. JFreeChart - Java library for creating charts. `http://www.jfree.org/jfreechart/`.

[32] Silvano Maffeis. *Encyclopedia of Computer Science*, chapter Client-server computing. Wiley, 2003.

[33] MathWorks. Matlab - programming environment for numerical computations. `http://www.mathworks.com/products/matlab/?s_cid=wiki_matlab_2`.

[34] Andrew McCallum and K. Nigam. A comparison of event models for naive bayes text classification. In *Proceedings of AAAI-98, Workshop on Learning for Text Categorization*, 1998.

[35] MicromataLabs. Java API for KML. `http://labs.micromata.de/display/jak/Home`.

[36] OpenStreetMap Foundation. OpenStreetMap. `http://www.openstreetmap.org/`.

[37] GNU Project. Octave - language for numerical computations. `http://www.gnu.org/software/octave/`.

[38] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, 2002.

[39] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. Mining multi-label data. *Data Mining and Knowledge Discovery Handbook*, 2010.

[40] Yiming Yang and Xin Liu. A re-examination of text categorization methods. In Marti A. Hearst, Fredric Gey, and Richard Tong, editors, *Proceedings of SIGIR-99, 22nd ACM International Conference on Research and Development in Information Retrieval*, pages 42–49, Berkeley, US, 1999. ACM Press, New York, US.

[41] Yiming Yang and Jan O. Pedersen. A comparative study on feature selection in text categorization. In Douglas H. Fisher, editor, *Proceedings of ICML-97, 14th International Conference on Machine Learning*, pages 412–420, Nashville, US, 1997. Morgan Kaufmann Publishers, San Francisco, US.

Appendix A

List of abbreviations

API Application Programming Interface

CSV Comma Separated Values

GPL GNU General Public License

GWT Google Web Toolkit

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

JAXB Java Architecture for XML Binding

JVM Java Virtual Machine

KML Keyhole Markup Language

k**-NN** k-nearest neighbours

MIME Multipurpose Internet Mail Extensions

ML Machine learning

NB Naive Bayes

PDF Portable Document Format

SQL Structured Query Language

SVM Support Vector Machines

URL Uniform Resource Locator

VSM Vector Space Model

XML Extensible Markup Language

Appendix B

User guide

B.1 Submitting incidents

Select *Incident* menu item from *Input* menu, screenshot is in Figure B.1.

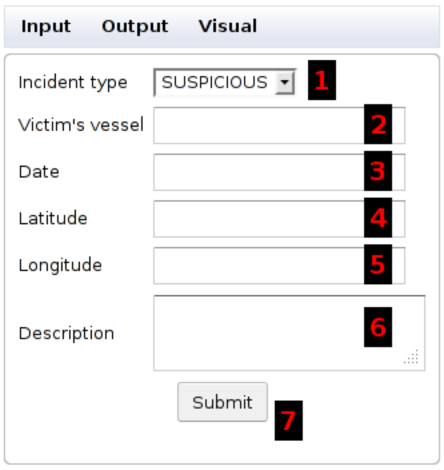


Figure B.1: Submitting incident

1. listbox for selecting the *Incident type* (currently supported types are SUSPICIOUS, BOARDED, FIRED_UPON, HIJACKED and ATTEMPTED).

2. type of *Victim's vessel* (e.g. chemical tanker).

3. by clicking on this textbox, you will be presented with *Date picker* widget. Use it for selecting the *Date*, when the incident occurred.

4. *Latitude* coordinate of the incident.

5. *Longitude* coordinate of the incident.

6. additional *Description* for the incident.

7. clicking on this button will *Submit* the incident (if all necessary fields were specified and are valid).

B.2 File upload

Select *File* menu item from *Input* menu, screenshot is in Figure B.2.

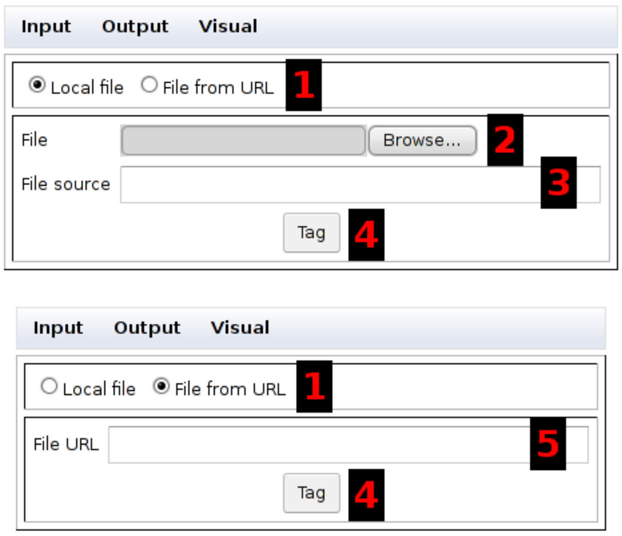


Figure B.2: File upload

1. with these radio buttons you can choose whether to upload a file from your computer (*Local file*) or from web page (*File from URL*).

2. clicking on the *Browse* button will open a file dialog, where you can locate the file to be uploaded.

3. original source of the file (e.g. web page). This textbox can't be empty!

4. clicking on the *Tag* button will upload the selected file/URL to the server, which will send back suggested tags (if any).

5. if you want to upload any publicly available resource from the web, you must provide its URL to this textbox. This textbox can't be empty!

After clicking on the button *Tag*, new page will be shown, where you can specify tags for the file/URL, screenshot is in Figure B.3.

1. *Existing tags* on the server. Clicking on any *tag* will add/remove this *tag* to/from *Selected tags*. You can hide this panel by clicking on *Existing tags* label.

2. *Selected tags* for the file. When this page loads, suggested *tags* from server will be filled in this textbox. *Tags* are separated by commas. You can write your own *tags*, however their names will be sanitized (i.e. removing trailing whitespaces, lowercasing, replacing spaces by underscore etc.).

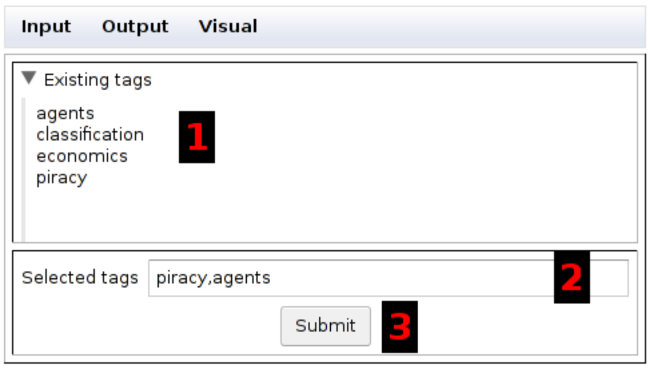


Figure B.3: File tagging

3. clicking on this button will open a confirmation dialog with sanitized *tags*. By confirming, *tags* are submitted to the server.

B.3 File search

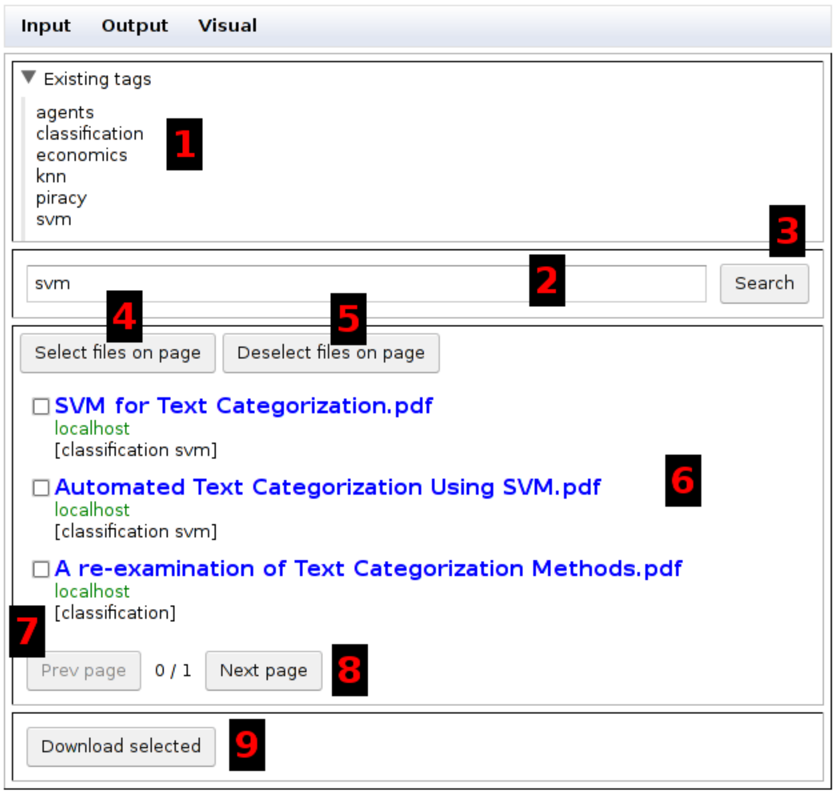
Select *Search* menu item from *Output* menu, screenshot is in Figure B.4.

1. *Existing tags* on the server. You can hide this panel by clicking on *Existing tags* label.
2. textbox for writing *search query*. Supported *fields*[1] for searching are *text*, *tags*, *type* and *source*. Default *field* is *text*.
3. button to perform the *search query*.
4. select all files on currently visible *page* for download.
5. deselect all files on currently visible *page* for download.
6. currently visible *page* of query results (i.e. files). Each result consists of a checkbox, name (blue text), file source (green text) and list of assigned tags (black text, tags are separated by space). Clicking on any component of the result will toggle the selection of that result for download.
7. button to fetch the previous *page* of results (if exists).
8. button to fetch the next *page* of results (if exists).
9. button to download all selected files. If more than one file is selected, then files are sent in ZIP archive.

B.4 Incident map

Select *Map* menu item from *Visual* menu, screenshot is in Figure B.5.

[1]Definition of *field* along with description of the query syntax can be found at `http://lucene.apache.org/core/3_6_0/queryparsersyntax.html`



agents
classification
economics
knn
piracy
svm

svm

SVM for Text Categorization.pdf
localhost
[classification svm]

Automated Text Categorization Using SVM.pdf
localhost
[classification svm]

A re-examination of Text Categorization Methods.pdf
localhost
[classification]

Figure B.4: File search

1. *Criteria* tree, from which you can select visible *Incident types* and *Years*. You must select at least one *Incident type* and *Year*, otherwise nothing will be shown!

2. the *Incident map*. You can change your current view position by dragging the mouse or by using the *directional arrows* in 7.

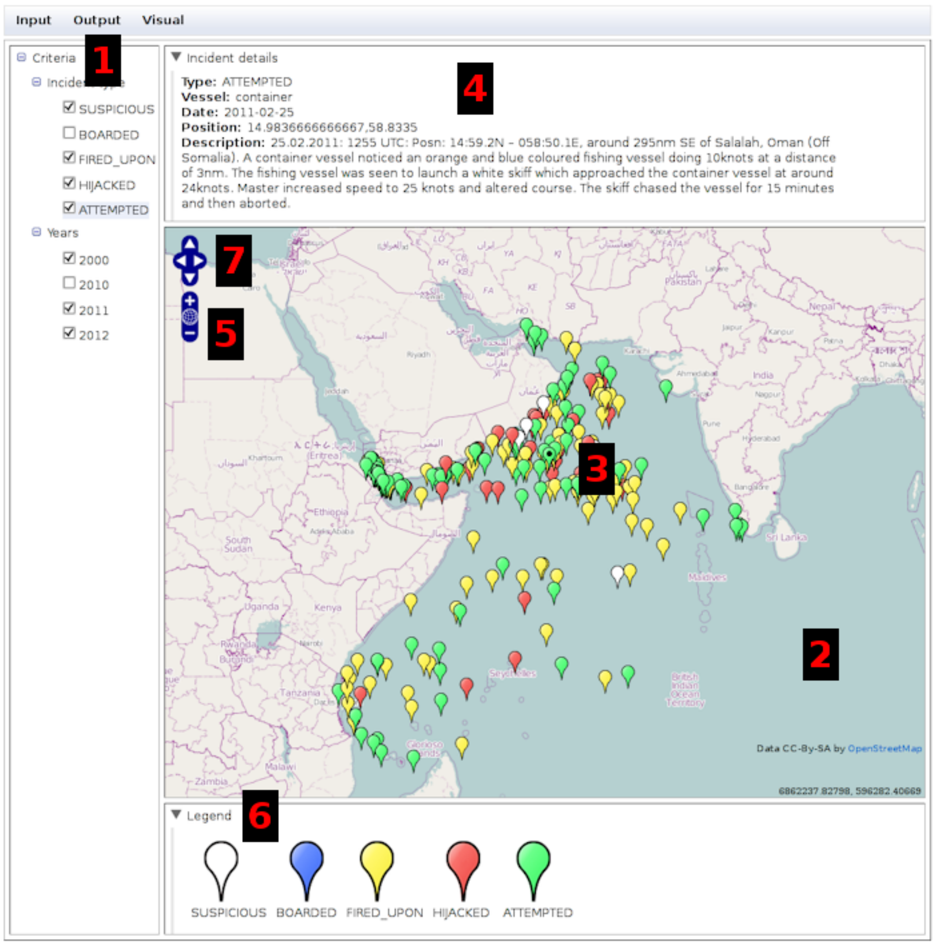
3. visible *incidents*. By clicking on *incident marker*, *details* about the *incident* will be shown. The selected *incident* is distinguished from others by drawing its *marker* with black circle.

4. *details* about the selected *incident*. You can hide this panel by clicking on *Incident details* label.

5. buttons for *zoom*. Clicking on the + will *zoom in* the *map*, clicking on to the − will *zoom out* the *map*.

6. *Legend* of incident *markers*, each *Incident type* has its own *marker*. You can hide this panel by clicking on *Legend* label.

7. *directional arrows* for changing the current view position.



Type: ATTEMPTED
Vessel: container
Date: 2011-02-25
Position: 14.9836666666667,58.8335
Description: 25.02.2011: 1255 UTC: Posn: 14:59.2N - 058:50.1E, around 295nm SE of Salalah, Oman (Off Somalia). A container vessel noticed an orange and blue coloured fishing vessel doing 10knots at a distance of 3nm. The fishing vessel was seen to launch a white skiff which approached the container vessel at around 24knots. Master increased speed to 25 knots and altered course. The skiff chased the vessel for 15 minutes and then aborted.

Figure B.5: Incident map

B.5 Incident charts

Select *Analysis* menu item from *Visual* menu, screenshot is in Figure B.6.

1. *Criteria* tree, from which you can select visible *Incident types* and *Years* in *charts*. You must select at least one *Incident type* and *Year*, otherwise nothing will be shown!

2. *Per year chart type* is a group bar chart, where group is represented by *Incident type* and the length of each bar represents the number of *incidents* in the given *Year* for that type. You can hide this *chart* by clicking on the *Per year* label.

3. *Per month chart type* is a line chart for each *Incident type*, where the value is represented by the number of *incidents* for given month and year. You can hide this *chart* by clicking on the *Per month* label.

4. buttons to download CSV files for each *chart type*.

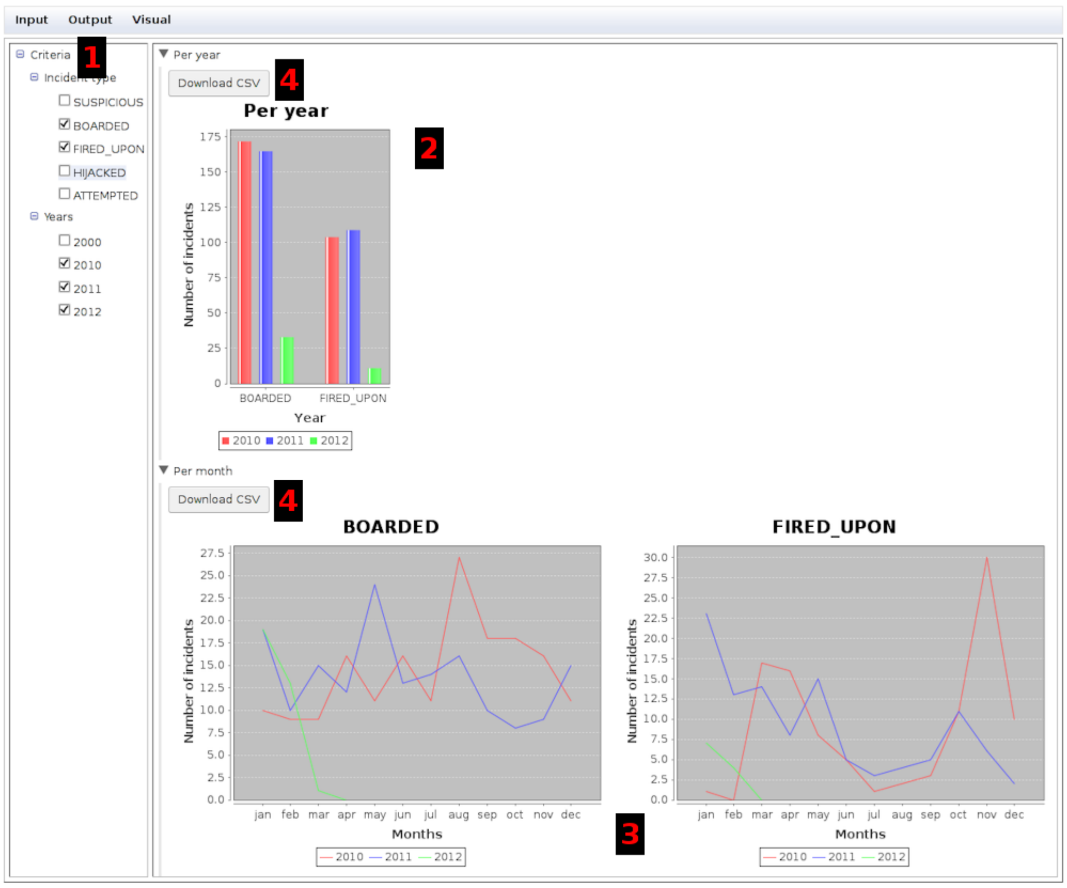


Figure B.6: Incident charts

Appendix C

KML syntax for incident submission

This section provides an internal syntax for incident submission in KML files[1]. Incidents are represented as a *Placemark* feature contained in one root *Folder* container (*Folder* can contain arbitrary number of incidents). The *Placemark* syntax for incident is in Fig. C.1. Mandatory values are denoted by square brackets, optional by curly brackets. Definition of values is the same as in Appendix B.1 except for *Date*, which is YYYY-MM-DD form.

```
<Placemark>
  <name>[Incident type],{Vessel type}</name>
  <TimeStamp>
    <when>[Date]</when>
  </TimeStamp>
  <Point>
    <coordinates>[Longitude],[Latitude]</coordinates>
  </Point>
  <description>{Description}</description>
</Placemark>
```

Figure C.1: Placemark syntax

[1]`https://developers.google.com/kml/` contains a reference and tutorial for working with KML

Appendix D

CD content

Root directory of the CD contains

- ACDataStorage - directory of the warehouse's Eclipse[1] project. Contains the source code for both server and client side. Deployable WAR file can be generated by *compile_build* bash script.
- eval - directory containing the Octave implementation of the evaluation.
- LibTag - directory of the tagging module's Eclipse project.
- libtag.jar - a command-line interface for tagging library.
- README - some general information about configuration and starting the warehouse.
- report - directory with source codes of this report. The generated PDF file is report/-main.pdf
- tagger - a trained tagger model, which can be used by the tagging module. Generated from *aamas* dataset and some documents from maritime piracy domain.

[1]`http://www.eclipse.org/`