



# First Phase Prototype Report

Fraunhofer IAIS  
Version 05  
2. October 2007



IST 2006 027600

AntiPhish

Anticipatory Learning for Reliable Phishing Prevention

Specific Targeted Research or Innovation Project

2.4.3 Towards a global dependability and security framework

## D 6.1 First Phase Prototype Report

Due date of deliverable: M20 (30 August 2007)

Actual submission date: 02 October 2007

Start date of project: 01. Jan. 2006

Duration: 36 months

Lead Contractor for this Deliverable: Fraunhofer IAIS

Revision: 05

|  |   |   |
|--|---|---|
| <b>Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)</b> |   |   |
| <b>Dissemination Level</b>   |   |   |
| <b>PU</b>  | Public  | X |
| <b>PP</b>  | Restricted to other programme participants (including the Commission Services)        |   |
| <b>RE</b>  | Restricted to a group specified by the consortium (including the Commission Services) |   |
| <b>CO</b>  | Confidential, only for members of the consortium (including the Commission Services)  |   |

## Revision history

| Deliverable administration and summary   |   |                            |
|--|---|----------------------------|
| Project acronym: AntiPhish   |   | <b>ID:</b> IST-2006-027600 |
| <b>Document identifier:</b>  | AntiPhish-del-D61-FirstPhasePrototypeReport-f-v05 |                            |
| Leading partner: Fraunhofer IAIS   |   |                            |
| Report version: v05  |   |                            |
| <b>Report preparation date:</b> 2. Oct. 2007   |   |                            |
| <b>Classification:</b> Public  |   |                            |
| <b>Nature:</b> Report  |   |                            |
| <b>Author(s) and contributors:</b> Gerhard Paaß, Andre Bergholz, Jeong-Ho Chang, Thorsten Merten, Anja Pilz, Frank Reichartz, Siehyun Strobel, Kejun Xu in collaboration with all partners |   |                            |
| <b>Status:</b>   |   | Plan                       |
|  |   | Draft                      |
|  |   | Working                    |
|  | X   | Final                      |
|  |   | Submitted                  |
|  |   | Approved                   |

The AntiPhish © Consortium has addressed all comments received, making changes as necessary. Changes to this document are detailed in the change log table below.

| Date          | Edited by    | Status  | Changes made                          |
|---------------|--------------|---------|---------------------------------------|
| -             | DoW          | Plan    | report template                       |
| 20.Aug.2007   | Gerhard Paaß | Draft   | A first draft for the project meeting |
| 24.Sept. 2007 | Gerhard Paaß | Working | Include proposals of partners         |
| 02.Oct. 2007  | Gerhard Paaß | Final   | Finalized for submission to EC        |

Notice that other documents may supersede this document. A list of latest public AntiPhish deliverables can be found at the AntiPhish webpage at [www.AntiPhishResearch.org/publications](http://www.AntiPhishResearch.org/publications).

## Copyright

This report is © AntiPhish Consortium 2007. Its duplication is allowed only in the integral form for anyone's personal use for the purposes of research or education.

## Citation

Gerhard Paaß, Andre Bergholz, Jeong-Ho Chang, Thorsten Merten, Anja Pilz, Frank Reichartz, Siehyun Strobel, Kejun Xu (2007). Deliverable D6.1 First Phase Prototype Report. Fraunhofer IAIS, AntiPhish Consortium , [www.antiphishresearch.org](http://www.antiphishresearch.org)

## Acknowledgements

The work presented in this document has been conducted in the context of the EU Framework Programme project IST 2006 027600 AntiPhish. AntiPhish is a 36-month project that started on January 1st, 2006 and is funded by the European Commission as well as by the industrial partners. Their support is appreciated.

The partners in the project are Fraunhofer Institute for Intelligent Analysis and Information Systems (FHG), Symantec LIRIC Limited (LIRIC), Symantec Ltd. (Symantec Ireland), TISCALI Services S.r.l. (Tiscali) and K. U. Leuven / ICRI-LIIR (K.U. Leuven). The content of this document is the result of extensive discussions within the AntiPhish© Consortium as a whole.

## More information

Public AntiPhish reports and other information pertaining to the project are available through AntiPhish public web site under [www.antiphishresearch.org](http://www.antiphishresearch.org).

## Table of contents

|  |    |
|--|----|
| Executive summary .....  | 6  |
| 1 Introduction .....   | 7  |
| 2 Machine Learning Framework .....                             | 9  |
| 2.1 Knowledge Discovery Processes and Operator Workflows ..... | 9  |
| 2.2 Operator Basics .....                                      | 10 |
| 2.3 Input and Output Objects .....                             | 11 |
| 2.4 Data Objects .....   | 11 |
| 2.5 Distributed Execution .....                                | 12 |
| 2.5.1 Proactive .....  | 12 |
| 2.5.2 Parallel Operators .....                                 | 13 |
| 2.5.3 Interfaces to other Frameworks .....                     | 14 |
| 2.6 Experiment Configuration Files .....                       | 14 |
| 3 Data and Features.....                                       | 16 |
| 3.1 Test Corpora.....  | 16 |
| 3.1.1 Corpora with Ham and Phishing Emails only .....          | 16 |
| 3.1.2 Email Corpora .....                                      | 16 |
| 3.1.3 Public Corpora .....                                     | 17 |
| 3.1.4 Financial Mails.....                                     | 17 |
| 3.2 Investigated Email Features.....                           | 17 |
| 4 Evaluation Criteria .....                                    | 18 |
| 4.1 Requirements .....   | 18 |
| 4.2 F-Value .....  | 19 |
| 4.3 Cross validation and Comparison of Methods .....           | 19 |
| 5 Results of Experiments with the first Prototype.....         | 21 |
| 6 Conclusions and future work.....                             | 22 |
| 7 References .....   | 23 |

## Executive summary

This document describes and documents the first prototype of the AntiPhish Learning Platform (APL) which is the outcome of WP6. A corresponding milestone M6.1 was the software demonstration at the end of the first phase in August 2007. The report concentrates on the software architecture of the first prototype. In addition few results are described which demonstrate the ability of the system to detect Phishing emails.

The APL architecture has the aim to allow the flexible generation of classification workflows and their efficient evaluation. The following components were implemented:

- A feature extraction subsystem to extract features from emails on demand. This system was designed in WP5 and integrated into the prototype.
- An operator workflow control system which executes nested sequences of operators for input/pre-processing, training, testing and evaluation.
- As a parallelization middleware ProActive was used, which allows the transparent distribution of operators on a network of compute nodes.
- Because of its modular architecture the operators in the APL platform are well suited for the integration into the software-framework of Brightmail AntiPhish centre.

A comprehensive documentation of the APL system based on JavaDoc comments is automatically extracted from the source code.

Since August 2006 Fraunhofer has worked on WP6 in a number of tasks:

- Online learning of classifiers
- Cascading classifiers
- Adapting to new types of phishing
- Semi-supervised learning.

The result of this work has been integrated into the APL system as a number of operators, which can be combined with various auxiliary operators. Besides the advancement of scientific research, the target of this work was the fulfilment of a number of requirements which were defined in deliverable D2.1 Yearly Updated Specification Requirements Document.

These requirements are all met by the current prototype. With respect to the filtering accuracy on a public corpus containing ham and phishing the methods collected in the prototype were able to beat the best published results and increase the f-value from 96.9% to 98.8%. This corresponds roughly to a reduction of the error of about 60%.

## 1 Introduction

The main task of WP6 is how to devise highly accurate filters that classify emails into the two classes “phishing” and “non phishing”. Such a filter needs to be adaptive and trainable based on examples of both types of emails. This whole work package has to be seen in close connection to WP5. As the tasks of feature extraction and classification are not independent from each other, there was a constant exchange between the results of both WPs. On the one hand K.U. Leuven, the work package leader of WP5, designed and implemented a feature extraction system (FES) as described in deliverable 5.1 and the upcoming deliverable 5.2. On the other hand Fraunhofer worked in WP5 on the tasks T5.3 Semantic Feature Extraction and T 5.6 Graphical Information.

For the successful development of classifiers several aspects have to be taken into account:

- The composition of workflows using modules of different granularity.
- The flexible provision of data and features.
- The reliable evaluation of estimators on test data.
- The efficient utilization of hardware resources, especially the ability to use distributed processing.

During the design of the system several alternatives were investigated. The UIMA framework of IBM offers the ability to specify workflows. It has large difficulties, however, to formulate loops, which are especially desirable for cross validation. In addition there is not much software that can be used within this framework. We also considered the Yale learning environment. It has a clean structure and nice workflow capabilities. However the support of distributed processing is low.

Therefore we decided to implement our own learning framework called AntiPhish Learning Platform (APL). It allows to formulate workflows as sequences of operators. In contrast to Yale different types of data objects may be transferred between objects. In addition we used the ProActive middleware for enabling distributed processing of operators. The feature extraction system (FES) provided by WP 5 was integrated using adaptor classes. In addition we implemented a number of features and integrated them in the FES.

The APL has extensive capabilities for splitting corpora and setting up training and test sets. These can be easily included in outer loops which are necessary for cross-validation and ensemble learning.

The platform has been conceived as an open framework. Using a skeleton Java class it is very simple to implement the functionality of new operators. They may operate on different levels, e.g. the document level, the corpus level or even on a meta level. In addition links to the Mallet machine learning package and the R statistical environment have been established. In future there will be links to Yale and Weka.

When the main parts of the platform were implemented the development of features and classifiers to be integrated in the prototype was extended. This resulted in a number of new developments which were included in the prototype:

- The development of “semantic” features which may take into account class information and are extremely successful to differentiate ham from phishing email.

- The development of an architecture combining cheap classifiers with more expensive classifiers. This allows to reduce the computational effort.
- The implementation of parameter optimization and feature selection approaches. This can happen, for instance, by changing method parameters or by introducing new features according to some optimization scheme.

The document is composed as follows. In the next section the design of the APL platform is explained in detail. In the following section the test corpora and the email features are described. Then there is a section on the criteria for evaluation and evaluation results for a public corpus.. The final section contains a summary.

## 2 Machine Learning Framework

The AntiPhish Machine Learning framework called APL is an environment for machine learning experiments. The processing is done by operators which can be recursively combined into workflows. Operators manipulate different Java objects, e.g. documents or complete document collections. There is a clearly defined interface between operators. Among others operators perform routine tasks like reading data, pre-processing observations, transforming features, evaluating and storing results. This allows the rapid implementation of new operators as routine tasks are already taken care of. In contrast to similar frameworks like Weka [Witten Frank 2005] and RapidMiner (formerly YALE) [Mierswa et al. 2006] APL incorporates the automatic distribution of tasks to a network of compute nodes and thus allows parallel executions of subtasks.

### 2.1 Knowledge Discovery Processes and Operator Workflows

Knowledge discovery (KD) processes can often be described as sequential workflows of operators. A prototypical sequence, e.g. for a classifier, consists of the following steps:

1. Read data  $D$ .
2. Split  $D$  into training and test sets  $Tr$  and  $Ts$ .
3. Do pre-processing on  $Tr$  yielding  $Tr^+$ .
4. Train a classifier on  $Tr^+$  generating a model  $M$ .
5. Do pre-processing on  $Ts$  with the same parameters as in step 3 yielding  $Ts^+$ .
6. Apply the model  $M$  to  $Ts^+$  yielding classifications  $K(Ts^+)$ .
7. Compute F-value  $f$  by comparing observed categories  $K_o(Ts^+)$  and predicted classifications  $K(Ts^+)$ .

Often such linear operator workflows are insufficient. If, for instance, the above process is done using cross validation, the workflows have to be nested. In addition nested workflows may occur if operators act on different entities, e.g. an operator that is applied to each document of a collection.

In the case of cross-validation the above workflow gets the following structure:

1. Read data  $D$ .
2. Split  $D$  into  $k$  subsets  $D_i$ ,  $i=1, \dots, k$
3. For  $i=1, \dots, k$ 
  - a. Split  $D$  into training and test sets  $Ts_i=D_i$  and  $Tr_i=D \setminus D_i$ .
  - b. Do pre-processing on  $Tr_i$  yielding  $Tr_i^+$ .
  - c. Train a classifier on  $Tr_i^+$  generating a model  $M_i$ .
  - d. Do pre-processing on  $Ts_i$  with the same parameters as in step 3 yielding  $Ts_i^+$ .
  - e. Apply the model  $M_i$  to  $Ts_i^+$  yielding classifications  $K(Ts_i^+)$ .

- f. Compute F-value  $f_i$  by comparing observed categories  $K_o(Ts_i^+)$  and predicted classifications  $K(Ts_i^+)$ .
4. Compute the average F-value  $f = \frac{1}{k} \sum_{i=1}^k f_i$

In APL the highest workflow always corresponds to the complete machine learning experiment. It can contain atomic operators which take some input objects, perform some data mining tasks and output some result objects. It may also contain meta-operators which contain other workflows, etc. In this way a hierarchy of workflows may be implemented.

If the inner workflows, as in the above example, are independent, they may be executed in parallel. APL allows to distribute the calculations of operators to different processors. This is done in a transparent way such that only minimal changes in the workflow specifications are required. The communication between operators is handled automatically.

A workflow hierarchy is specified in an XML (eXtensible Markup Language) format. XML is well suited for describing structured objects and has become a standard format for data exchange. In addition this format is easily readable by humans and machines. All APL experiments are described in an easy XML format. Sequences of operators appear as a sequence of XML-entries and nested operators are nested in the XML notation.

The operators are implemented primarily in Java. In the program code they define their expected input objects and created output objects as well as their necessary and optional parameters. This is used by APL to automatically check the consistency of operator workflows, the types of the objects passed between the operators, and the mandatory parameters. By specifying a Java wrapper also operators coded in C or other programming languages may be included.

## 2.2 Operator Basics

Each operator `<index>operator</index>` is implemented by a specific Java class which implements the Operator interface. Operators are combined to a workflow by Experiment Configuration Files described below. An operator is defined by a number of aspects:

**List of Input Arguments.** An operator requires the presence of zero or more input arguments which are Java objects. In the operator's Java implementation the name of each input argument and its type are defined in the operator's Java class, by the method `getInputSignature()`.

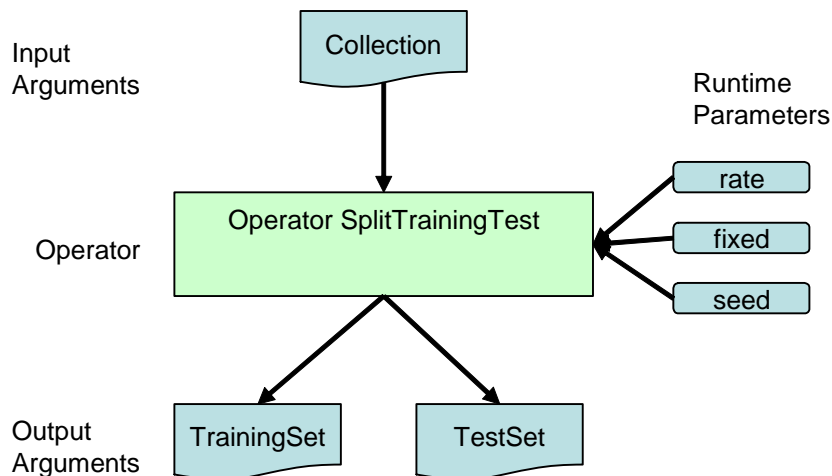
**List of Output Arguments.** During its execution an operator produces zero or more output arguments, which are Java objects. In the operator Java implementation the name of each output argument and its type are defined in the operator's Java class, by the method `getOutputSignature()`.

**List of Runtime Parameters.** Usually the execution of an operator is controlled by a number of parameters. These parameters usually are simple numbers or strings and are specified in the XML experiment configuration files. The names of

parameters, their types and their value range are defined in the operator's Java class, by the method `setupParameters()`.

**Implementation of Functionality.** The functionality of an operator is implemented by the operator Java class by the method `run()`. The execution of the operator may start if all input arguments and parameters have been supplied.

The following figure shows these parts for the operator `SplitTrainingTest`, which splits a `Collection` into a `TrainingSet` and a `TestSet`. The parameter `rate` specifies the fraction of documents to be used for the training set. The parameter `fixed` specifies if a fixed random seed is to be used and the parameter `seed` specifies a fixed seed.



**Figure 1: Example of an operator, which splits a document collection into a training and a test set.**

## 2.3 Input and Output Objects

APL's most important characteristic is to accept input objects, modify them by processing and generate output objects. This can be done in nested operator workflows. Intentionally there is no restriction on the objects which the operators can accept as inputs. This may be

- single examples or documents,
- collections of cases like corpora, training sets, test sets or even database and search engines delivering objects on demand,
- statistics like dictionaries, frequency tables, etc.
- estimated models, e.g. classifiers or semantic models.

For each input our output argument the operators specify a freely selected name and check if the corresponding Java class is correct.

## 2.4 Data Objects

The current implementation of APL is targeted towards email handling. It provides the following data objects

**Collection** describes a set of documents. It contains

- a list of Documents,
- a list of corresponding real-valued case weights. These weights have the default value 1.0.

**Document** represents a single case, example or document. It contains

- a filename, which may contain the name of the file where the document is stored,
- a node, which may contain the name of the virtual compute node in the compute network where the document is located,
- a DataPoint, which contains the a sparse vector of data and a label.
- a timeStamp indicating the Date of the document,
- a flag indicating if the document is valid or contains some errors which prevent processing.

**DataPoint** contains a sparse vector with integer indexes and real values. A LabeledDataPoint may in addition contain a class label provided as a String.

**LabelManager** contains a dictionary of possible document labels.

**FeatureManager** contains a dictionary to map feature values to identifiers and enables feature selection.

**Cache** currently contains the different features which are available for a document. These features are calculated on demand by specific feature providers. To save computation effort classifiers may require "expensive" features only for specific difficult cases. Most feature providers and the cache have been implemented by K.U. Leuven.

## 2.5 Distributed Execution

### 2.5.1 Proactive

Usually the workflow defined in an Experiment Configuration file is executed on the local machine. One of APL's important features is the possibility to distribute computing tasks to different remote computers called nodes. We use Proactive (<http://www-sop.inria.fr/oasis/ProActive/>), a Java based open source middleware for parallel, distributed and multi-threaded computing. ProActive provides a comprehensive framework and programming model to simplify the programming and execution of parallel applications: within multi-core processors, distributed on Local Area Network (LAN), on clusters and data centres, on intranet and Internet Grids.

The ProActive library is based on an Active Object pattern that is a uniform way to encapsulate a remotely accessible object, which runs in its own thread on a local or remote machine. Every Java object may be generated as an active object and the library transparently performs the transfer of data between machines. The topology of the available grid has to be specified in a ProActiveDescriptor file (see ProActive Documentation). ProActive provides the following advanced features:

- Remote objects can be created and are accessible via remote method invocation (RMI), the created objects are called Active Objects. Each Active Object has a queue for storing the incoming method calls (also called requests) and has only one thread

to execute the requests in the queue. These requests are served in first in, first out (FIFO) manner by default.

- The communication between Active Objects is asynchronous.
- Active Objects can be created in one machine and then migrated to remote machine. This feature can be used for load balancing between computers.
- The deployment is done via XML-based descriptor files, which has made the deployment away from the source code. The names of the remote machines, the communication protocols will be specified in the descriptor file.
- Several communication mechanism have been implemented, such as, rsh, ssh for secure communication purpose.
- IC2D, which is a monitor tool with GUI, is provided in ProActive for monitoring the activities of Active Objects on remote computers.

ProActive has some more advanced features, such as, group communication and component models. For *group communication*, the Active Objects are partitioned into groups. A group of remote objects can be invoked asynchronously and the replies are automatically collected. With the *component model*, composite, sequential, and parallel components can be recursively formed and a component may wrap legacy code if needed. These features may be quite useful for the project in the long run.

Although existing Java code can be used in ProActive without change, we have to take into account the communication overhead. Non-Active, "passive" objects can be the attributes of an object. When we create an Active Object on a remote machine, these passive member objects are serialized and sent to the remote machine. Therefore these passive objects have to be serializable, i.e. have to implement the `java.io.Serializable` interface. For small passive objects this is not a problem, but for large passive objects of several megabytes, the serialization can take some time. And if many passive objects have to be serialized, the network communication can cause a lot of overhead and the execution could be slowed down.

An inconvenience of ProActive is the debugging support. IC2D can be used to monitor the interactions between objects. To check the functionality of active objects on remote nodes the debugging support is more limited and usually we have to insert some printouts. But these difficulties occur with most parallel execution environments.

## 2.5.2 Parallel Operators

If for a meta-operator the "active" parameter is set to true, then the meta operator may distribute the inner operators to different nodes using the ProActive middleware. Groups of target nodes may be specified with the "nodes" tag in the Experiment Configuration File.

The system simply checks the available nodes and creates Active Objects on the remote machines. If the "active" parameter of an operator is set to false, the code of the object will run on the local machine. The processing time can be reduced by distributing operators with high computation demand to remote machines to achieve parallel execution. However, there is an overhead for starting a new JVM, serializing the passive objects, and invoking a method of the remote object. Therefore the processing speed can be slowed down, if operators with simple computation logic are distributed, because the overhead is too high compared with the time needed to execute this operator.

The "active" parameter may also be set to true for a "non-meta" operator. In this case the operator is executed on a remote node, again specified by the "nodes" tag in the Experiment Configuration File.

### 2.5.3 Interfaces to other Frameworks

APL has interfaces to other machine learning frameworks. Currently there is an interface with Mallet [McCallum 2002]. Mallet is an integrated collection of Java code useful for statistical natural language processing, document classification, clustering, information extraction, and other machine learning applications to text (see operator CreateMalletTrainer).

There exists a second interface to the statistical analysis system R. R is a programming language and software environment for statistical computing and graphics. It supports a wide variety of statistical and numerical techniques (see the operator RTest).

## 2.6 Experiment Configuration Files

Experiment configuration files are XML documents specifying the flow of actions and data in a complete Machine Learning experiment. They are composed of recursive sequences of operators and meta operators thus define complex nested workflows. In addition run-time parameters may be supplied to control the execution of operators. Parameters can have a single value or consist of a list values. Descriptions can be used to comment your operators.

The syntax of Experiment Configuration Files is described in detail in the User Manual of the AntiPhish Machine Learning framework. It contains also the description of the available machine learning operators and the description of workflows. Most parts of the user manual are directly compiled from the JavaDoc comments in the operator source code. Therefore it will always be up to date as long as the implementers maintain their Java code.

The following workflow is a typical sequence of operations which may be conveniently described by an Experiment Configuration File. It contains the input of data, preparatory transformations as well as training, test and computation of the classification performance of an SVM classifier.

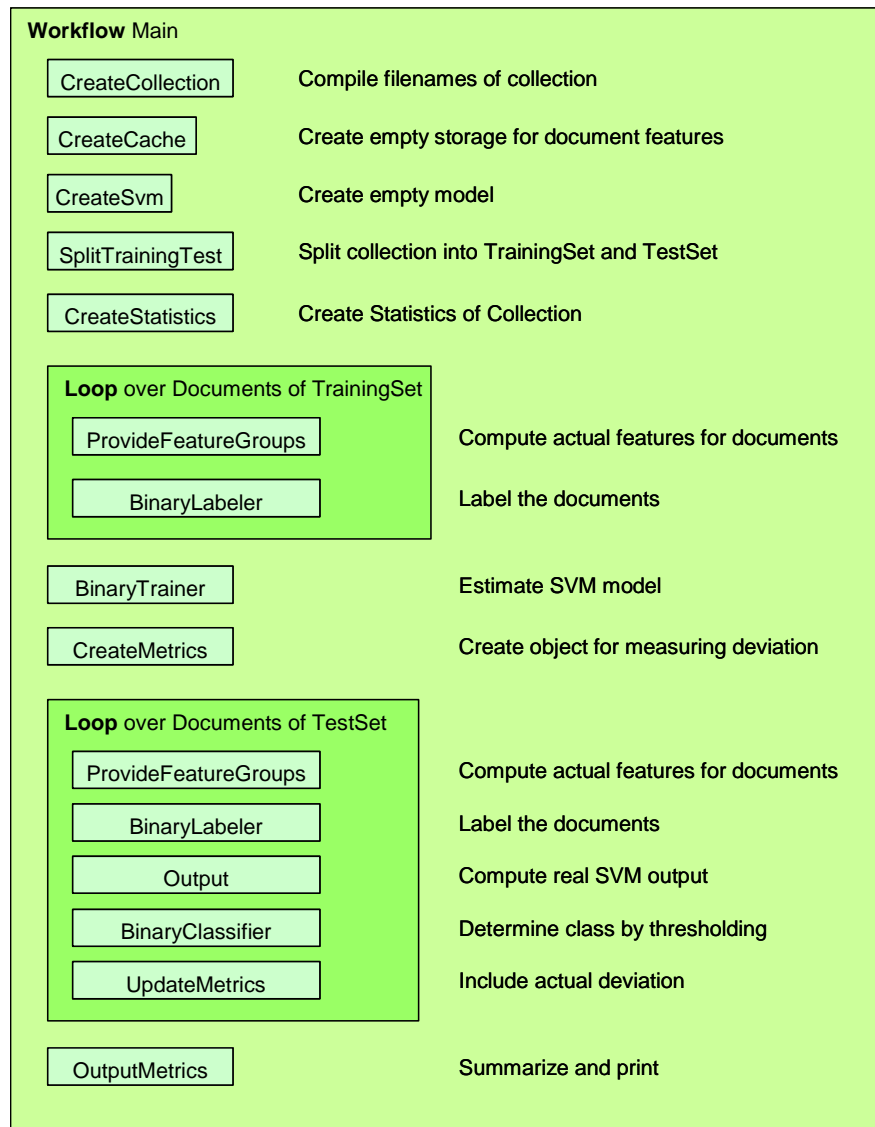


Figure 2: Workflow for training and test of an SVM classifier

## 3 Data and Features

### 3.1 Test Corpora

#### 3.1.1 Corpora with Ham and Phishing Emails only

These corpora were constructed to test the ability of filters to distinguish ham from spam. A small corpus for rapid tests and a larger corpus were formed.

**HamPhish:** The HamPhish corpus was extracted from the first set of Symantec data CDs. It contains a roughly equal number of ham and phishing emails (1018 vs. 1001) and no spam for a total number of 2019 emails.

**Bighamphish :** The Bighamphish corpus contains ham and phishing emails from September to December 2006 in the real-life distribution of 37:3. In total, there are 24518 emails, 22666 ham and 1852 phishing. This corpus was extracted from the HSP-4-months corpus described below.

#### 3.1.2 Email Corpora

**HSP-4-months:** This corpus was extracted from the 345GB delivered by Symantec. It respects the real-life distribution of 37% ham, 60% spam, and 3% phishing. It contains data from September 2006 to December 2006. Altogether it contains 61346 emails, 36828 spam, 22666 ham, and 1852 phishing. Only verified Phishing emails have been included, potential Phishing was not used. This corpus is the basis for other corpora.

**HSP-333:** An equally distributed subset of the 4-month corpus. Contains 5235 emails, 1744 ham, 1765 spam, and 1726 phishing.

**HSP-1106:** A real-life distributed one month (November 2006) subset. Contains 12103 emails, 4507 ham, 7234 spam, and 362 phishing.

During our experiments it turned out that the corpora contain a considerable number of Non-English emails, which use different character sets and somewhat distort the classification results. We decided to set up sub-corpora of English mails to determine the reliability of filtering approaches under these circumstances. To remove Non-English emails from our corpora we employed an N-Gram based text categorization [Cavnar Trenkle 1994]. This approach offers a good trade-off between running time and accuracy of the classification. For this algorithm a free implementation in Perl exist which we have used to construct our corpora. The classification sometimes suggests more than one language for a text if this happened we considered the mail as good if English where among them. The new sub-corpora are

**HSP-333eng:** The English-language subset of HSP-333 with equal counts of ham, spam and Phishing emails. Contains 4268 emails: 1237 ham, 1478 spam, and 1553 phishing.

**HSP-1106 eng:** The English-language subset of HSP-1106. Contains 9657 emails, 3157 ham, 6180 spam, and 320 phishing.

### 3.1.3 Public Corpora

Phishing and spam emails are much easier to obtain than ham (non-phishing and non-spam) emails, because ham emails may contain personal or private information. This is the main reason why there are only a few publicly available corpora containing ham emails. This creates the demand for a public corpus which can be used to evaluate and compare different approaches for phishing email detection.

**Fette07:** To evaluate their approach called PILFER for phishing email detection the authors constructed a public corpus out of two public available resources [Fette et al. 07]. The ham emails are from the SpamAssassin public corpus (<http://spamassassin.apache.org>) (both the 2002 and 2003 collections with a total of approximate 6950 ham emails). The phishing emails are from the public phishing corpus, which contains approximately 860 email messages. These phishing emails were collected during the period between November 2004 and November 2005 by John Nazzario. This corpus covers all of the common phishing schemes. The emails were labelled as “ham” if they are from the SpamAssassin ham corpus and are labelled as “phishing” if they are from the public phishing corpus.

**Table 1: Frequency of Ham, Spam and Phishing emails in the different corpora**

| corpus       | Counts |       |          |       | Percentages |      |          |
|--------------|--------|-------|----------|-------|-------------|------|----------|
|              | ham    | spam  | phishing | total | ham         | spam | phishing |
| HamPhish     | 1018   | -     | 1001     |       | 50.4        | -    | 49.6     |
| BigHamPhish  | 22666  | -     | 1852     | 24518 | 92.4        | -    | 7.6      |
| HSP-4-months | 22666  | 36828 | 1852     | 61346 | 37.0        | 60.0 | 3.0      |
| HSP-333      | 1744   | 1765  | 1726     | 5235  | 33.3        | 33.7 | 33.0     |
| HSP-1106     | 4507   | 7234  | 362      | 12103 | 37.2        | 59.8 | 3.0      |
| HSP-333eng   | 1237   | 1478  | 1553     | 4268  | 29.0        | 34.6 | 35.4     |
| HSP-1106eng  | 3157   | 6180  | 320      | 9657  | 32.7        | 64.0 | 3.3      |
| Fette07      | 6950   | -     | 860      | 7810  | 89.0        | -    | 11.0     |

### 3.1.4 Financial Mails

The financial corpus will be used to distinguish between ham emails that are finance-related (branded) and ham emails that are not. It was extracted from the 345GB delivered by Symantec. The branded ham emails were extracted using scripts delivered by Symantec. The corpus contains 84664 emails, 1854 branded and 82819 unbranded.

## 3.2 Investigated Email Features

The upcoming deliverable D 5.2 Feature Extraction report describes the available features in detail.

## 4 Evaluation Criteria

### 4.1 Requirements

In Deliverable D21 Yearly Updated Requirements Specification Document from September 2006 a number of criteria for the AntiPhish system were specified:

- **Phishing False Negative Rate** - The number of phishing messages not blocked at policy enforcement points divided by the number of phishing messages sent through policy enforcement points.  
The target for the prototype is 0.1, i.e. at most 10% of the Phishing messages may be not blocked.
- **Phishing False Positive Rate** - The number of legitimate messages blocked (as phishing) at policy enforcement points divided by the number of total messages sent through policy enforcement points.  
The desired rate for the prototype is 0.01, which means that a fraction of 1% of all messages may be blocked ham emails.
- **Recall** – The ratio of phishing messages blocked divided by the number of total phishing messages.  
This is just  $1 - \text{Phishing False Negative Rate}$ .
- **Precision** – The ratio of phishing messages blocked divided by the total number of messages blocked.  
This value is difficult to evaluate as not only Phishing emails but also spam emails may be blocked. If all messages are considered together this value may be never reached as spam messages outnumber Phishing messages. In practice it is a difficult classification task to distinguish Phishing messages from spam messages.
- **Labelled Content Processing Message Throughput** – The volume of labelled data to be processed by the analysis system in generating blocking criteria to be disseminated to enforcement points, measured by the number of messages per second through the analysis system.
- **Labelled Content Processing Volume Throughput** – The volume of labelled data to be processed by the analysis system in generating blocking criteria to be disseminated to enforcement points, measured megabytes per second through the analysis system.
- **Latency** – The latency from the time a sample of a phishing is acquired from a labelled source to the time an effective rule can be deployed for blocking the phishing messages, measured in seconds.
- **Personnel** – The number of people required to operate the system at any given time to maintain the desired performance.

According to deliverable D21 the different systems should fulfil the following criteria:

**Table 2: Performance Criteria for the Prototype**

| <b>Criterion</b>             | <b>Prototype</b> | <b>Production</b> | <b>Goal</b> |
|------------------------------|------------------|-------------------|-------------|
| Phishing False Negative Rate | 0.10             | 0.05              | 0.01        |

|   |       |       |          |
|---|-------|-------|----------|
| Phishing False Positive Rate                          | 0.01  | 0.001 | 1/1M*    |
| Recall  | 0.90  | 0.95  | 0.99     |
| Precision   | 0.95  | 0.99  | 0.999999 |
| Labelled Content Processing Message Throughput [/sec] | 12    | 1,200 | 10,000   |
| Labelled Content Processing Volume Throughput MB/sec  | 0.12  | 12    | 100      |
| Latency [sec]   | 3,000 | 300*  | 30       |
| Personnel [Persons]                                   | 1     | 20    | 5        |

## 4.2 F-Value

An important summary measure is the F-value, which is a compromise between precision and recall. It is defined as the weighted harmonic mean of precision and recall

$$F = 2 * (precision * recall) / (precision + recall)$$

During our experiments we have noticed that our corpora contain some emails whose text content is nearly identical. A good measure for the similarity of two strings e.g. email text is the so called Levenshtein distance [Levenshtein66]. It is a metric to determine the edit-distance between two strings. i.e. how many edit operations are needed to transform the one string into the other. To study our observation about duplicate emails we have conducted some test on the HamPhish corpus. We have used this metric to compute the similarity between each possible pair of phishing email in the corpus. To this end we considered an email as duplicate if the distance between two strings was less or equal to 15 % of the length of the longer string. The result for the HamPhish corpus was that 293 from the 1001 emails were identical. This implies that 29% of the phishing emails have nearly identical text content. This circumstance must be the target of further studies to determine whether it could be utilised for phishing email detection or not.

## 4.3 Cross validation and Comparison of Methods

The performance of classifiers may be estimated by an independent test set. A prerequisite is that the examples in the training and test set are mutually independent and follow the same distribution as the examples in the training data. In this case the mean of the estimated performance figures, e.g. precision, recall or f-value, is an unbiased estimator of the true performance. However, it is difficult to estimate the variance around the true value. The APL framework provides workflows for training and testing classifiers by a holdout set.

In **k-fold cross-validation**, the original sample is partitioned into  $k$  subsamples. Of the  $k$  subsamples, a single subsample is retained as the test set data for testing the model, and the remaining  $k - 1$  subsamples are used as training data. The cross-validation process is then repeated  $k$  times, with each of the  $k$  subsamples used exactly once as the test set. The  $k$  results from the folds then can be averaged (or otherwise combined) to produce a single estimation. There is a workflow for executing crossvalidation in APL. This can be done on a single compute node or in parallel on different computers.

The different performance values obtained from a cross-validation vary because of the random selection of test and training sets and are distributed around the true value. A

number of papers have investigated a way to estimate a confidence interval for the true value [Bengio Grandvalet 04]. Based on theoretical analysis and a large number of empirical evaluations [Bouckaert Frank 2004] propose the following **r-times repeated k-fold cross-validation procedure**, where  $r > 0$  and  $k > 1$ . For fold  $i \leq k$  and run  $j \leq r$  we get differences  $x_{ij} = a_{ij} - b_{ij}$  between the performance criteria for two classifiers to be compared, e.g. their f-values. Then we define estimates for the mean and variance by

$$m = \frac{1}{k \cdot r} \sum_{i=1}^k \sum_{j=1}^r x_{ij} \quad \hat{\sigma}^2 = \frac{1}{k \cdot (r-1)} \sum_{i=1}^k \sum_{j=1}^r (x_{ij} - m)^2$$

As the folds are not independent we may not use  $\hat{\sigma}^2$  directly in a significance test. [Bouckaert Frank 2004] propose an inflation factor which increases the variance. They use the following test statistic

$$t = \frac{\frac{1}{k \cdot r} \sum_{i=1}^k \sum_{j=1}^r x_{ij}}{\sqrt{\left(\frac{1}{k \cdot r} + \frac{n_2}{n_1}\right) \hat{\sigma}^2}}$$

where  $n_1$  is the number of instances used for training and  $n_2$  the number of instances used for testing. The test is called “corrected repeated k-fold cv test”. Theoretical analysis and many empirical experiments show that this procedure can reliably compare the quality of different classifiers [Bouckaert Frank 2004]. APL provides a workflow to evaluate this test.

## 5 Results of Experiments with the first Prototype

Recently [Fette et al. 2007] published results on Phishing detection. They used 10, e.g. contains IP-based URLs (are there URLs with IP numbers), age of linked-to domain names, contains nonmatching URLs (is the displayed link text different from the URL), “Here” links to non-modal domain (links to infrequently called domains), contains html content, number of links, number of domains (last two ids), number of dots (in links), contains JavaScript, SpamAssassin output. [Fette et al. 2007] applied their PILFER algorithm to corpus publicly available “Fette” corpus. It contains 6954 non-phishing and 857 phishing emails. Their results are listed in the first row of the following table.

| Approach                                 | % F-measure | % Precision | % Recall | % FP  | % FN |
|--|-------------|-------------|----------|-------|------|
| PILFER [Fette et al. 2007]<br>10-fold CV | 96.9        | 97.9        | 95.9     | 0.240 | 3.6  |
| ALL FEATURES<br>SVM 10-fold CV           | 98.8        | 99.4        | 98.2     | 0.071 | 1.7  |

We have conducted some experiments on the Fette corpus to compare classifiers in our prototype with their published results. We arrive at an F-value of 98.8% (row 2 of the table). This means a reduction of the error from 3.1% to 1.2% which is a reduction for more than 61%. Note that our system has not access to all features of the PILFER system, e.g. the age of linked domains.

## 6 Conclusions and future work

The result described in the preceding section shows that the AntiPhish Learning Platform (APL) fulfils the requirements specified in deliverable D2.1. However for the next phase of the project many things remain to be done. First the efficiency of APL for large corpora and email streams has to be improved.

With respect to scientific research we will investigate the differentiation of Branded from Non-Branded ham, and combine semi-supervised learning with active learning. Finally much effort will be devoted to the adaptive detection of new Phishing patterns.

## 7 References

- [Bengio Grandvalet 04] Yoshua Bengio and Yves Grandvalet. No Unbiased Estimator of the Variance of K-Fold Cross-Validation. *Journal of Machine Learning Research* 5 (2004) 1089–1105.
- [Bouckaert Frank 2004] Remco R. Bouckaert and Eibe Frank: Evaluating the Replicability of Significance Tests for Comparing Learning Algorithms. *Advances in Knowledge Discovery and Data Mining*. Springer. P.3-12. 2004.
- [Cavnar Trenkle 1994] Cavnar, W. B. and J. M. Trenkle, N-Gram-Based Text Categorization'. In *Proceedings of Third Annual Symposium on Document Analysis and Information Retrieval*, Las Vegas, NV, UNLV Publications/Reprographics, pp. 161-175, 11-13 April 1994.
- [Fette et al. 2007] Ian Fette, Norman Sadeh, Anthony Tomasic, "Learning to Detect Phishing Emails," in *Proceedings of the 16th International World Wide Web Conference (WWW)*, 2007
- [Levenshtein 1966] V. I. Levenshtein, Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady* 10 (1966):707--710.
- [McCallum 2002] McCallum, Andrew Kachites. "MALLET: A Machine Learning for Language Toolkit." <http://mallet.cs.umass.edu> 2002.
- [Mierswa et al. 2006] Mierswa, Ingo and Wurst, Michael and Klinkenberg, Ralf and Scholz, Martin and Euler, Timm: *YALE: Rapid Prototyping for Complex Data Mining Tasks*, in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-06)*, 2006.
- [Witten Frank 2005] Ian H. Witten; Eibe Frank (2005). *Data Mining: Practical machine learning tools and techniques*, 2nd Edition. Morgan Kaufmann, San Francisco.