INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Master's Thesis

Detection of unsolicited mails based on similarity in structural traits

Lukas Grob

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Master's Thesis

Detection of unsolicited mails based on similarity in structural traits

Lukas Grob

Aufgabensteller:	PD Dr. Vitalian Danciu
Betreuer:	Jan Schmidt Steffen Ullrich (genua GmbH)
Abgabetermin:	20. Dezember 2018

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 20. Dezember 2018

. . . . . .
(Unterschrift des Kandidaten)

Abstract

Email is one of the easiest and most popular means for attackers to distribute malware or to lure unsuspecting victims into giving away sensitive information. Even when the attacks are unsuccessful and no harm is done, these mails, together with common spam, are a major annoyance for both, business and private users. Common countermeasures to these threats include spam filters and anti-virus software, both of which heavily rely on traits found in the mails' contents or attachments. Attackers can evade spam filters by carefully designing their mails in an inconspicuous way, and anti-virus software often does not protect against new samples of malware. However, attackers often reuse their templates and tools to generate these mails, resulting in structurally similar mails compared to prior attacks.
This thesis therefore explores a way to detect unsolicited mails regardless of their contents, but solely based on similarity in the mails' structural traits, and evaluates which machine learning classifiers are suitable for this task. Since this requires labeled training data, this thesis also proposes a solution how the necessary training data can be collected and labeled automatically.

Kurzfassung

E-Mail ist für Angreifer einer der einfachsten und beliebtesten Wege Schadsoftware zu verteilen oder Opfer dazu verleiten vertrauliche Informationen preiszugeben. Selbst in Fällen in denen diese Angriffe erfolglos bleiben und keinen Schaden anrichten, sind diese Mails ein großer Störfaktor für Endanwender und Unternehmen. Gewöhnliche Abwehrmaßnahmen umfassen Spamfilter und Antivirensoftware, die beide stark auf Merkmale vertrauen, die im Inhalt oder Anhang der E-Mails zu finden sind. Spamfilter können leicht umgangen werden indem Angreifer ihre Mails unauffällig gestalten und Antivirensoftware bietet oft keinen verlässlichen Schutz gegen bisher unbekannte Schadsoftware. Beim Erstellen ihrer Mails bedienen sich Angreifer aber häufig der gleichen Programme und Vorlagen, wodurch Mails neuer Angriffe häufig eine ähnliche Struktur zu vorhergegangenen Angriffen aufweisen.
Diese Arbeit untersucht deshalb Wege wie sich die Erkennung unerwünschter E-Mails anhand von Strukturmerkmalen verbessern lässt, ohne auf Inhaltsmerkmale zu vertrauen, und untersucht welche Klassifikatoren aus dem Bereich des maschinellen Lernens dafür geeignet sind. Da ein solcher Klassifikator vorgelabelte Trainingsdaten benötigt, wird in dieser Arbeit außerdem ein System vorgeschlagen, das die benötigten Daten automatisch sammeln und labeln kann.

Contents

1 Introduction

According to Symantec's Internet Threat Security Report of April 2017, a substantial amount of malware is distributed by email and the danger of unsolicited mail in general was described as "one of the prime sources of disruption for end users and organizations" [Sym17, p. 24]. While those malicious mails were often easily identifiable in the past, attackers improved their methods to craft malware-carrying mails, making it harder to detect them. Especially in business environments where mail communication is part of many users' work flow, it is impossible to train all employees to identify every malicious mail. Therefore it is of the utmost importance to minimize the amount of malicious mails reaching the user.

Networks of almost all larger organizations are protected by firewalls securing incoming and outgoing connections and mails. These firewalls combine several techniques to reduce the amount of malicious mails delivered to the user. A spam filter tries to eliminate unsolicited bulk mail, but might not succeed against today's sophisticated attacks since the mails are specifically designed to look like regular, expected mails. Methods like DKIM, SPF, and DMARC are used to detect and prevent spoofing, but since their adoption lags behind they do not offer absolute protection against mail spoofing. Lastly anti-virus software tries to detect malicious attachments. While this provides solid protection against known malware, it can take several hours to days until vendors update their signatures. This leaves an exploitable window where anti-virus software does not protect against new malware.

In the recent years there were several new academic approaches to mitigate the problem of malicious mails using machine learning. Duman et al.[DKCE+16] proposed using stylometric features of mails to detect spearphishing attacks. Ali et al.[AON+15] developed a crawler to determine if mails have URLs referencing to malicious content and therefore are malicious themselves. Alqatawna et al.[AFJ+15] and Gascon et al.[GUSR18] proposed a sets of features to detect spear phishing.

While applying machine learning to these problems worked well, these proposals aim specifically at protecting against spear phishing, increasing the detection rates of spam filters, or rely only on features found in the mail's text or subject. This leaves demand for an approach that makes use of non-content-based features, but is designed as a general purpose detection method for all kinds of unsolicited mail.

1.1 Problem statement

Hence, this thesis will examine whether the features proposed by Gascon et al.[GUSR18] can be used to detect unsolicited mail in general. Specifically, it will answer the question which subset of the proposed features is best suited for this task.

Since these features need a classification system to be used in, the question which of several well-established classification algorithms is best suited for this approach will be answered. The experiments conducted will also give information about important questions when applying machine learning algorithms, namely how often retraining has to be performed to sustain the classification system's performance, and how much training data is required.

Since training data needs to be pre-classified, it is usually difficult to acquire a sufficient amount of training data for real-life applications. Therefore, this thesis will examine if and how the necessary training data can be automatically collected.

1.2 Concept

The proposed solution is quite simple. A machine learning classifier will be trained on a prelabeled dataset containing benign and malicious mails. This classifier can be applied to determine the malignity of unclassified, previously unknown mails. This is a promising approach because experience has shown that a significant amount of malicious mails share structural similarities, even when their contents are different. This classifier could be deployed at a central position of an organization's network with the ability to filter all incoming mail and protect the users.

The features used by this classifier can be divided into three categories:

- **behavior features** reflecting users' individual preferences when writing mails (e.g. usage of digital signatures, number of attachments)
- **composition features** derived from mail user agent specific traits in the structure of an mail (e.g. used encodings, mail headers)
- **transport features** extracted from transport related information in the mail (e.g. `Received` headers, DKIM records)

Since this classification system is intended as an addition to existing techniques where false-positives would severely reduce acceptance of the new methods, it is important to maintain a low false-positive rate while improving the detection of malicious mails.

The proposed features are highly dependent on the users' behavior, preferences, and their usual communication partners, making it unlikely that a existing training dataset from one network, can be used to efficiently detect unsolicited mail in another network. Therefore, new training data has to be collected for every network that should be protected by the proposed classification system.

This collection task can be done by the proposed automatic labeling system. It would be deployed at the network's mail server where it can see all incoming and outgoing mails. For every mail that can be unambiguously identified as either benign or malign with predefined rules, it will add the mail to the training dataset.

The concept for the collection of training data rely on the assumption that only a small fraction of the incoming mails need to be labeled to collect enough training data after some time. It should be sufficient if this process is finished within a few days to weeks.

The rules can partly be based on the results of existing methods, like anti-virus software, or real-time blacklists, assuming these methods have a low false-positive rate. Examples for such rules could be:

- Label as **benign**, if incoming mail is a response to a previously sent mail.
- Label as **malign**, if anti-virus detects malicious content in incoming mail.
- Label as **malign**, if the sender's IP address is found on a real-time blacklist.

If none of the rules apply, an incoming mail is passed on to the proposed classifier, which will then make a final decision about the mails maliciousness.

1.3 Outline

The following will provide a short overview over this thesis' structure. After the preceding problem statement and a short explanation of the proposed solution, Chapter 2 gives an overview over work related to this proposition. Section 2.1 describes other approaches to detect unsolicited mail. This shows that existing methods are not sufficient to successfully protect users from the danger of malicious mails and the nuisance by spam. Section 2.2 gives an introduction into the topic of machine learning, describing common classification algorithms and evaluation techniques.

The previously outlined concept for this thesis is explained in a more detailed way in Chapter 3, where based on the research in Chapter 2, the used classification algorithms are determined: a Support Vector Machine, a Naive Bayes classifier, a Random Forest, k-Nearest Neighbors, and Logistic Regression.

Further, an in-depth description of the used features is given. They are divided into three categories, which form the three independent featuresets *mail features*, *mail user agent features*, and *transport features*.

This proposed solution will then be evaluated in Chapter 4. First, in Section 4.1, the setup for the evaluation experiments is shown. This includes a description of the used dataset, the steps necessary for feature extraction and data preprocessing, and a description of the Python scripts used for the experiments and for analyzing the results.

The evaluation approach is shown in Figure 1.1. As it can be seen, this is split into two parts. The first part consists of evaluating the featuresets and selecting the most suitable one for each classifier, and then examining which of the five classifiers performs best for this problem.

The three different featuresets are evaluated in Section 4.2 by performing experiments with various amounts of training data with each classification algorithm and featureset. These experiments reveal that generally all proposed featuresets yield good results. However, some algorithms show slightly improved performance when using only a smaller subset of features.

Then, in Section 4.3, the five chosen classification algorithms are evaluated. Several experiments are conducted to examine the classifiers' performance in situations where only little training data is available, where a lot of training data is available, and situations where only outdated training data is available. These experiments show that for this task, the Support Vector Machine is the best suited of the five classification algorithms, and the classifier's performance does not significantly increase when more than 2000 samples, equivalent to about two weeks of mails, are used for training. Overall, these experiments show that the proposed features and some of the selected algorithms can identify unsolicited mail with a recall of up to 98 %, and a false-positive rate below 4 %.

So far, the experiments only examined the case where a extensive pre-labeled dataset is available for training. The second large part shown in Figure 1.1 considers situations where the training dataset has to be collected or generated first. An approach to this problem is discussed and evaluated in Chapter 5. After describing the general concept of the proposed automatic labeling solution in Section 5.1, Section 5.2 discusses different possible information sources for the automatic labeling. Section 5.3 evaluates the combination of the previously examined classifier, trained with data collected from the automatic labeling systems. These experiments show that the automatic labeling works, and the combined system can provide a benefit to the currently common detection methods. However, it also shows, that the overall false-positive rate is still too high for deployment in real-life systems, and proposes several

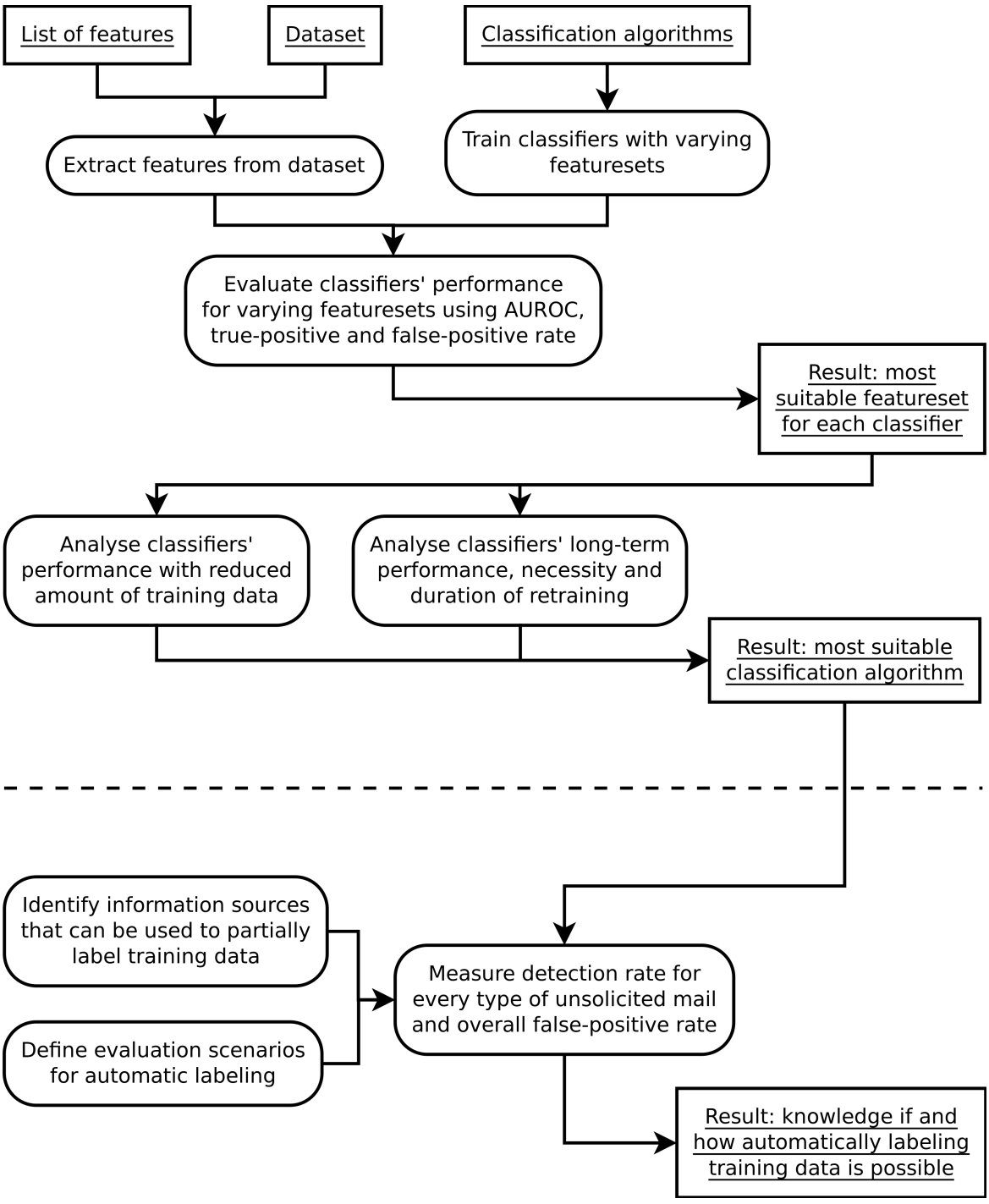


Figure 1.1: Proposed evaluation approach

ideas how the false-positive rate can either be reduced, or its impact mitigated.

Chapter 6 concludes this thesis by first recapitulating the results from previous chapters, showing that while the approach is promising, more work is needed to develop the proposed solution into a system that can be deployed in real-life situations.

Especially finding a better trade-off between high recall and low false-positive rate is made out as one important topic for future research, and propositions how this can be improved are made.

2 Related Work

This chapter gives an overview over work related and relevant to this thesis. Section 2.1 focuses on methods to detect unsolicited and malicious mails that have been used in the past, are currently deployed, or are currently under research and might be used in the future. Section 2.2 introduces machine learning techniques that will be used later in this thesis.

2.1 Detection of unsolicited mails

Countermeasures to spam is a thoroughly explored topic. A general overview over deployed techniques can be found in [SJJ16]. Guzella and Caminhas reviewed the efforts to fight spam with methods of machine learning in 2009 [GC09].

One of these approaches, DomainKeys Identified Mail (DKIM), was described in 2007 by Barry Leiba and Jim Fenton in [LF07] and aims at preventing spoofing. However, spoofing is only a small part of the problem, and its prevention cannot keep attackers away reliably. Furthermore, the effectiveness of DKIM and other provider based mechanisms was examined by Foster et al. in 2015 [FLM+15], showing that their adoption and enforcement lacks behind, leaving their security gain inadequate.

One of the first approaches already featured probabilistic methods from the field of machine learning, when the increased amount of unsolicited emails lead to a demand in automated spam detection and elimination techniques: Sahami et al. proposed the use of Bayesian networks to detect unsolicited junk mail in 1998 [SDHH98]. These filters were very effective at the time and are still in use today.

Comparable attempts using neural networks have been undertaken at the University of Sydney in 2003 [CKP03], and Pace University in 2004 [SCT04], showing the suitability of neural networks for spam detection.

The effectiveness of another content-based approach, the use locality-sensitive hash algorithms in fighting spam, has been demonstrated by Damiani et al. [DDPS04].

Researchers at the Carnegie Mellon University showed that content-based features and machine learning can be used to detect phishing mails, a special case of spam [FST07].

Alqatawna et al. examined whether the use of features found especially in malware-carrying mails can improve the detection accuracy compared to the use of traditional features [AFJ+15].

However, these studies were mostly limited to content-based features. Therefore these approaches will not be regarded any further. While content-based detection works well as protection against large parts of unsolicited mails, circumventing these filtering techniques requires little additional effort by attackers, because they can arbitrarily alter the contents of their spam mails.

This problem was also recognized by researchers who started looking for other, harder to forge features and more sophisticated or specialized ways to detect unsolicited mails.

One example of an unusual attempt at fighting targeted malicious mails was undertaken by Duman et al. in 2016: they used stylometric analysis and mail headers to detect spearphishing [DKCE+16]. While they have shown that this approach works against targeted attacks, the paper considers only perfectly spoofed mails. Since these are only a small part of unsolicited mails, it is unclear if this approach is suited for general purpose detection of unsolicited mail and will not be regarded further.

Martin et. al from the University of Berkeley proposed to use behavioral features to distinguish regular outgoing mail from worm-propagating mails. Behavioral features reflect users' preferences like the number of attachments, their MIME type, or the number of unique sender addresses [MNS+05]. While this approach is not directly aimed at fighting incoming spam, it shows that behavioral features heavily differ between regular and malicious mails, making them possibly useful at fighting unsolicited mail as well. The application of behavioral features will therefore be part of the thesis.

In 2011, Rohan M. Amin evaluated a new featureset to detect targeted malicious mails. This featureset contained recipient oriented features like the receiver's role in the targeted organization, and features that have been found to be connected to persistent threats like the used encoding and the `X-Mailer` Header. The evaluation has shown that the examined featureset is suited to detect targeted malicious mails [Ami10][ARD12] and will in parts be examined in this thesis.

Based on these results, Gascon et al. developed a method to detect spear-phishing mails by creating sender-specific profiles and detecting if a mail does not match the alleged sender's profile [GUSR18]. Their proposed features appear to be very suitable for detection of unsolicited mails that is not based on the mails' content and will therefore be used in this thesis.

These recent approaches already show a trend in anti-spam measures: the application of machine learning using various proposed featuresets.

2.2 Machine learning

Since machine learning became a common tool in the detection of unsolicited mail and will be used in this thesis as well, the following section will give a short introduction to relevant algorithms, terminology, and evaluation methods.

One common task of machine learning is classification. The goal of performing classification is to let a algorithm identify to which class a sample belongs. These classes have to be pre-defined, and the classifier has to be trained first, which requires training samples with known class labels.

2.2.1 Classification algorithms

k-Nearest Neighbors

Classification is one of the oldest tasks in machine learning. One of the easiest algorithms, the k-Nearest Neighbor classifier, reaches back to at least 1962, when Sebestyen described it as the "proximity algorithm" [Seb62]. It is an example of so-called *lazy-learning*, where no calculations are done during training, but only when classifying samples.

When determining the label of a new sample, the classifier compares it with all training data and looks for the k training samples that are closest. The label that is the most

common among these neighbors will be assigned to the new sample. The parameter k is an integer, usually in the range between one and ten. A higher k, leads to a classifier that is less susceptible for noise or outliers, however if k becomes to high, it becomes increasingly difficult to find a clear distinction between classes. [Mit97]

Common metrics for the distance between samples are the Manhattan distance and euclidean distance.

The fact that for every sample that has to be classified, the distance to every single training sample has to be calculated, can make KNN slow for a large amounts of training data.

As the simplest concept its performance for this thesis' problem will be examined.

Support Vector Machines

Another common classification algorithm is the Support Vector Machine. The concept has been shown in [BGV92] and [CV95].

Contrary to the k-Nearest Neighbors algorithm, Support Vector Machines (short *SVM*) perform a large part of the necessary calculations during training. They view every sample as a vector in a multidimensional space and try to find a plane that separates the classes in a way that the margin between the plane and the vector is a large as possible.

The vectors closest to the planes, determining the exact discrimination boundary, are called support vectors. When classifying a new sample, only its location in relation to the discrimination boundary has to be calculated to determine its class label [Mur12]. A larger distance of a classified sample from the discrimination boundary correlates with a higher confidence of the classifier's decision.

This concept works best if the classes are linearly separable. If not, the algorithm tries to find a good trade-off between a wide margin separating classes, and as few as possible misclassifications. This trade-off can be influenced with variables of the error function that have to be optimized for every application.

A single Support Vector Machines only allows for separation of two classes. One of the available schemes to solve multiclass problems is the one-vs-rest strategy which works by creating a classifier for every single class that determines whether a new samples is in that class or any other class (*rest*). The sample will then be assigned to the class which has the highest confidence.

Since they have already been used for text classification, and Brutlag and Meek have shown their suitability for email classification [BM00], Support Vector Machines appear to be suitable for the proposed task, and will therefore be considered in this thesis.

Logistic Regression

The general principle of Logistic Regression has been first described by David Cox in 1958 [Cox58]. It can be used as a classification model.

Similar to Linear Regression, it uses a linear combination of all input variables (*features*) with their respective weights, called the *hypothesis* [RN09].

$$h(x) = w_0 + \sum_{i=1} w_i x_i$$

The output of this hypothesis is then passed into the logistic function.

$$Logistic(x) = \frac{1}{1 + e^{-h(x)}}$$

The final result indicates the probability of the sample x having a positive class label in a binary classification problem.

Training a Logistic Regression classifier means finding the optimal weights given the training data. For this task, common methods like coordinate descent [Mur12] or gradient descent can be used. Both are iterative approaches to gradually reduce the error function, and as a result approximate the optimal weights. [RN09]

Decision Trees and Random Forests

The general concept of decision tree learning was introduced by Breiman et al.[BFOS84].

Using training data, the learning algorithm creates a binary decision tree, that can be used for classification. Starting from a root node, the algorithm determines which feature gives the most information. Several metrics exist to determine which feature is the best suited, for example the entropy as described by Shannon [Sha48]. Once the best feature is assigned as a leaf to the tree, and the data is split using the feature. For each part of the data, this process is repeated. New leafs get assigned to the tree and the data is split, unless only one class is left in the data split. The class label of the remaining samples will then be assigned as a final leaf in the tree. However, if there are no more features left and there are still different labels in the remaining samples, the most probable label will be assigned. [Mit97]

Based on the work of Ho [Ho95][Ho98], Breiman proposed to combine multiple decision trees as *Random Forests* [Bre01], where each tree uses a different subset of input features and training samples.

Oshiro et al. examined how many trees should be used in Random Forests [OPB12]. They determined the classification performance of Random Forests with different numbers of trees for several real-life classification problems from the biomedical domain. They concluded that performance did not significantly increase when using more than 128 trees.

Since Random Forests tend to be less prone to overfitting compared to single decision trees and their construction as well as the classification can be parallelized easily [CF18], they will be used as a classification algorithm in this thesis.

Naive Bayes

Naive Bayes classifier calculate the probability of a sample belonging to a certain class, given the labeled training data. They assume that every feature is conditionally independent of every other feature given the sample's class label. Even though this assumption is usually false, Naive Bayes classifiers have been used with great success for various different classification problems. [RN09][Mur12]

The probability of a sample x belonging to a class c is as follows:

$$P(C = c|x_1, ..., x_n)$$

Where $x_1, ...x_n$ are the separate features of the sample x.

While this cannot be computed directly, using the assumption of conditional independence and the Bayes' theorem, it is possible to simplify the formula:

$$P(C = c|x_1, ..., x_n) = \frac{1}{P(X)} P(C = c) \prod_{i=1}^{n} P(x_i|C = c)$$

This probability is calculated for each class and the one with the highest probability is selected as the predicted class. The prefix $\frac{1}{P(X)}$ is independent of the class, and therefore does not have to be calculated. [Mur12]

Since Naive Bayes classifiers have been used with great success in spam filtering [SDHH98], their application will also be examined in this thesis.

2.2.2 Evaluation

When creating a machine learning system it is useful to predict how well it will perform. There is a large number of different evaluation metrics which share the goal of representing a classifier's performance as well as possible.

Comparing machine learning classifiers itself is a topic that has been studied before. Salzberg's recommended approach [Sal97] describes a way to reliably measure the performance of a new classifier. Since this approach was intended to measure the benefits of new classifiers compared to existing ones, not all the described steps might be applicable to this thesis. However his general procedure can be adopted for this thesis to make sure the classifiers' performance will be measured in a credible way.

However, this describes only a procedure. Another important part in the evaluation are different performance metrics.

Confusion matrix

As shown in [Pow11], a lot of the metrics can be derived from a basic confusion matrix. The confusion matrix presents an easily recognizable way to display a classifier's prediction results in contrast to the actual classes.

	True class		
	Hotdog	not Hotdog	
Hotdog	80	10	90
not Hotdog	5	5	10
	85	15	100

Table 2.1: Example of a confusion matrix

Table 2.1 shows the confusion matrix of an hypothetical classifier whose purpose it is to determine whether a food item is a hotdog or not. In this case a hotdog can be considered a positive sample, others a negative samples. 85 of classified items are hotdogs (*positives*), 15 are not (*negatives*). However, the classifier predicted 90 are hotdogs (*predicted positives*) and ten are not (*predicted negatives*).

Of the 85 true hotdogs, 80 were detected (*true positives*). Ten items were predicted to be hotdogs while they are not (*false positives*). Five were not recognized as hotdogs while they should have been (*false negatives*). And another five have been correctly classified as being no hotdogs (true negatives).

Using this matrix, a lot of different values indicating performance can be calculated. The most obvious would be the general classification *accuracy*. In this case it is 0.85.

$$Accuracy = \frac{true\ positives + true\ negatives}{overall\ number\ of\ samples} = \frac{80 + 5}{100} = 0.85$$

However, the accuracy is a bad metric as it can be deceiving about a classifier's performance in situations where the classes are unevenly common. If the classifier would always predict all items as hotdogs without any discrimination, it would be a very bad classifier. But it's accuracy score of then 0.8 would still suggest reasonable classification performance, while the classifier is useless in practice.

Another metric is the *precision*. It states how many of the positively predicted samples were actually positive. In our example it's 0.8889.

$$Precision = \frac{true\ positives}{predicited\ positives} = \frac{80}{90} \approx 0.8889$$

Another metric is the *true-positive rate*, also called *recall* or *sensitivity*. In our example the true-positive rate is 0.9412. It states how many positive samples are actually detected by the classifier.

$$True\ positive\ rate = \frac{true\ positives}{number\ of\ positive\ samples} = \frac{80}{85} \approx 0.9412$$

The true-positive rate alone is only partially useful. It is often complemented by the *false-positive rate*, which basically tells how often a false alarm occurs.

$$False\ positive\ rate = \frac{false\ positives}{number\ of\ negative\ samples} = \frac{5}{15} \approx 0.3333$$

These metrics are usually not used alone, but in combination with another one. Common combinations are precision and recall, or true-positive and false-positive rate, depending on the researchers preferences or the condition under which the machine learning system has to operate [Pow11].

Receiver operating characteristics (ROC)

True-positive and false-positive rate can be used to show the trade off between detecting as many positive samples as possible while generating as few as possible false alarms. Most classifiers allow to move their decision threshold to adjust true-positive and false-positives rates according to the use case's requirements. A way to reflect this is the *Receiver operating characteristics curve (ROC curve)* as described in [Faw06].

An example is shown in Figure 2.1. The curve shows which true-positive rate can be achieved, given a specific false-positive rate. It is generated by slowly lowering the decision threshold, starting at a point where every sample is classified as negative and both rates are 0.0. While lowering the threshold, more and more samples get classified as positive, resulting in a higher true-positive and false-positive rates. Finally the threshold is low enough to classify every sample as positive, leading to both rates being 1.0.

The worst possible classifier has a ROC curve identical to the diagonal line between the points (0.0, 0.0) and (1.0, 1.0). It is also the curve a random classifier would achieve.

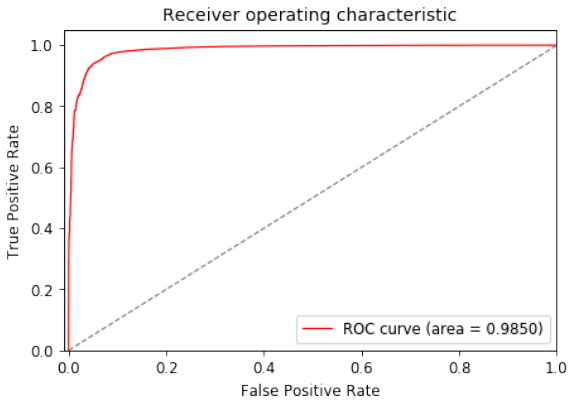


Figure 2.1: An example of a ROC curve

This curve can be used to compare the possible true-positive vs. false-positive trade-offs of different classifiers.

The technique works best when using classifiers that give continuous real numbers as a confidence value of their decision, like Support Vector Machines or Logistic Regression. Classifiers like k-Nearest Neighbors or Random Forests with a small k or a small number of trees, have limited possible confidence values. k-Nearest Neighbors with $k = 3$ for example only returns confidence values of 0, $\frac{1}{3}$, $\frac{2}{3}$, and 1. This reduces the expressiveness of the ROC curve in these situations.

However, comparing multiple ROC curves can quickly become confusing, especially since it might not be obvious which curve is better. It is possible that a curve shows superior results in one part, but worse in another part. Therefore, basic ROC curves will not be used for evaluation in this thesis.

Area under ROC curve (AUROC/AUC)

One way to solve the issue of comparison is the *area under the ROC curve* (*AUC* or *AUROC*). A random guessing classifier with an ROC curve equivalent to the diagonal line has a AUROC of 0.5.

While the AUROC does not necessarily reflect the optimal true-positive and false-positive rate, it allows for a good estimation about the flexibility when finding a trade-off that is suitable for the given problem. Therefore AUROC is generally considered to be a good metric for classifiers' performance. [Faw06]

Since the AUROC is easily comparable and finding a classifier that allows for a good true-positive vs. false-positive trade-off will be an important task in this thesis, AUROC analysis will be one of the evaluation tools used.

Concept drift

Another topic that has to be taken into account is concept drift, which has been described by Tsymbal in [Tsy04]. The underlying issue is that real world data and data distributions change over time. The models learned from past data do not fit to new data anymore, resulting in a decline of classification performance as time passes.

A way to detect drift has been proposed by Gama et al.[GMCR04]. Since the methods used by attackers to create unsolicited mails change over time, concept drift might be an issue for this thesis and has to be considered.

The insights gained by examining related work from the domains machine learning and detection of solicited mails are used in Chapter 3 to formulate the approach and outline the experiments to assess the efficiency of the proposed solution.

3 Analysis and approach

After having examined research related to the detection of unsolicited mail and the basics of machine learning in Chapter 2, this chapter will analyse the different types of unsolicited mail in Section 3.1, and give an in-depth explanation of the proposed solution to automatically detect unsolicited mails in Section 3.2. Section 3.3 describes the approach to assess the effectiveness of the proposed detection system, followed by a detailed description of the used features in Section 3.4 and the requirements which the dataset used for the experiments has to fulfill in Section 3.5.

3.1 Types of unsolicited mail

Unsolicited mail can generally be divided into three different categories:

1. **Spam**
   Spam describes unsolicited mail, sent in large numbers to a large number of recipients. While spam is usually considered to be only a nuisance, it can have an impact on the productivity of employees. When they receive a spam mail that has not been filtered out, they have to classify it as spam themselves and delete it. As long as this is does not happen too often, it is not a serious problem. But if the amount of spam reaching the user gets out of hand, it can keep employees from fulfilling their work. However, spam filters usually work good enough to keep a significant amount of spam away from users. [GC09]

2. **Phishing**
   A more direct threat to the success of organizations and companies is phishing. Its goal in general is to obtain sensitive information, by relying on the user to give out the wanted information. To achieve this, the attacker tries to gain the user's trust by pretending to be a trustworthy, known person or entity. The user is often directed to a website that imitates the look of a trustworthy website, for example online banking pages, with the request to enter personal information like credit card numbers or login credentials. While most phishing attempts are easy to recognize, unwary users can fall for the attackers' deceptions. More sophisticated phishing attempts are becoming more common, are harder to detect, and can do severe financial harm to victims [Sym17].

   Despite the risks of phishing for both, end-users and organizations, and the successful research efforts shown in Chapter 2, practically applied methods against phishing are lacking. Countermeasures are usually limited to spam filters and anti-spoofing techniques. Spam filters may be easily bypassed by crafting a phishing mail that looks distinctively like a regular, benign mail, and additionally to the lacking effectiveness of anti-spoofing approaches [FLM+15], spoofing isn't even necessary for a successful phishing attack.

3. **Malware**
   The dangers of malware spread by mail became obvious in 2016, when ransomware became a popular source of income for criminals. Ransomware encrypts the victim's data and demands a payment to be made (*ransom*), usually in the form of some crypto currency. Only after this payment has been received, the victim may or may not receive a key to decrypt the data. To spread ransomware, attackers send the malware to victims, urging them to open the malicious attachment. If the attacker put enough effort into the mail and the request to open the attachment seems legitimate, it is difficult for end-users to detect such an attack beforehand, and technical precautions are usually limited to spam filters and anti-virus software.

   However, attackers can test their malware against common anti-virus software and design it specifically to remain undetected. This can be done by creating permutations of the same malware. To detect such a new and undetected malware sample, anti-virus software manufacturer have to update their detection signatures and heuristics, and distribute the updated information to the users. This leads to an exploitable time-span of hours to days, after the first occurrence of new malware. Once the new signatures have spread, the attackers slightly modify their malware, so that it can't be detected by the existing signatures, making signature-based anti-virus software ineffective again.

All three issues have one thing in common: the countermeasures rely mostly on information found in either the mail's content or attachment. Meanwhile, the structure of the mails remains largely the same, even between attacks. Attackers usually use tools to generate their mails. Whenever they start a new attack, they change the message body and the attachment, but still use the same tools and templates as in previous attacks. These introduce distinctive peculiarities in the mails' structure that remain unchanged.

Such traits describing the mails' structure might be deviations from standards (RFCs), or odd interpretations of standards. This thesis explores if such peculiarities can be used to improve the detection of unsolicited mails. However, even common mail user agents used by regular users and victims have such distinctive features. Therefore, the goal is not to detect behavior that goes against standards, but to detect similarity in these structural traits to known benign or malign mails.

The features proposed in [GUSR18] have specifically selected to represent the structural traits in mails. They can be divided into three featuresets:

- Behavior features, similar to the ones described in [MNS+05]. They reflect the sender's behavior and preferences when writing mails and mostly contain information about the content, the sender, and the receiver.
- Composition features that are dependent on the sender's mail user agent (MUA).
- Features based on transport related information of the mails.

More detailed descriptions of these features can be found in Section 3.4.

3.2 Automated detection of unsolicited mail

A machine learning classifier has been used to inspect these traits (Section 4), and if a high similarity to known unsolicited mails can be detected, the mail is marked as unsolicited.

Such a classifier could be placed at a networks perimeter firewall or mail server, where it could automatically filter out mails before they reach the end-users.

However, it would not replace existing methods to detect unsolicited mail. These methods have been optimized for years, and provide a valid benefit. Instead, the proposed classifier would form an additional step when detecting malicious mails, introducing structural analysis.

Designing the classifier as an additional step leads to a special requirement: the classifier must not introduce a significant number of false alarms. A high false-positive rate of the classifier would reduce acceptance of the proposed solution, since the additional step would be considered more of a nuisance than a security benefit. This will have to be considered during the evaluation process of the proposed solution.

According to [GC09], several machine learning classifiers have been used for spam detection in the past. Based on this, the following classification algorithms have been selected for further evaluation in this thesis:

- k-Nearest Neighbor
- Support Vector Machine
- Logistic Regression
- Random Forest
- Naive Bayes

3.3 Approach

First, the different featuresets are evaluated in Section 4.2, by training all classifiers with every possible featureset combination with various amounts of training data. By using the AUROC metric, the most suitable featureset for every classifier is determined.

This information will then be used in Section 4.3 to assess the classification algorithms. Special attention is payed to three main issues:

- **Required amount of training data**
  While a large pre-labeled dataset is available for this thesis, real-life applications might have to work with significantly less training data. Therefore the classifiers' performance with reduced amounts of available data has to be considered.

- **Long-term stability**
  The classifiers' adaptability to concept drift will be examined. Spam mails have most likely changed in the course of the last few years and the used dataset should show this. Generally, ways to detect concept drift have been described by Gama et al. [GMCR04] and Alexey Tsymbal [Tsy04].

  To examine the effect of concept drift on the proposed solution, the classifiers will be evaluated with training data that is older than the test data.

- **Computational complexity**
  If such a classification system is supposed to be employed in real-life, the computational complexity has to be examined. Therefore, the training and classification times will be measured for every classifier.

All these points will be examined in regards to the classifiers' true-positive rate (recall), false-positive rate (false alarm), as they describe directly the characteristics that are most noticeable in real life applications. To gain additional insight about the possible trade-offs between these two, the AUROC will be regarded as well.

Because there is probably no such dataset available in most cases, an automatic labeling system is needed. To determine the feasibility of such a system, possible information sources that could be used to automatically label training data will be identified, for example DNS blacklists, results of anti-virus software, or relations between `In-Reply-To` and `Message-Id` headers. A more detailed description of this concept and its evaluation can be found in Chapter 5.

3.4 Featuresets of the mail transport system

In [GUSR18], Gascon et al. proposed a set of features to detect spearphishing. These features have been specifically selected to describe a mails structure and therefore are ideally suited to evaluate the approach of detecting unsolicited mail without relying on content-derived traits. The proposed features can generally be divided into three categories:

1. Behavior features, similar to the ones described in [MNS+05]. They reflect the sender's behavior and preferences when writing mails and mostly contain information about the content, the sender, and the receiver.
2. Composition features that are dependent on the sender's mail user agent (MUA).
3. Features based on transport related information of the mails.

3.4.1 Message features

This featureset is also referred to as the *mail* featureset. It is made of features describing the structure of a message itself. This includes traits from the message's body as well as certain headers. It's goal is to represent the sender's behavior when writing mails and to describe the kind of mails a specific recipient usually receives. Similarities to previous malicious mails can indicate that a newly received mail is malicious as well. The following lists the traits found in the *mail* featureset:

- Usage of HTML tags
  Describes the occurance and combinations of HTML tags in a mail's `text/html` part.
- Attachment type
  Simplified file types of attachments.
- Domains in links
  Describes the relation of HTTP or HTTPS link to the sender's or recipient's domain. Phishing mails for example often have links to websites that are not-related to the supposed sender's address.
- Mail addresses
  These features represent the `To`, `Cc`, `From`, and `Reply-to` header fields. They note the number of occurrences of these headers fields, the number of elements per field, and the anonymized addresses found in them.

- MIME parts
  Includes features describing number of MIME levels, the path to each MIME part, and their sizes and types.

- Reply or forwarding
  Features noting if the mail's subject suggests the mail to be a forwarded message or a reply to a previous message, for example "Re: next meeting". Also included are anonymized addresses found in the `Reply-to` and `Return-path` headers. These features can help identifying spam. For example having a subject suggesting a mail to be a reply without having a `Reply-to` header is usually only found in spam mails.

  Another feature contains the amount of quoted text inside inline text or plain parts as a fraction of the total amount of text. This can also help to determine whether a mail is actually a response to a previous message.

3.4.2 Mail user-agent features

Relevant to the detection of malicious mails are traits that describe the mail user agent's behavior when composing the mail. The relevant RFCs often allow for different ways to achieve the same result. Examining the mail user agent's behavior can reveal peculiarities that hint at the usage of a particular software used to compose the mail and indicate a mail's potential maliciousness if this software has been used to compose malicious mails in the past.

The relevant mail user agent features can be classified into features describing the used content-transfer-encoding, the use of MIME parts and the way how attachments are included, and the type and value of the headers set by the user agent.

These features are usually specific to certain mail user agents and can be used to partially identify mail clients. Malign senders often use the same tools to generate mails, having their own specific behavior and should be distinguishable from common mail user agents using these features:

- Content-transfer-encoding
  Several features represent the mail's content-transfer-encoding, including 7 bit, 8 bit, base64, and quoted-printable encoding.

  They describe what kind of encoding has been declared, the declaration's capitalization, and deviations from the standards defined by RFCs. Possible deviations are the use of 8 bit characters in declared 7 bit encoding, or lines longer than 1000 characters.

  Non-standard base64 behavior is also noted, for example overlong lines with more than 76 character per line, differing line lengths, or whitespaces within base64 encoded data.

  Other features describe peculiarities of quoted-printable encoding, like multiple successive unencoded whitespaces, unencoded whitespaces at the end of a line, encoded characters that didn't need encoding, line lengths over 76 characters, or unencoded 8 bit characters within quoted-printable encoded data.

  These deviations are not uncommon and are often generated and accepted even by common mail user agents, and are therefore harmless most of the time. However, attackers can exploit differences in the interpretation of non-standard behavior to bypass anti-virus software or intrusion detection systems by deliberately using specific deviations.

- MIME parts and attachments
  Significant to the description of the mail user agent's behavior are features regarding the usage of MIME. These include the structure of the used boundaries, an anonymized digest of the MIME preamble, and MIME specific peculiarities regarding unusual declarations of `Content-disposition` and `Content-type`.

  Another important part is what kind of attachments are used and how they are attached. The features describe file extensions, fake file extensions (e.g. `file.doc.exe`), and how attachments have been declared. This includes whether the filename in `Content-disposition`, or the name in `Content-type`, or both was given, if it was given quoted or unquoted, as plain text, base64 encoded, or quoted-printable encoded.

  While the file extension of an attachment is more dependent on the user's behavior, it is required here to add more context to the way how the attachment is included, because different file types are usually attached in different ways by the mail user agents.

  The way how certain attachments occur in a mail is often typical for specific mail user agents.

- Parsing warnings and errors
  This includes warnings or errors with their impact on stringent interpretation when parsing mails, for example invalid characters within quoted-printable or base64 encoding, or missing boundary strings. Certain combinations of parsing warnings and errors are often characteristic for mails from a specific mail user agent.

- Other headers
  Included in the *mail user agent* featureset are features representing the headers found in a mail. These are all headers that are not part of the *mail* featureset and were not set during the mail's transport. Headers found in individual mail parts are included as well.

  The representation also includes the syntax how each header has been set, for example capitalization.

  Some headers are simplified, for example the exact `Message-Id` is not relevant to detect mail user agents. Therefore only the `Message-Id`'s structure is extracted. The user-agent extracted from the `X-Mailer` or `User-Agent` header is set to `other` for uncommon user-agents.

  The occurrence, combinations and syntax of certain headers can be used to identify specific user-agents.

3.4.3 Transport features

This featureset is made up of traits that describe the transport of mails and includes the following traits:

- DKIM and SPF
  Represent the existence and validity of a DKIM signature and the results of an SPF lookup.

- `Received` headers
  Features from the mail's received headers like the protocol, timezone and hostname of

the sender and each mail transfer agent. Additionally it documents the use of TLS, the TLS version and cipher, and if the client's certificate was verified. Other headers set during the mail's transport are also included, those are found above a mail's first `Received` header.

These features are suited to describe a mail's path on it's way from sender to recipient and the timezones correspond to the mail's geographical path. This can be used to identify malicious mails, as their path is for example often shorter, and different malware mails are often sent from the same botnet, resulting in similarities at the beginning of a mail's path.

- Source IP address
  Features derived from the source IP address, including the sender's ASN and the result of DNS RBL lookups of the sender's IP address. This is a useful feature since some source IP addresses or ASN are more than others involved in sending malicious mails.

3.5 Dataset

The experiments proposed in Section 3.3 require a dataset on which the experiments are conducted on. Based on the nature of these experiments, the following requirements have been determined:

1. **Good representation of real life data**
   The goal is to protect a whole network or organization from incoming malicious mails based on their structural similarity. The used classifiers would be employed at a central point within the network, for example at the mail server. This means, the classifier would have to deal with mails addressed at users within the same organization. Therefore, the dataset used for the experiments in this thesis must reflect this type of recipients and contain mails addressed at users in the same organization.

2. **Sufficient amount of data**
   To reliably perform classification, machine learning algorithms need a certain amount of data. How much data is needed exactly has to be determined during the experiments, but the used dataset has to be large enough to simulate various situations with more or less available training data.

3. **Long time span covered**
   Since unsolicited mail evolves over time, the conducted analyses have to examine how the classifiers' performance changes over time and how older classifiers perform with newer data. As these effects might only be visible after a time span of several months, the dataset has to cover a time span of at least a few years.

4. **Has to be pre-labeled**
   To train machine learning classifiers, the training data has to be pre-labeled. To evaluate their performance, the test data has to be labeled as well. These labels may either be binary, *malign* and *benign*, or more detailed and distinguishing between *spam*, *phising*, and *malware*.

5. **Must contain malign and benign mails**
   Since classifiers are supposed to learn to distinguish between malign and benign mails, the dataset must contain both classes in sufficient numbers.

6. **Must contain recent mails**
   While it is interesting to see how the classifiers would have behaved in the past, it is necessary to determine if they perform well in the present. The dataset therefore has to contain recent data.

The approach proposed in this chapter is implemented in Chapter 4, where the used dataset is evaluated in regards to the previously defined requirements, and the experiments as outlined in Section 3.3 are conducted.

4 Assessment of algorithms

In this chapter, the different featuresets and the selected classification algorithms will be evaluated. First, Section 4.1 will provide an overview over the experimental setup, the applied workflow, and the used software.

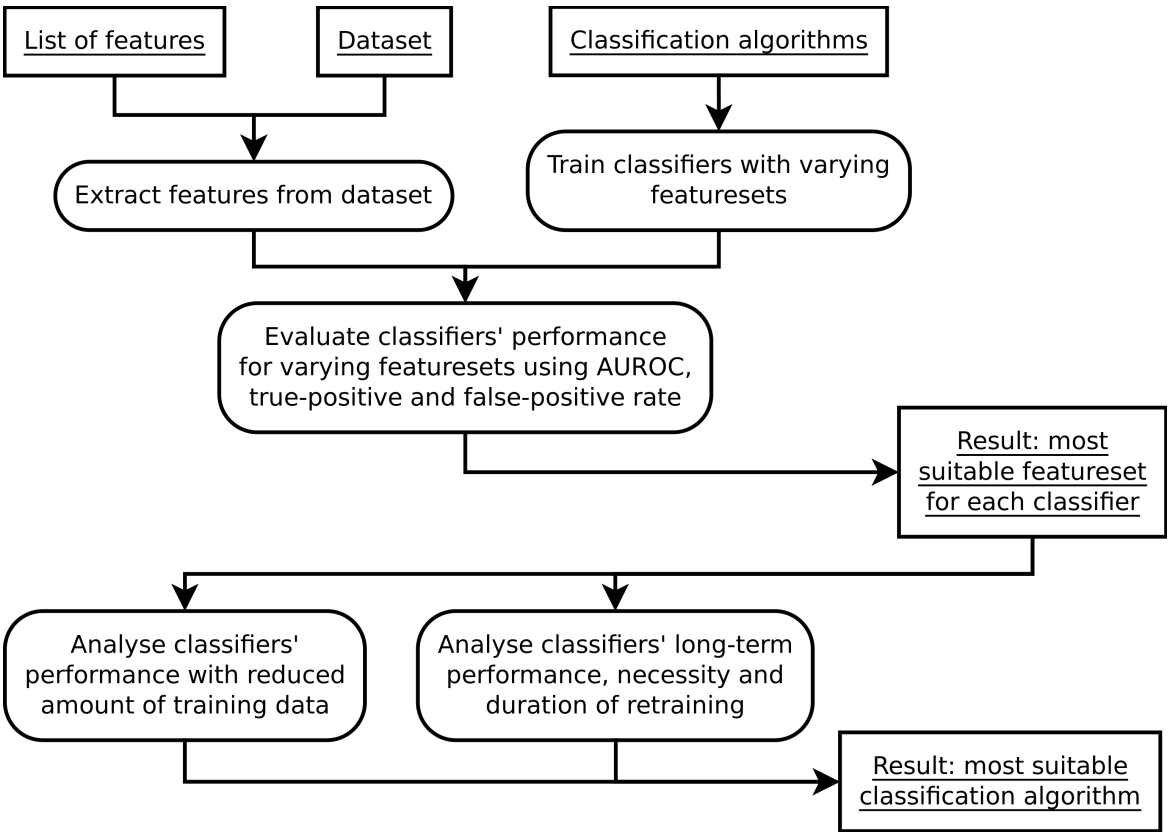


Figure 4.1: Assessment of featuresets and classifiers

The evaluation process is shown in Figure 4.1. First, the most suitable featureset for each classification algorithm is determined in Section 4.2. For this, all five classifiers are trained with different amounts of training data and all possible combinations of featuresets, and their classification performance in regards to true-positive rate, false-positive rate, and AUROC is observed.

After the optimal featureset of each classifier has been found, all classifiers are evaluated in regards to their performance with low amounts of training data, their resilience against concept drift, and the necessity and duration of retraining in Section 4.3.

Finally, Section 4.4 concludes this chapter by giving a final recommendation about which combination of classification algorithm and featureset is to be used to achieve the best possible results.

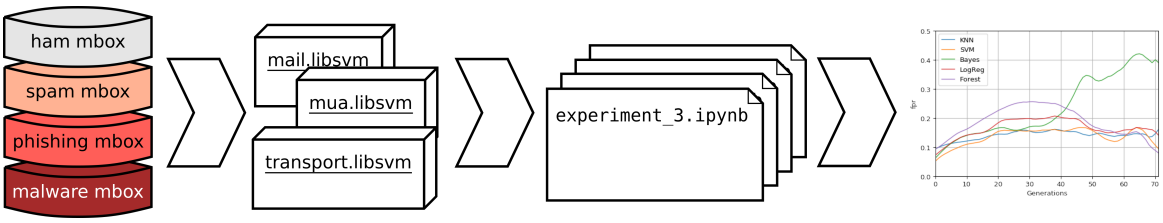


Figure 4.2: Experimental setup

4.1 Experimental setup

The general procedure for the evaluation experiments is shown in Figure 4.2. On the left side, the dataset is depicted. In Section 4.1.1 the used dataset will be presented and it will be discussed if the dataset fulfills the requirements defined in Section 3.5. Since the data is only available in the mbox format, it will be converted into the easier to process LIBSVM[1]/SVM light[2] format as explain in Section 4.1.2.

Once this transformation process is done, the data will be read with Jupyter notebooks. These Python scripts and the used helper library are described in Section 4.1.3. Training and prediction of the used classifiers is done by these script, and for analyzing the results of classification, these script are also used to generate plots or tables.

4.1.1 Dataset

Starting point for all experiments is the dataset. It is used for both, training the classifiers and evaluating their performance. The requirements which have to be fulfilled by the used dataset have been defined in Section 3.5.

Some publicly available datasets are described by Guzella and Caminhas in [GC09]. There are a few datasets freely available that fulfill some of these requirements. However, they are either outdated (e.g. Enron mail corpus), are not addressed at users in the same organization and are composed from multiple dissimilar sources, or contain only one class of mail. This makes them unsuited for the evaluation of the proposed system.

The main benefit of using a previously examined dataset would be the possibility to compare this thesis' approach with other works. Since the proposed system is not intended as a replacement or improvement of existing anti-spam methods, but as an addition, this is not that important for this thesis.

Therefore, this thesis will be based not on a publicly available dataset, but on real life data provided by the IT security company genua[3], extracted from the company's request system. While some queues are exempt for legal reasons, for example human resources, a large part of the company's communication with the outside is present in the available data. To extend the collection of malicious mails, the dataset is augmented by mails from genua's mail honeypot that has been in use for multiple years. These systems already performed rough labeling of the data, which has been manually refined for other experiments within the company. The following evaluates the dataset in regards to the previously defined requirements:

[1] `https://www.csie.ntu.edu.tw/~cjlin/libsvm/`
[2] `https://www.cs.cornell.edu/people/tj/svm_light/`
[3] `https://www.genua.de`

1. **Good representation of real life data**
   Apart from the mails collected on the honeypot, the dataset consists exclusively of mails collected at a central point of a single organization. It therefore directly reflects data the proposed classifier would have had to analyze if it had been employed a few years ago.

2. **Sufficient amount of data**
   With 385958 mails, the dataset contains enough data.

3. **Long time span covered**
   The dataset contains mails from 2014 until 2018. This time span is long enough to cover changes in the trends of unsolicited mail, for example the appearance of ransomware in large quantities in 2015 and 2016.

4. **Has to be pre-labeled**
   All mails in the dataset are categorized as either *ham*, *spam*, *phising*, or *malware*.

5. **Must contain malign and benign mails**
   The dataset contains 99591 ham mails, 213824 spam mails, 59774 phishing mails, and 12769 malware mails. Both classes are therefore present in sufficient numbers.

6. **Must contain recent mails**
   With data up to 2018, recent trends in unsolicited mails should be present.

In conclusion, the dataset fulfills all previously noted requirements. This makes it ideally suited for the following experiments.

The mails in this dataset have been collected in the mbox format as defined in RFC 4155. Each source (genua's request system and the honeypot) consists of four mboxes, one mbox for each class: *spam*, *phishing*, *malware*, and *ham*.

4.1.2 Feature extraction

However, the mails in the mbox format cannot be used without further preprocessing. The dataset has to be transformed into a format that is easier to use. Furthermore, the mboxes still contain company-confidential information, and may not leave genua's network. This makes handling of the data impractical. The first step therefore is to convert the mboxes, and remove all sensitive information from the data.

For this task, an internal Perl script called `mails2features.pl`, which was also used by Gascon et al.[GUSR18] is used. It converts each class' mboxes into JSON files. Every mail becomes one JSON object, and every feature defined in Section 3.4 becomes one key-value pair. All sensitive information like mail addresses or contents are anonymized or removed during this process. Furthermore, every feature gets assigned to it's featureset. This results in one JSON file per feature set and category. For example the file `ham-transport.json` contains the transport features of all ham mails.

However, these JSON files are still very impractical, as they have a combined size of 3,5 GB and parsing that amount of JSON data is inefficient and slow. Therefore, these JSON files are given to another Perl script by Gascon et al.[GUSR18] called `features-embed.pl`, which combines the files for each featureset and transforms them into the LIBSVM[4]/SVM light[5]

[4] `https://www.csie.ntu.edu.tw/~cjlin/libsvm/`
[5] `https://www.cs.cornell.edu/people/tj/svm_light/`

format, which allows sparse data. Sparse data means, that zeros do not have to be stored. This results in one file per featureset. Each line in these files describe the features from the corresponding featureset that are actually present in the sample, as well as the samples' label. Using this sparse format greatly reduces the overall size of the dataset to around 500 MB.

4.1.3 Experiments

Now that the dataset is available in an easily processable format, it can be used for the experiments. These are conducted with different Python scripts using Python 2.7 and the machine learning library scikit-learn in version 0.18[6].

To implement the experiments, Jupyter Notebooks are used because they allow the interactive execution of code blocks independent of their order. These are basically still just Python scripts, and can be exported as such to be run as a whole with minor adjustments, without the necessity of using Jupyter.

Every experiment is done within a separate Notebook. However, all Notebooks make use of a developed library called `testtools`.

`testtools.py` library

This library is used by all experiments, as it includes some useful helper functions. The most important functions are:

- `test_classifier(...)`
  This function has been written explicitly to handle training, testing and evaluation of each classifier and data chunk. The training data with labels, test data with labels, and a classifier object are passed to this function. It then performs the following steps:
  1. The classifier is trained by calling the classifier's `fit` method. If the training data is empty, no training is performed. This may be the case if a previously trained classifier is tested with new training data.

     If the classifier is wrapped in a `GridSearchCV` object, the `fit` method call also determines the optimal hyperparameters before actually training the classifier.
  2. The test data's labels are predicted by calling the classifier's `predict` method, returning a list of predicted labels.

     Additionally, the results of the classifier's decision function or per-class probability is calculated by calling its `decision_function` or the `predict_proba` method.

     Because *KNeighborsClassifier's* implementation uses too much memory when predicting large chunks of test data, only 2000 samples at a time are given to the prediction functions, and the results are joined afterwards.
  3. The predicted and true labels are then compared to calculate F_1 score, general accuracy, and true-positive, true-negative, false-positive, and false-negative rates.

     The result of the decision function or the per-class probability is used to calculate the area under the ROC curve and the information necessary to draw it.

[6] `https://scikit-learn.org/0.18/`

4. Finally, the function returns these calculated evaluation metrics as well as the predicted labels in a Python dictionary.

- `get_data(...)`
  This function is given a list of featuresets that are requested. It reads these featuresets from their libsvm files using scikit's `load_svmlight_file` function. These different featuresets are then combined into one scipy CSR matrix by use of scipy's `hstack` function.

  The function then returns data and labels, randomly split into training, tuning and test subsets, using a 40 %, 30 %, 30 % split.

- `get_data_sorted(...)`
  Similar to `get_data(...)`, this function loads and combines the requested featuresets from the files. However, data and labels are in their chronological order and are not split into subsets.

  If needed, the labels can be requested in a more detailed format, not only giving information whether samples are malign/benign, but also to which subclass of unsolicited mail (*spam*, *phishing*, *malware*) they belong.

- `load_results(...), save_results(...)`
  This functions are used to load or save the results from or to disk, using Python's msgpack library.

- `multiclass_eval(...)`
  Two lists can be given to this function: one containing the true labels (with subclasses) of the classified samples, one containing the predicted labels. The function then calculates detection rates for each subclass separately, returns these results, and visualizes them in a table using the PrettyTable library.

Several other functions are defined in the `testtools.py` library which are of no further relevance to this thesis as they are only used for testing purposes.

Jupyter Notebook

The following gives an overview over all relevant Jupyter notebooks that have been written to conduct this thesis' experiments.

- `10_hyper_svm.ipynb`, `11_hyper_KNN.ipynb`, `12_hyper_log_reg.ipynb`, `13_hyper_bayes.ipynb`, `14_hyper_random_forest.ipynb`
  These scripts are used to conduct experiments to determine the optimal hyperparameters for the used classification algorithms.

  The results of these experiments are used in Section 4.2.1.

- `20_featureset_selection.ipynb`
  This script is used to determine the optimal featureset for every classifier.

  The results of this experiment are used in Section 4.2.2.

- `21_featureset_selection_reduced_training.ipynb`
  This script is used to determine the optimal featureset for every classifier with reduced amounts of training data.

  The results of this experiment are used in Section 4.2.3.

- `30_reduced_training.ipynb`
  This script is used to evaluate the classifiers' performance with various amounts of training data.

  The results of this experiment are used in Section 4.3.1.

- `32_training_duration.ipynb`
  This script is used to evaluate the classifiers' computational complexity and their training and classification durations.

  The results of this experiment are used in Section 4.3.3.

- `33_concept_drift_fixed.ipynb`
  This script is used to evaluate the classifiers' susceptibility to concept drift and the necessity of periodic retraining.

  The results of this experiment are used in Section 4.3.2.

- `41_per_class.ipynb`
  This script is used to examine the detection rates for each subclass of unsolicited mail (*spam*, *phishing*, *malware*) separately, when using the SVM classifier.

Several other scripts have been written to test some implementation details, or try out different approaches to evaluate featuresets or classifiers. The yielded results however are not directly of any use for this thesis, and will therefore not be discussed further.

All these notebooks share a very similar structure. The first step in each of these scripts is to load the data. Depending on the experiment, either `get_data` or `get_data_sorted` from `testtools.py` is used.

If only parts of the dataset are to be used at a time, the dataset is split up into chunks using Python's list operations. The chunk size depends on the requirements of each experiment.

When the data is ready and available for training, the classifiers have to be initialized. If the classifiers' hyperparameters are set manually, the classifiers are created using scikit-learn's *LogisticRegression*, *KNeighborsClassifier*, *LinearSVC*, *BernoulliNB*, and *RandomForestClassifier* classes directly.

If the hyperparameters have to be determined specifically for the data, scikit-learn's `GridSearchCV` functionality is used. The `GridSearchCV` class is basically a wrapper around the classifier classes. It gets passed a classifier object and the list of possible hyperparameters. It can then be used like any other classifier in scikit-learn with the difference that it determines the optimal hyperparameters using the training data and cross-validation, before the actual training starts.

Now that the training and test data is preprocessed and the classifier objects are ready, the actual training and testing begins. In most experiments, this is done by calling `testtools`' `test_classifier` function.

The returned results are either analyzed directly, or appended to a list of results if classification is done more than once. In most experiments, the process of splitting the dataset

into chunks, training the classifiers with one chunk and testing them with another chunk, is done repeatedly by iterating over the whole dataset in a for-loop.

Finally, the results are written to disk using `testtools`' `save_results` function. This allows to just load the results when doing analyses without having to repeat the potentially time-consuming progress of training and predicting.

Analysis

Analysis of each experiment is done in the corresponding Jupyter notebook. At first, the results are read from disk, using the `load_results` function from `festtools.py`. The following steps depend highly on the wanted information.

In most cases the results are either printed using the PrettyTable library, or Matplotlib is used to visualize the results.

4.2 Evaluation of the featuresets

This section covers the evaluation of the featuresets. The goal is to find the best featureset for each classification algorithms. To achieve this, every algorithm is trained and tested with every possible combination of the described featuresets. For these experiments, the dataset is randomly split into three subsets as described in [CF18]:

- 40 % training data, used to train the classification algorithms
- 30 % validation data, used to optimize the algorithms' hyperparameters
- 30 % test data, used to evaluate the classifiers' performance

The split between validation and test data is necessary because the final evaluation of a classifier's performance can not be done with data that has been used to determine the hyperparameters. It always has to be done with previously unseen data.

4.2.1 Hyperparameters

For each classification algorithm the relevant hyperparameters will be determined using the documentation provided by the scikit-learn library and a range of typically used values is preset. Hyperparameters are parameters that define the algorithms' behavior which cannot be learned by training.

The following are the relevant hyperparameters for each algorithm, and their corresponding values from which has to be selected. For the specific meaning of the hyperparameters refer to Section 2.2.

- Support Vector Machine
  Parameter: C, regularization parameter
  Values: $[0.001, 0.01, 0.1, 1.0, 10, 100, 1000]$
- k-Nearest Neighbors
  Parameter: k, number of neighbors to look for
  Values: $[3, 5, 10, 20]$

- Random Forest
  Parameter: n, number of trees in the forest
  Values: $[5, 10, 50, 100]$

- Naive Bayes
  Parameter: α, smoothing parameter
  Values: $[0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1, 1.0]$

- Logistic Regression
  Parameter: C, regularization parameter
  Values: $[0.001, 0.01, 0.1, 1.0, 10, 100, 1000]$

For each featureset and every possible value of the mentioned hyperparameters, the corresponding classification algorithm is trained using the training data, and it's performance evaluated on the validation dataset using the AUROC metric. An exception is made for the k-Nearest Neighbors classifier, because it's AUROC is not necessarily a valid performance indicator for small k. Therefore the F_1 score is used to select KNN's best hyperparameters.

The hyperparameter values with the best result for each algorithm and featureset are found in Table 4.1. For the Support Vector Machine and the Logistic Regression classifier, the optimal values are within a typical and expected range.

The Random Forest classifier performs best when using 100 trees, and it might achieve even better results for a larger number of trees. However, training time and memory usage grow linearly with the number of trees, and as shown by Oshiro et al. in [OPB12], the performance usually doesn't improve much when using more than 100 trees. Therefore it remains limited to 100 in this case.

The Naive Bayes classifier requires very small regularization parameters for some featuresets. This usually indicates that the classifier failed to properly generalize the information found in the data. This has to be kept in mind when interpreting the classifier's results. Section 4.2.3 shows that the classifier indeed behaves strangely for the affected featuresets.

The k-Nearest Neighbors classifier always performed best for $k = 3$, as the smallest tested value. For even smaller values of k, the classification performance might increase further, however the classifier would become too susceptible to outliers. Depending on the data, a small k usually indicates that a neighbor search might not be the best approach because the clusters of objects with identical labels are very small.

Featureset	SVM: C	Log. Reg: C	Forest: n	Bayes: α	KNN: k
mail	0.1	1.0	100	0.000001	3
mua	0.1	10.0	100	0.000001	3
transport	0.1	10.0	100	0.1	3
mail + mua	0.1	1.0	100	0.000001	3
mail + transport	0.01	1.0	100	0.001	3
mua + transport	0.1	1.0	100	0.01	3
all	0.01	1.0	100	0.000001	3

Table 4.1: Optimal hyperparameters per classifier and featureset

4.2.2 Results using the full dataset

To find the best suited featureset for the classification task, a comparison of the featuresets for each classifier is needed. Thus, every classifier is now trained with each featureset and the corresponding optimal hyperparameter combination using the training data. Its' performance is then measured by testing it on the test data. By regarding the AUROC results as shown in Table 4.2, it becomes clear that all algorithms are able to predict the malignity of the mails in the test data very well, since all algorithms show an AUROC of at least 0.99. However it is difficult to determine a single featureset that is best suited for the classification task, since the differences between the results are very small and no single featureset can be made out to be the best for all classifiers.

Featureset	SVM	Log. Reg.	Forest	Bayes	KNN
mail	0.996950	0.997683	0.997666	0.989515	0.989226
mua	0.996858	0.997500	0.996658	0.986941	0.988356
transport	0.997225	0.998278	0.996976	0.996454	0.986904
mail + mua	0.998020	0.998042	0.997721	0.986853	0.990351
mail + transport	0.998020	0.998507	0.998298	0.991975	0.990368
mua + transport	0.998407	0.998662	0.997696	0.990943	0.989449
all	0.998443	0.998644	0.998198	0.989294	0.990796

Table 4.2: AUROC results with optimal hyperparameters

To gain further insight it is necessary to amplify the differences between the individual results.

4.2.3 Reduced amount of training data

Making differences between the results more visible can be achieved by reducing the amount of training data. This also makes the experiment more realistic, since a real life system rarely has access to data collected over several years.

For this experiment, the data is ordered chronologically and broken up into chunks of n samples. All classifiers are now trained using the first chunk, and tested against the following chunk. The classifiers are then reset, trained with the second chunk, and tested with the third chunk. Basically, the classifiers are trained with each chunk, and tested against the following chunk, until there are not enough samples left to form a complete test chunk. Finally, the results of every test are averaged. This procedure is visualized in Figure 4.3.

This process is repeated for every featureset and for different n. The varying chunk sizes reflect different time intervals:

- $n = 300$, corresponding to one to a few days of collected mails
- $n = 1000$, corresponding to about one week
- $n = 5000$, correspoding to about one month
- $n = 50000$, corresponding to about one year

Since the training data is constantly changing during this process, using static values for the hyperparameters is not the optimal way. Instead, a procedure comparable to the

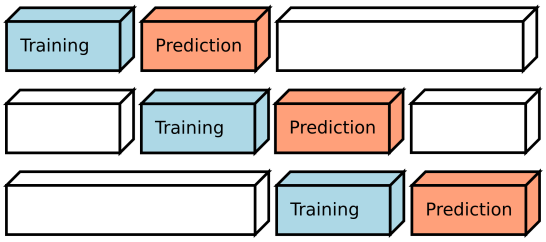


Figure 4.3: Partitioning of training and test data to evaluate the classifiers' performance with reduced amount of training data

approach recommended by Salzberg in [Sal97] is used: before the classifiers' training on each chunk, the optimal hyperparameters are determined by used k-fold cross-validation on the training chunk. In this case, $k = 3$ will be used. Since cross-validation is not used to evaluate the classifier, but only to determine the hyperparameters, a larger k is not necessary and would only lead to issues for small chunk sizes.

For the case of $n = 1000$, the results for every classifier have been visualized in Figure 4.4. The best suited featureset for every classifier has been highlighted by color. The complete results of this experiment can be found in Table A.1 in the Appendix. It can be observed that there is no single featureset that works best on all classifiers. Each classifier shows different behavior and has its own preferred featureset. The following describes which featureset fits best for each classifier:

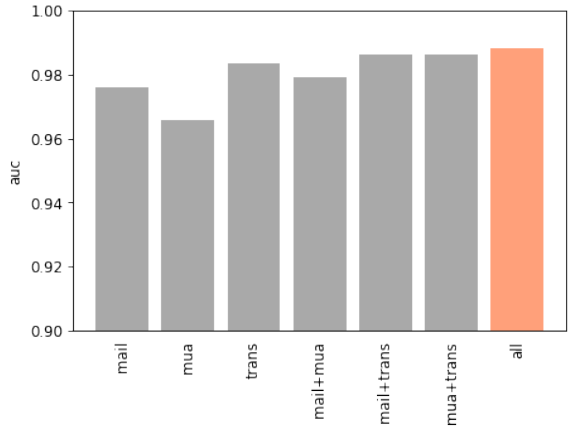
- SVM: *all*
  The Support Vector Machine works best with the *all* featureset consisting of all three featuresets. There is one deviation for $n = 300$, where it performs better with the *mail + transport* featureset. However the gain compared to the *all* featureset is small, and *all* performs better in all other cases.

- Logistic Regression: *mail + transport*
  Logistic Regression works well with the *mail + transport*, *mua + transport* and *all* featuresets with only marginal differences. However, performance with little training data is important for practical applications and for smaller amounts of training data, like $n = 300$ and $n = 1000$, Logistic Regression performs best with the *mail + transport* featureset.

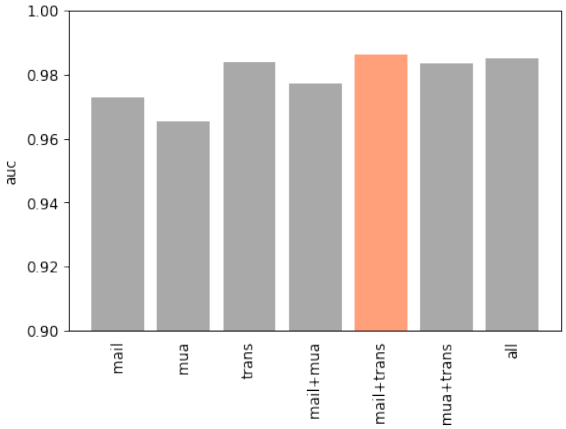
- Random Forest: *mail + transport*
  With only one exception, the Random Forest classifier performs best with the *mail + transport* featureset.

- Naive Bayes: *transport*
  Without exception, the Naive Bayes classifier works best with the *transport* featureset. It can be noted that the best results are achieved where the smoothing parameter α was the highest (see Section 4.2.1). This would fit the assumption that for some featuresets the algorithms fails to properly generalize the information found in the training data, therefore requiring a very small smoothing parameter.

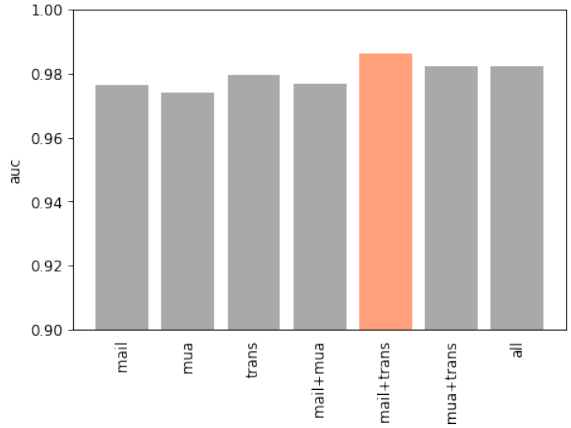
  A reason for this behavior could be that the features' format in those featuresets does not work well with the probabilistic approach of a Naive Bayes classifier.



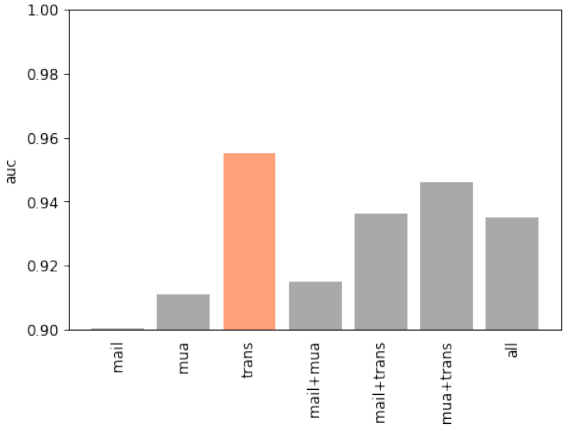
(a) Support Vector Machine



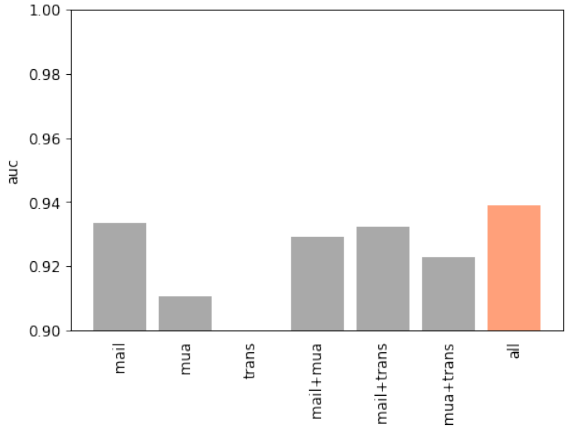
(b) Logistic Regression



(c) Random Forest



(d) Naive Bayes



(e) KNN

Figure 4.4: Averaged AUROC with $n = 1000$ for different classifiers and featuresets

- k-Nearest Neighbors: *all*
  Without exception, the k-Nearest Neighbors classifier works best with the *all* featureset.

Interesting to note is that all classifiers work best when the *transport* features are present in the used featureset. This shows that information regarding a mails path is particularly useful in detecting unsolicited mails.

Apart from the Naive Bayes classifier, which seems to struggle generalizing from the training data, all classifiers work reasonably well with all featuresets. However, to achieve the best possible results, each classifier will have its own optimal featureset for the following experiments.

4.3 Evaluation of the classification algorithms

Now that the optimal featureset per classifier has been found, the best suited classifier has to be determined. The following sections describe a number of experiments that aim at examining the classifiers' behavior in situations relevant to a possible practical application.

The following experiments will be conducted:

- **Required amount of training data**
  Section 4.3.1 investigates how much labeled training data is necessary to achieve good results. This will give precious information about how much labeled data is required to employ these algorithms in practical applications.

- **Long-term stability**
  The classifiers' susceptibility to concept drift as described in [GMCR04] and [Tsy04], and their long-term performance is looked at in Section 4.3.2. These experiments will allow to estimate how often retraining has to be performed if such a classifier would be used in real-life.

- **Computational complexity**
  Section 4.3.3 will examine the training and classification durations of each classifiers. This will tell how much processing power would be needed in real-life applications.

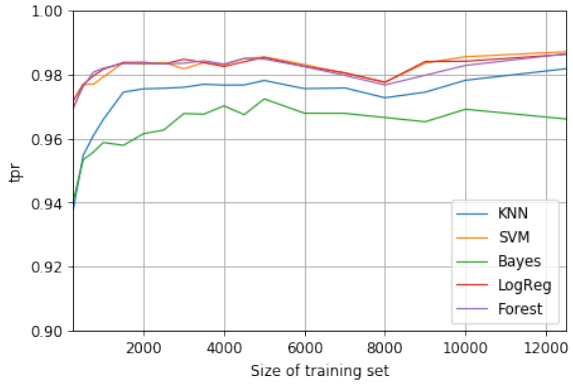
4.3.1 Required amount of training

The biggest restriction in practical applications is the availability of labeled training data. Only a small part of the mails can be labeled automatically, and manual labeling is a very time-consuming task. Therefore it is important to employ classifiers that already provide good results given only a small dataset for training. The following experiment will show the classifiers' performance for various amounts of training data.

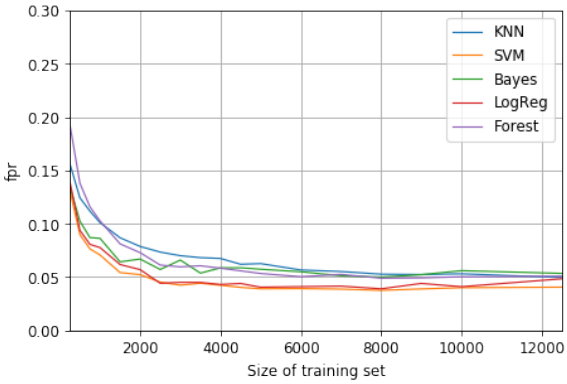
The conducted experiment from Section 4.2.3 as shown in Figure 4.3 will be repeated, with two small changes:

- Each classifier will only be tested with its best performing featureset.

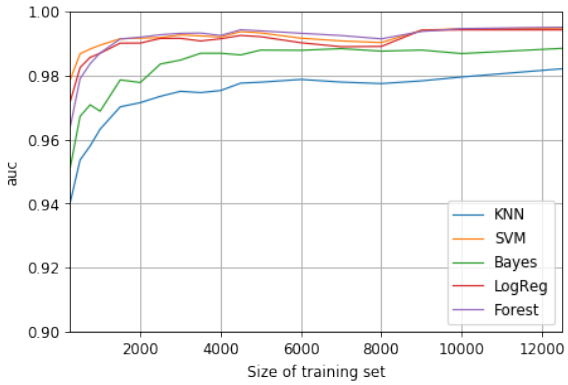
- Instead of only four chunk sizes n, the tests will be run with chunk sizes ranging from 250 to 12500.



(a) True-positive rate



(b) False-positive rate



(c) AUROC

Figure 4.5: Results with different amounts of training data

AUROC and false-positive rate is averaged for each chunk size and will be plotted in relation to the chunk size. This can be seen in Figure 4.5.

The AUROC values (Figure 4.5c) are mostly equal for the SVM, Random Forest and Logistic Regression classifiers. The Naive Bayes and the k-Nearest Neighbors classifiers perform noticeable worse, especially for small training sets. While the better classifiers already reach AUROC values close to their maximum with only 2000 training samples, Naive Bayes requires around 5000 samples to achieve the same result. K-Nearest Neighbors' performance is still clearly improving at more than 10000 samples. This shows that SVM, Random Forest and Logistic Regression therefore are generally better suited for situations where only limited amounts of training data is available.

The corresponding false-positive rate is shown in Figure 4.5b. Generally it can be said that it requires more samples to achieve the best false-positive rate, than to achieve the optimal AUROC. All five classifiers reach a false-positive rate of around 5 % with enough training data. However, SVM and Logistic Regression reach this point with less training samples, around 2000, than the others, and are the only classifiers staying below 5 %.

Considering this and the Support Vector Machine's AUROC and false-positive rate, it seems to be the best suited classifier for applications with limited available training data.

4.3.2 Long-term stability

However, behavior with recent data is only one part in selecting the optimal classification algorithm. Another important part is the performance of the classifiers as their models get older, since malicious and unsolicited mails might change their characteristics, or attackers might change their behavior and methods. To examine this, the following experiment is conducted:

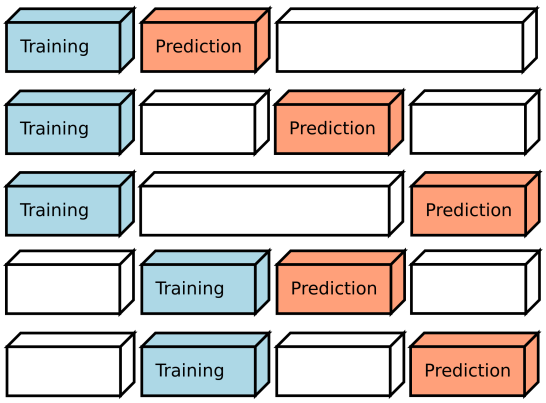


Figure 4.6: Partitioning of training and test data to evaluate the classifiers' long-term stability

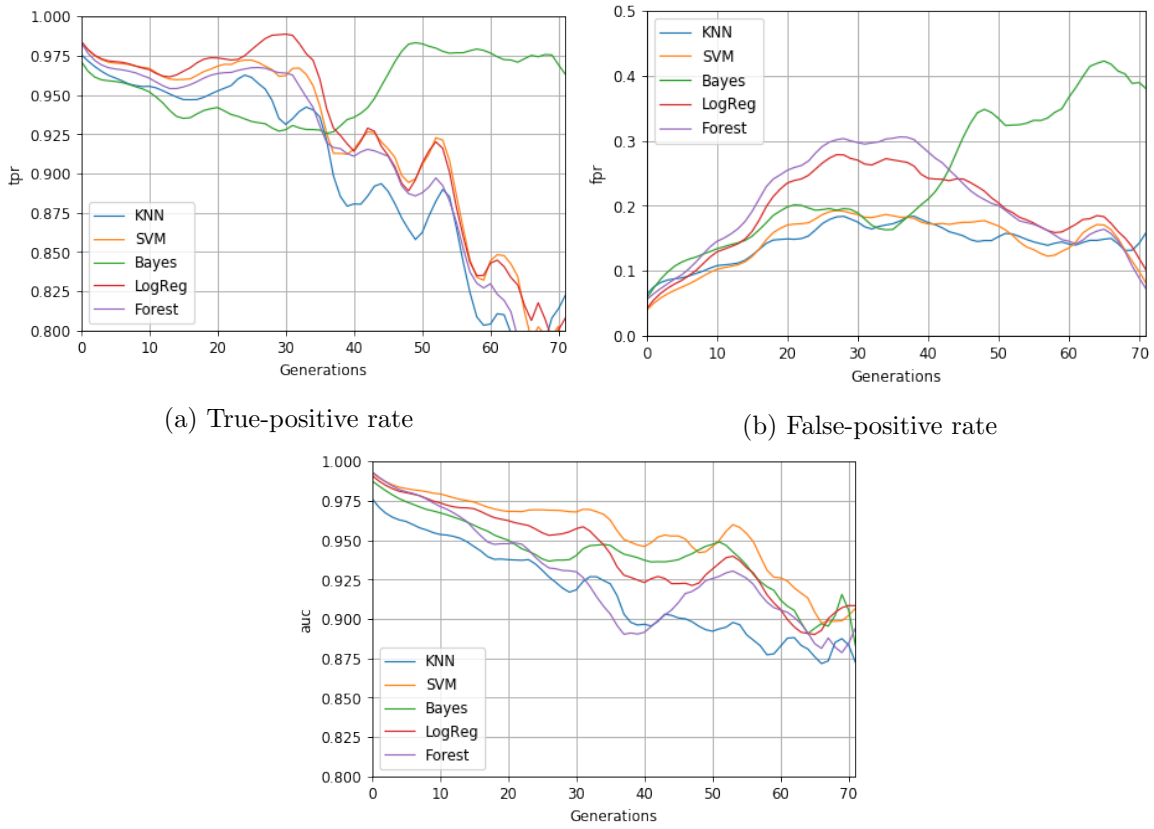
The dataset is split into chunks of n samples, and the classifiers are trained on one chunk. They are then tested against each following chunk, containing newer samples. The chunk immediately following the training data is called generation zero. For every later chunk the generation counter is increased by one. The training is done on every chunk, testing on all following chunks. The results are averaged for every generation. The number of samples per chunk n is set to 5000, the size where most classifiers reached results close to their peak performance, equivalent to about one month of data. This procedure is visualized in Figure 4.6.

This leads to a lot of results for early generations, but only a few for late generations. To avoid skewed results, generations where less than five results to calculate the average from are discarded.

Figure 4.7 shows the results of this experiment over zero to 71 generations for chunks of 5000 samples.

For the SVM, Random Forest, and Logistic Regression classifiers, the true-positive rate (Figure 4.7a) is steady for a surprisingly long time and stays above 95 % for the first 30 generations. KNN performs similar, but it's true-positive rate drops off a bit sooner and faster. During that time however, the false-positive rate (Figure 4.7b) increases for these four classifiers. It finally levels out after 50 generations at around 15 %, with a maximum of around 25 % for Random Forest and 20 % for Logistic Regression.

Naive Bayes' behavior is similar with a lower true-positive rate for the first 35 generations, but then changes completely: after 40 generations the true-positive rate climbs back up to 97.5 %, at the cost of an also very high false-positive rate between 30 % and 40 %. This means, that the classifier loses the ability to properly distinguish between benign and malign samples and tends to overall tag a lot of them as malicious, leading to high true-positive and



(a) True-positive rate

(b) False-positive rate

(c) AUROC

Figure 4.7: Results with gradually older training data

false-positive rates.

The AUROC steadily decreases for all classifiers as their training data gets older, however, Logistic Regression's and SVM's AUROC decreases the slowest, making those the best suited classifiers for situations where no recent data is available.

The fact that the true-positive rate is initially steady and the false-positive rate increases has implications for practical applications. For example a system that is supposed to identify malicious incoming mails. Ideally it should be retrained on a regular basis. However, if retraining is not possible for some reason, this mostly effects usability, not security, because the system will still tag roughly the same amount of malicious emails correctly. It just generates more false alarms.

This experiment has also been conducted with a smaller chunk size of only 2000 samples, where it showed comparable results.

4.3.3 Training and classification duration

Closely linked to the topic of retraining is the question how long it takes to retrain a classifier. To examine this, all classifiers are trained using chunks of 10.000 samples and then classify 2.000 samples, while measuring training and classification durations. These times are then averaged per classifier and shown in Table 4.3.

Algorithm	Training	Classification
SVM	0.076 s	0.001 s
Log. Reg.	0.769 s	0.001 s
Forest	27.813 s	0.131 s
Bayes	0.015 s	0.017 s
KNN	0.005 s	2.418 s

Table 4.3: Training (10k samples) and classification (2k samples) duration

While there are significant differences in training and classification duration, no classifier should have problems with computational complexity in practical applications. In a worst-case application no mails can be delivered to end users during retraining. The training duration is highest for the Random Forest classifier with 28 seconds. A delay of 28 seconds in mail delivery will barely be noticeable to end users, as long as it isn't done after every received mail. The worst case when classifying mails of 2.4 seconds for the k-Nearest Neighbors classifier will also not be critical since it corresponds to a time of less than 0.002 seconds per mail.

All classifiers thus are viable options from the aspect of computational complexity.

4.4 Conclusions

The results from the previous experiments are now combined to select one classification algorithm. Table 4.4 presents a quick overview over the performed experiments and how each classification algorithm performed in the corresponding category.

Section 4.3.1 showed that the Support Vector Machine is best suited for applications where the amount of labeled training data is limited. Section 4.3.2 points out that the same classifier also works best in situations where no current data is available for training. The differences in computational complexity as shown in Section 4.3.3 should have no implications on the choice.

This means that a Support Vector Machine is best suited for the proposed approach of detecting unsolicited mails. The best featureset depends on the used classification algorithm, as shown in Section 4.2. For the Support Vector Machine it contains all features described in Chapter 3.4.

However, in a lot of situations the differences are only small. For example, Logistic Regression or Random Forest would also be good choices. For both, the preferred featureset is *mail + transport*, however, Section 4.2.3 indicates that both also perform well with all features.

The main benefit of the Support Vector Machine is the low false-positive rate compared to other algorithms, which is relevant to the planned application as an additional detection system. In situations where a low false-positive rate is not as important, a decision for one of the other classifiers can be reasonable. Only the Naive Bayes and k-Nearest Neighbors classifiers should not be used, as they provide no benefit compared to other, better performing algorithms.

This chapter shows that the proposed solution generally works to detect unsolicited mail, and that best performance can be achieved using the Support Vector Machine and all features. This knowledge is used in the following Chapter 5, where the proposed solution's

	SVM	Log. Reg	Forest	Bayes	KNN
Required amount of data	+	+	+	–	–
Long-term stability	+	+	o	–	–
Training duration	+	o	–	+	+
Classification duration	+	+	+	+	–

Table 4.4: Evaluation results

adoption into real life applications is considered. For the previously conducted experiments, an existing pre-labeled dataset could be used for training. Such a dataset might not be available when applying the solution in real-life. Chapter 5 will therefore examine how the necessary training data can be collected automatically.

5 Automatic labeling

Chapter 4 has shown that the proposed features are indeed suited to detect malicious mails, given a labeled dataset. In most cases however, there is no pre-labeled dataset available. Using a generic dataset might not work, since the detection is based on similarity in structural traits, which might take on distinct manifestations for different destination networks or organizations.

Ideally, for every organization that should be protected with the proposed approach, a unique training dataset is used. This unique dataset represents the structural traits in the form that are specific to the mails received by the organization's mail server. This chapter will present a new approach how such a unique training dataset can be generated by automatically labeling incoming mails, and examine the detection performance of this approach compared to the previous results where a pre-labeled dataset has been used.

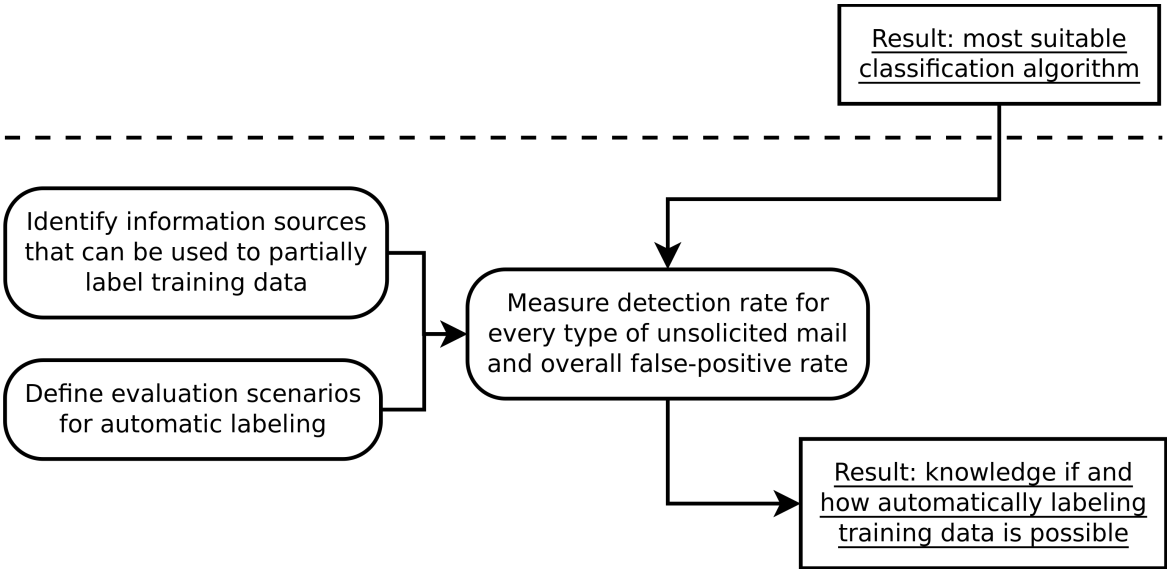


Figure 5.1: Evaluation of featuresets and classifiers

Figure 5.1 shows the procedure to examine such an automatic labeling system. Section 5.1 describes the general concept how this system should work. The possible information sources that can be used are defined in Section 5.2.

In Section 5.3 this system is evaluated by first simulating incoming mails passing through the mail server, anti-virus software, and spam filter before reaching the user. Since it is difficult to simulate the behavior of anti-virus software or a spam filter at the time the mail has been received, will we assume various detection and false-positive rates for these steps. The amount of unsolicited mails reaching the user as well as false-positive rates are measured. These results represent the currently common, state-of-the art protection against unsolicited mail. In the second step, the automatic labeling system and a classifier will be added, and

the simulation is repeated with the same set of mails. Detection and false-positive rates are measured and compared to the previous, common detection approach. This will show if the automatic labeling and the use of a classifier will actually provide a benefit in the detection of unsolicited mails.

5.1 Concept

The idea behind the automatic labeling is to automatically generate training data on the mail server of the network that has to be protected. Every incoming mail passes through the common steps to detect unsolicited mails: DNS real-time blacklists, anti-virus software, and a spam filter. Information gathered during these steps, complemented by other sources as described in Section 5.2, will be used by the labeling system to unambiguously identify every incoming mail as either malign or benign. For example if a mail is a response to a previously sent mail, it can be identified as benign, or if a virus has been found by anti-virus software, it can be identified as malign. In these cases, the mail will be added to the training data. On the other hand, if no decision can be made using such predefined rules, it is left undecided, and the mail will not be used for future training.

If the automatic labeling rules can identify a small fraction of mails of both classes, the training dataset will grow over time to provide a sufficient foundation for a classifier to identify unsolicited mails. As soon as the training dataset has reached a size that is sufficient for proper detection of unsolicited mail, it will be used to train a classifier. In Chapter 4, the required amount of training data for this classification task has been examined. Regarding these results, training in the following experiments will start once 2500 samples have been added to the training dataset. To make sure the classifier stays up to date, it will be retrained after every 2500 newly received mails.

This classifier will be added as a last step for incoming mails. Every mail then has to pass real-time blacklists, anti-virus software, a spam filter, and lastly the classifier that has been trained using automatically labeled data.

5.2 Information sources

The success of the automatic labeling depends on the rules and the possible information sources that can be used to determine to which class a mail should be assigned. Since the automatic labeling is performed on a system which can see all incoming and outgoing mail traffic, it can combine multiple different information sources to achieve the best possible results and ideally a large training dataset.

The following information sources are imaginable:

- **Anti-Virus**
  Anti-virus software can detect a significant amount of malware without raising a lot of false-positive alarms. However it's only effective against known malware and new malware samples are often not detected by common anti-virus products. Therefore, if anti-virus detects malware, an incoming mail will be automatically labeled as malicious. If not, no decision is made.

  The information gained by anti-virus software could even be enhanced by conserving every mail after delivery to the recipient. It can then be given to the anti-virus software

a few days after its arrival, when new patterns for the virus scanner are available. This should allow to automatically label almost every mail that contains malware as malicious for the training dataset. However, storing mails after delivery to the recipient might violate against data protection regulations.

- **DNS real-time blacklists**
  If a SMTP connection is made to send a mail to the receiving server, it is usually checked if the sender is a known source of unsolicited mail, by checking if the sender's IP is found on common blacklists. If the sender is identified as a source of unsolicited mail, the SMTP connection will usually be terminated during the handshake before the mail is sent to the receiving server.

  However, it would be possible to wait with the termination of the SMTP connection until the whole mail has been sent. Instead of then accepting the transmission with SMTP status code `250 OK`, the dialog can be ended with status code `554 Transaction failed`, and the mail is labeled as malicious.

- **Greylisting**
  Similar to DNS real-time blacklists, mails filtered by greylisting could be used as malicious samples for training.

- **Reference to sent mail**
  The mail server performing automatic labeling has access to all outgoing mails. This can be used to store every sent mail's `Message-Id`. Every incoming mail that references this `Message-Id` in its `In-Reply-To` or `References` headers is most likely a valid reply to a sent message and can be labeled as benign.

  A time limit for this mechanism should be considered to prevent abuse of mails that have been sent to public mailing lists.

- **Feedback from the user**
  Ideally, some kind of user feedback could be implemented into the system. This could for example be achieved by monitoring the users' IMAP folders to detect when a user moves a mail into his spam folder or moves a mail from spam to the regular inbox.

  A less sophisticated but simpler way would be to provide a mail address to which the user can forward false-positive or false-negative mails to, or a daily report which mails have been filtered.

- **Feedback from the classifier**
  Another information source is the classifier that is trained with data from the automatic labeling. If the classifier can return it's confidence about a decision, the mails where the classifier was the most confident can be automatically labeled. This should be limited to only very few mails, where the confidence is the highest, for example the 1 % with the highest confidence value.

- **Spam filter**
  Spam filters usually calculate a score for each mail. If the a mail's score exceeds a predefined threshold it's marked as spam. This score could also be used to automatically label the training data. However, spam filters are prone to false-positives which should not be introduces into the training data. Therefore a second, higher threshold would be set and only mails exceeding this threshold would be labeled automatically.

5.3 Experiments

This section will cover the evaluation of the proposed automatic labeling system. The goal is to determine whether a classifier using the structural features described in Chapter 3.4 can improve the current situation of detecting unsolicited mail, if the classifier is only trained using automatically labeled data.

5.3.1 Experimental setup

To achieve this, we compare two detection pipelines, which are depicted in Figure 5.2:

- **State-of-the-art detection**
  The simulated state-of-the-art detection system represents the currently common approach to protect users from unsolicited mail. It consists of three steps: DNS real-time blacklists, anti-virus software, and a spam filter. Every incoming mail passes through these three systems, and is delivered to the end user if no system detected anything malicious.

- **Proposed detection with automatic labeling and classifier**
  The proposed detection system enhances the state-of-the-art system by introducing two new steps, as well as the automatic labeling system. Every incoming mail is first checked if it is a reply to a previously sent mail. If it is, the mail is registered as a benign sample by the automatic labeling system, and is delivered to the user.

  If it is not a reply, the mail then has to pass the DNS real-time blacklist, anti-virus software, and spam filter. If the mail is detected as malicious by either of these systems, it is discarded. If the maliciousness was detected by the blacklist or the anti-virus software, the mail is registered as a malign sample by the automatic labeling system.

  If the mail has not been detected as a reply or as malicious by any of the systems, the classifier trained with the automatically labeled data makes a final decision whether the mail is benign or malign. A benign mail is delivered to the user, a malign mail discarded. If the classifier is unavailable because there wasn't enough training data collected previously, the mail is considered benign. If the classifier's confidence value about the decision is higher than the percentile value of the 95th percentile, it will be registered as a sample of the predicted class by the automatic labeling system. As the classification algorithm, the same Support Vector Machine as in the previous chapters will be used.

  This pipeline will be tested in two variations. A basic version where all available data is used for training, and a balanced version, where the training data is limited to 25000 samples and both classes are present in equal parts.

Ideally, the used dataset would contain the results of anti-virus software or spam filters at the time of the mail's arrival, to correctly replicate if a malicious mail reached the end user undetected, or if a benign mail was falsely detected as malicious. Unfortunately, this information has not been collected. Therefore it is necessary to make assumptions about the detection and error rates for every step.

These rates will then be used to simulate the detection systems. For example, if a true-positive rate of 50 % is assumed for the anti-virus solution, 50 % of the malware mail reaching the anti-virus step will randomly count as successfully detected.

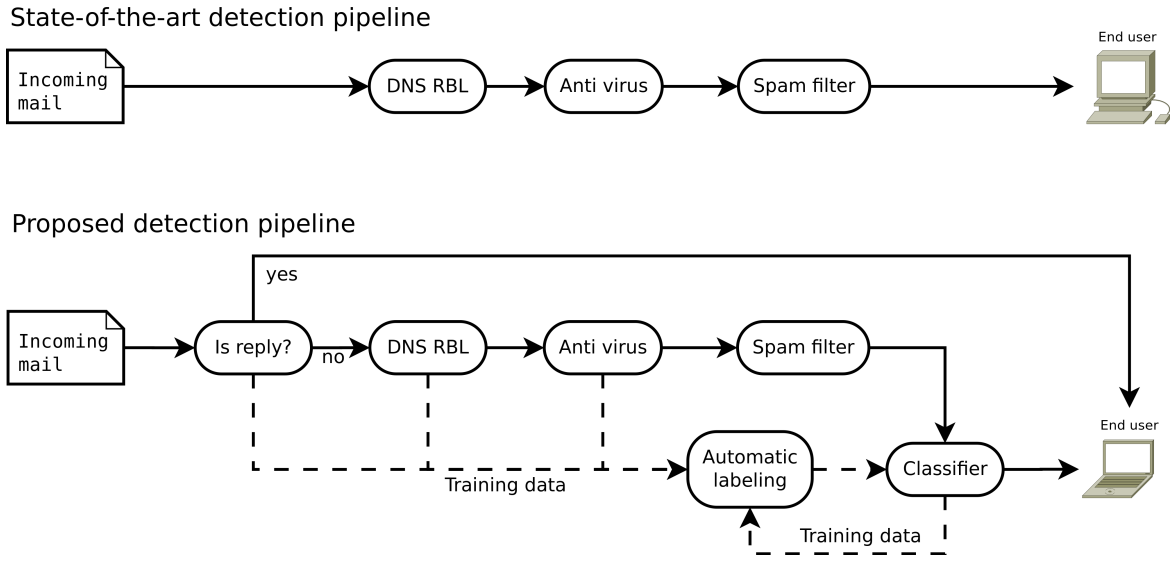


Figure 5.2: Experimental setup

To completely simulate all steps, true-positive and false-positive rates of DNS real-time blacklists, anti-virus software, and spam filter has to be assumed. Furthermore, the ratio of mails that are replies to previously sent mails has to be determined. Since about 30 % of the ham mails in the used dataset are replies, this value will be used for all following experiments.

However, it is hard to make good assumptions about the detection rates of real-time blacklists, anti-virus software, and spam filter, because these rates are most likely very dependent on the type of mails the user usually receives and the configuration of the spam filter.

Therefore, these experiments do not focus on trying to correctly simulate these detection systems. Instead, various scenarios with different detection rates are created, and it is examined how the overall detection and error rates change by adding the classifier with automatic labeling.

5.3.2 Test scenarios

Three scenarios are used to evaluate the proposed detection system. One with high detection rates of existing systems without any false-positives. The second scenario with the same high detection rates, but also with significant false-positive rates to examine how a large number of falsely classified training samples influences the classification performance. And the last scenario with lower detection rates, as well as low false-positive rates.

Scenario 1, high detection, low false-positive rates

Scenario 1 represents the ideal world, where only a very small fraction of unsolicited mails stays undetected, while no false-positives occur. For the first step in the pipeline, the DNS real-time blacklist, a detection rate of 50 % of all unsolicited mail is assumed. The detection rate of anti-virus software is set to 95 %, and the spam filter detects 90 % of all spam in this scenario. False-positives rates are 0 % for all steps.

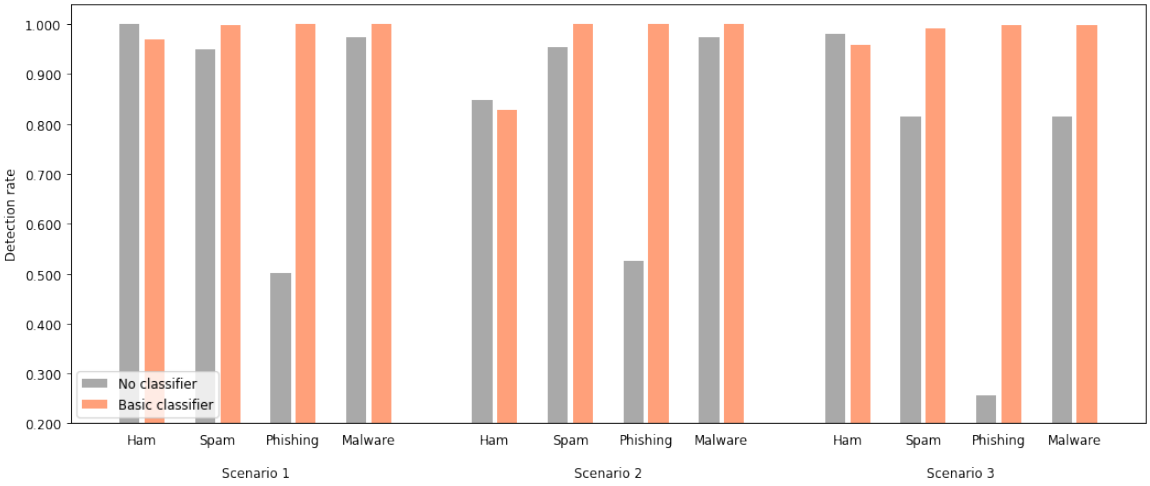


Figure 5.3: Detection rates without and with basic classifier

Scenario 2, high detection, high false-positive rates

This scenario is supposed to show how the classifier performs if the conventional steps produce a large number of false-positives, that are also introduced into the training data. Therefore the detection rates are the same as in the previous scenario, but false-positive rates of 1 % for the real-time blacklist, 5 % for the anti-virus software, and 10 % for the spam filter are assumed.

Scenario 3, medium detection, low false-positive rates

In this scenario, the simulated detection systems are configured in a way that generates less false-positives and therefore also has lower detection rates. It is assumed that the real-time blacklist detects 25 % of unsolicited mail without any false-positives, anti-virus software detects 75 % of malware with a false-positive rate of 1 %, and the spam filter detects 75 % of spam with a false-positive rate of also 1 %.

5.3.3 Results and practical considerations

Every mail in the dataset will pass through the described detection pipelines for each scenario. At the end, the predicted class labels are compared with their true labels. The evaluation focuses on the detection rates for each class separately.

Figure 5.3 and 5.4 show the results of the experiment. Both figure show the detection rates of the four classes *ham*, *spam*, *phishing*, and *malware* for every scenario and experiment. The detailed, numeric results can also be found in Table A.2 in the Appendix.

In this case, the *ham* detection rate is equivalent to the true-negative rate. Since the true-negative rate is the complement to the false-positive rate ($1 - tpr = fpr$), a higher true-positive rate equals a lower false-positive rate and vice versa.

Figure 5.3 shows only the difference between the state-of-the-art pipeline without classifier and the proposed pipeline with a basic classifier using all available training data, because the differences between both classifiers is very small.

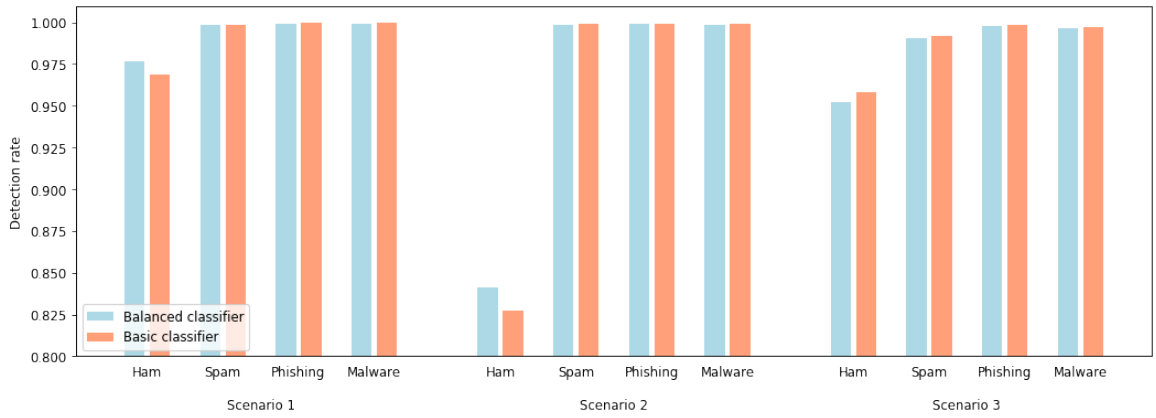


Figure 5.4: Detection rates compared between basic and balanced classifier

The differences between the two classifiers is shown in Figure 5.4. Please note the changed scaling of the y-Axis.

The detailed, numeric results can also be found in Table A.2 in the Appendix.

Scenario 1

For the first scenario, two observations can be made. First, the detection rates of the three malicious classes improved significantly by introducing the classifier. For the *spam*, *phishing*, and *malware*, the detection improved from 95.0 %, 50 %, and 97.4 %, to 99.9 % for all three classes. For these metrics, the differences between the basic and the balanced classifier are very small.

Second, the false-positive rate increased from 0 % to 3.1 % for the basic classifier, and to 2.3 % for the balanced classifier.

Automatic labeling collected the necessary training samples after circa 15000 incoming mails.

Scenario 2

The results in the second scenario are very similar to the previous one. By adding the classifier, detection rates reach 99.8 %, with not significant differences between the basic and the balanced classifier.

The false-positive rate of 15.2 % without classifier increases by 2 percentage points for the basic classifier, and by 0.7 pp. for the balanced classifier.

Overall it can be said that the classifiers' performance did not suffer from the increased amount of mislabeled training samples.

After 12500 incoming mails, enough training data could be collected to train the classifier..

Scenario 3

The third scenario again shows similar results. The detection rates of malicious mails improve from 81.5 %, 25.2 %, and 81.4 %, to 99.0 %, 99.8 %, and 99.7 % for the balanced classifier. The basic classifier achieves 0.1 percentage points more in these three cases.

The false-positive rate increases from 2.0 % without classifier to 4.2 % with the basic classifier, and 4.7 % with the balanced classifier.

Automatic labeling collected the necessary training samples after circa 20000 incoming mails.

5.4 Conclusions

Looking at the three scenarios, it can be said that the detection of malicious mails can be significantly improved by employing the classifier with automatic labeling. Even with mislabeled training data, the system is robust enough to still detect a large part of the previously undetected malicious mails.

However, the high detection rate comes at a price. In all scenarios, the false-positive rate increased by up to 3.1 percentage points. Since this is about the same amount that has been observed in Chapter 4, it is most likely unrelated to the automatic labeling.

No clear decision can be made about the question whether the classifier should use all available training data or if the training data should be balanced. The necessary amount of training data could be collected after 12500 to 20000 incoming mails, about two to four months of data in the dataset. This seems like an acceptable time span for preparation, during which the existing detection methods have to provide protection.

All things considered, the concept of automatic labeling of training data seems to be applicable without restrictions.

The classification system on the other hand has to be optimized further. While the detection rates are very promising, the false-positive rate is definitely too high to recommend this system as an additions to existing detection methods. There are possible ways to mitigate or improve this situation:

- **Warning instead of deletion**
  When the classifier labels an incoming mail as malicious, it doesn't have to be deleted. Instead, a warning could be shown to the user, for example as a prefix in the mail subject. Also possible would be a system that makes it's reaction dependent on the confidence value of the classifier. For example it could embed a warning in case of a low confidence value, or instantly delete if the confidence value is very high.

- **Raising the decision threshold**
  It is possible to raise the classifiers decision threshold. This means, only mails where the classifier's confidence value exceeds this threshold are being detected as malicious. This should lower the amount of false-positives as well as the amount true-positives. Since the classifier's detection rate is very high, it should be possible to find a more reasonable trade-off between true-positive and false-positive rate.

  However, this threshold should not be set manually since it has to fit the mails and the data that the classifier has to work with. Instead some automatic way has to be found to detect an increased amount of false-positives and correct the threshold correspondingly.

- **Class weights**
  Usually, classification algorithms have ways to define class weights. This means, that

errors made in regards to that class are considered more severe. This could be used to give benign mails more importance when defining the decision threshold.

- **Self-testing the classifier**
  The automatic labeling takes into account if a mail is a response to a previously sent mail. These response mails could be used to test the classifier. Instead of just using these responses as training, they could be given to the classifier. If the classifier detects them as malicious, the decision threshold can be adjusted to make sure these benign mails are not filtered.

- **Using the classifier to detect false-positives**
  The classification system could maybe be converted so that it does not detect malicious mails itself, but detects false-positives of existing detection systems.

- **Replacing other sources of false-positives**
  When eliminating all other sources of false-positives, for example the spam filter, the overall false-positive rate might be the same as when using conventional detection methods, while offering a better detection rate. This would require more testing under real-life conditions.

- **Implementing user feedback**
  If users can influence the classifier by correcting its errors, will have more training data in regards to those mails he struggles to correctly identify. This might reduce false-positive rates in the long run, especially if this feedback is given a higher weight.

6 Conclusion

This thesis examined the feasibility of detecting unsolicited mail based solely on structural traits, without taking the mails' contents into account. To represent these structural traits, the features as proposed by [GUSR18] have been adopted.

To evaluate this proposition, five classification algorithms have been selected: a Support Vector Machine, Logistic Regression, Random Forests, Naive Bayes, and k-Nearest Neighbors. For every classification algorithm, the best suited subset of features has been determined. Using this optimal featureset, and while looking at each classifiers' true-positive detection rate and false-positive error rate, each classifier has been analyzed in regards to the required amount of training data, the long-term stability and resilience against concept drift, and the computational complexity of the training and classification processes (Chapter 4).

During this evaluation process, the best results could be achieved with the Support Vector Machine, using all features. While the classification performance slowly increases with more training data, it could be shown that good results can already be achieved with as little as 2000 training samples. However, the experiments showed a constant decline in detection rate as well as an increase in false-positive rate, when the classifier has not been retrained regularly. This means that recent training data is needed at all times, and retraining of the classifier should be performed on a regular basis.

In conclusion, these results show that the approach of detecting unsolicited mail based on similarity in structural traits is possible.

Furthermore, this thesis also addresses the most significant issue, when applying such a learning based approach to real-life systems: the collection of the necessary training data. To solve this issue, an automatic labeling system has been proposed (Chapter 5).

For this labeling system, possible information sources have been identified. Using the previously evaluated classifier and these possible information sources, three scenarios have been determined to simulate whether training data can be collected in sufficient amounts.

These simulations showed that it indeed is possible to automatically collect training data in real-life scenarios, and that a classifier trained with automatically labeled data can contribute to the detection of unsolicited mails even if the automatic labeling system introduces mislabeled training data.

However, all these experiments showed one shortcoming of the approach: the rather high false-positive rate. In the simulations with automatic labeling, the false-positive rate increased by up to 3 percentage points. In the experiments with pre-labeled training data, the false-positive rate was at around 4 %. While some of the errors are due to false labels in the dataset, these values are too high for the intended use as an addition to existing detection methods like anti-virus or spam filters.

Therefore, propositions to solve this issue have been made. This includes direct measures to reduce the amount of false-positives like raising the classifier's decision threshold, as well as ideas to mitigate the effect of false-positives.

6.1 Future work

Reducing the false-positive rate should also be the starting point for further research. The high AUROC values in the conducted experiments indicate that a more favorable trade-off between true-positive and false-positive rate should be possible. The two obvious approaches to this issue are either directly raising the decision threshold of the classifier, or increasing the weight of false-positive errors in the classifier's error function.

Additionally, the classifier's decision confidence for false-positives in its current form could be examined further to answer the question if misclassifications result in confidence values close to the decision threshold. This information could be used to define staggered reactions, if a mail has been classified as malicious. A low confidence value could trigger a warning to the user, while a high confidence value would lead to a more decisive reaction, for example quarantining the mail.

An interesting addition to such a staggered reaction model would be to extend the classifier to not only differentiate between benign and malign, but to give probabilities to which subclass of unsolicited mail (*spam*, *phishing*, *malware*) a sample might belong.

Other future research could evaluate methods to implement some form of user feedback. Misclassifications detected by the end-user could be used to improve the accuracy of the classifier.

A Appendix

A.1 Evaluation

n	Featureset	SVM	Log. Reg.	Forest	Bayes	KNN
300	mail	0.961771	0.960180	0.957029	0.899578	0.907136
	mua	0.951028	0.950048	0.954872	0.906732	0.871504
	transport	0.970638	0.969394	0.960917	**0.945979**	0.839099
	mail + mua	0.966833	0.964306	0.962980	0.908056	0.897702
	mail + transport	**0.979981**	**0.978198**	**0.971232**	0.930886	0.915629
	mua + transport	0.977210	0.975828	0.969284	0.934360	0.894307
	all	0.979469	0.977594	0.970760	0.929516	**0.917044**
1000	mail	0.975965	0.972911	0.976315	0.900361	0.933484
	mua	0.965878	0.965257	0.974069	0.911206	0.910432
	transport	0.983452	0.983873	0.979675	**0.955057**	0.866896
	mail + mua	0.979086	0.977418	0.976759	0.915031	0.929042
	mail + transport	0.986180	**0.986169**	**0.986195**	0.936213	0.932259
	mua + transport	0.986353	0.983712	0.982401	0.945980	0.922674
	all	**0.988171**	0.985281	0.982434	0.935025	**0.938902**
5000	mail	0.984741	0.986778	0.986287	0.919437	0.958163
	mua	0.980706	0.984263	0.985503	0.915439	0.949456
	transport	0.989085	0.990807	0.987859	**0.988676**	0.919224
	mail + mua	0.988063	0.989618	0.990139	0.933209	0.959541
	mail + transport	0.989517	0.990643	0.991048	0.962000	0.955429
	mua + transport	0.990474	**0.992861**	**0.991625**	0.963299	0.954975
	all	**0.990870**	0.992091	0.991083	0.940204	**0.964773**
50000	mail	0.992377	0.993574	0.993111	0.902038	0.976828
	mua	0.987811	0.990590	0.991918	0.912478	0.969475
	transport	0.989956	0.990851	0.992114	**0.990235**	0.935607
	mail + mua	0.992874	0.993656	0.993175	0.901357	0.975185
	mail + transport	0.994996	0.995674	**0.995203**	0.975504	0.971769
	mua + transport	0.994076	0.994217	0.994629	0.986091	0.976253
	all	**0.995159**	**0.995973**	0.994876	0.982174	**0.979680**

Table A.1: Averaged AUROC results with reduced amount of training. (Best result per classifier and n in bold)

A.2 Automatic labeling

	Detection pipeline	Ham	Spam	Phishing	Malware
Scen. 1	No classifier	1.0	0.950244	0.500167	0.974469
	Basic classifier	0.968752	0.998681	0.999465	0.999452
	Balanced classifier	0.976504	0.998120	0.999247	0.999060
Scen. 2	No classifier	0.847667	0.952980	0.525814	0.974469
	Basic classifier	0.827434	0.999004	0.999398	0.999217
	Balanced classifier	0.841030	0.998377	0.999063	0.998434
Scen. 3	No classifier	0.979456	0.814684	0.256299	0.813768
	Basic classifier	0.958269	0.991558	0.998662	0.997416
	Balanced classifier	0.952526	0.990385	0.997792	0.996632

Table A.2: Detection rates during the automatic labeling experiments

List of Figures

Bibliography

[AFJ+15] Alqatawna, Ja'far ; Faris, Hossam ; Jaradat, Khalid ; Al-Zewairi, Malek ; Adwan, Omar: Improving Knowledge Based Spam Detection Methods: The Effect of Malicious Related Features in Imbalance Data Distribution. 8 (2015), 04, S. 118–129

[Ami10] Amin, Rohan M.: *Detecting Targeted Malicious Email Through Supervised Classification of Persistent Threat and Recipient Oriented Features.* Washington, DC, USA, Diss., 2010. – AAI3428188

[AON+15] Ali, Siti-Hajar-Aminah ; Ozawa, Seiichi ; Nakazato, Junji ; Ban, Tao ; Shimamura, Jumpei: An Online Malicious Spam Email Detection System Using Resource Allocating Network with Locality Sensitive Hashing. In: *Journal of Intelligent Learning Systems and Applications* 7 (2015), Nr. 2, S. 42–57

[ARD12] Amin, R. ; Ryan, J. ; Dorp, J. van: Detecting Targeted Malicious Email. In: *IEEE Security Privacy* 10 (2012), May, Nr. 3, S. 64–71. `http://dx.doi.org/10.1109/MSP.2011.154`. – DOI 10.1109/MSP.2011.154. – ISSN 1540–7993

[BFOS84] Breiman, Leo ; Friedman, Jerome ; Olshen, R.A. ; Stone, Charles J.: *Classification and regression trees.* 1984. – ISBN 978–0–412–04841–8

[BGV92] Boser, Bernhard E. ; Guyon, Isabelle M. ; Vapnik, Vladimir N.: A training algorithm for optimal margin classifiers. In: *Proceedings of the fifth annual workshop on Computational learning theory* ACM, 1992, S. 144–152

[BM00] Brutlag, Jake D. ; Meek, Christopher: Challenges of the email domain for text classification. In: *ICML*, 2000, S. 103–110

[Bre01] Breiman, Leo: Random forests. In: *Machine learning* 45 (2001), Nr. 1, S. 5–32

[CF18] Chio, Clarence ; Freeman, David: *Machine Learning and Security: Protecting Systems with Data and Algorithms.* "O'Reilly Media, Inc.", 2018

[CKP03] Clark, James ; Koprinska, Irena ; Poon, Josiah: A neural network based approach to automated e-mail classification. In: *Web Intelligence, 2003. WI 2003. Proceedings. IEEE/WIC International Conference on* IEEE, 2003, S. 702–705

[Cox58] Cox, David R.: The regression analysis of binary sequences. In: *Journal of the Royal Statistical Society. Series B (Methodological)* (1958), S. 215–242

[CV95] Cortes, Corinna ; Vapnik, Vladimir: Support-vector networks. In: *Machine learning* 20 (1995), Nr. 3, S. 273–297

[DDPS04] Damiani, Ernesto ; De Capitani di Vimercati, Sabrina ; Paraboschi, Stefano ; Samarati, Pierangela: An Open Digest-based Technique for Spam Detection. In: *in Proceedings of the 2004 International Workshop on Security in Parallel and Distributed Systems*, 2004, S. 15–17

[DKCE+16] Duman, S. ; Kalkan-Cakmakci, K. ; Egele, M. ; Robertson, W. ; Kirda, E.: EmailProfiler: Spearphishing Filtering with Header and Stylometric Features of Emails. In: *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)* Bd. 1, 2016, S. 408–416

[Faw06] Fawcett, Tom: An introduction to ROC analysis. In: *Pattern recognition letters* 27 (2006), Nr. 8, S. 861–874

[FLM+15] Foster, Ian D. ; Larson, Jon ; Masich, Max ; Snoeren, Alex C. ; Savage, Stefan ; Levchenko, Kirill: Security by Any Other Name: On the Effectiveness of Provider Based Email Security. In: *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*. New York, NY, USA : ACM, 2015 (CCS '15). – ISBN 978–1–4503–3832–5, 450–464

[FST07] Fette, Ian ; Sadeh, Norman ; Tomasic, Anthony: Learning to Detect Phishing Emails. In: *Proceedings of the 16th International Conference on World Wide Web*. New York, NY, USA : ACM, 2007 (WWW '07). – ISBN 978–1–59593–654–7, 649–656

[GC09] Guzella, Thiago S. ; Caminhas, Walmir M.: A review of machine learning approaches to spam filtering. In: *Expert Systems with Applications* 36 (2009), Nr. 7, S. 10206–10222

[GMCR04] Gama, Joao ; Medas, Pedro ; Castillo, Gladys ; Rodrigues, Pedro: Learning with drift detection. In: *Brazilian symposium on artificial intelligence* Springer, 2004, S. 286–295

[GUSR18] Gascon, Hugo ; Ullrich, Steffen ; Stritter, Benjamin ; Rieck, Konrad: Reading Between the Lines: Content-Agnostic Detection of Spear-Phishing Emails. In: *International Symposium on Research in Attacks, Intrusions, and Defenses* Springer, 2018, S. 69–91

[Ho95] Ho, Tin K.: Random decision forests. In: *Document analysis and recognition, 1995., proceedings of the third international conference on* Bd. 1 IEEE, 1995, S. 278–282

[Ho98] Ho, Tin K.: The random subspace method for constructing decision forests. In: *IEEE transactions on pattern analysis and machine intelligence* 20 (1998), Nr. 8, S. 832–844

[LF07] Leiba, Barry ; Fenton, Jim: DomainKeys Identified Mail (DKIM): Using Digital Signatures for Domain Verification. In: *CEAS*, 2007

[Mit97] Mitchell, Thomas M.: *Machine Learning*. 1. New York, NY, USA : McGraw-Hill, Inc., 1997. – ISBN 0070428077, 9780070428072

[MNS+05] Martin, Steve ; Nelson, Blaine ; Sewani, Anil ; Chen, Karl ; Joseph, Anthony D.: Analyzing Behavioral Features for Email Classification. In: *CEAS*, 2005

[Mur12] Murphy, Kevin P.: *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. – ISBN 0262018020, 9780262018029

[OPB12] Oshiro, Thais M. ; Perez, Pedro S. ; Baranauskas, José A.: How many trees in a random forest? In: *International Workshop on Machine Learning and Data Mining in Pattern Recognition* Springer, 2012, S. 154–168

[Pow11] Powers, David M.: Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. In: *Journal of Machine Learning Technologies*, Bioinfo Publications, 2011, S. 37–63

[RN09] Russell, Stuart ; Norvig, Peter: *Artificial Intelligence: A Modern Approach*. 3. Upper Saddle River, NJ, USA : Prentice Hall Press, 2009. – ISBN 0136042597, 9780136042594

[Sal97] Salzberg, Steven L.: On comparing classifiers: Pitfalls to avoid and a recommended approach. In: *Data mining and knowledge discovery* 1 (1997), Nr. 3, S. 317–328

[SCT04] Stuart, Ian ; Cha, Sung-Hyuk ; Tappert, Charles: A neural network classifier for junk e-mail. In: *International Workshop on Document Analysis Systems* Springer, 2004, S. 442–450

[SDHH98] Sahami, Mehran ; Dumais, Susan ; Heckerman, David ; Horvitz, Eric: A Bayesian Approach to Filtering Junk E-Mail. In: *Learning for Text Categorization: Papers from the 1998 Workshop*. Madison, Wisconsin : AAAI Technical Report WS-98-05, 1998

[Seb62] Sebestyen, George S.: Decision-making processes in pattern recognition. (1962)

[Sha48] Shannon, C. E.: A mathematical theory of communication. In: *The Bell System Technical Journal* 27 (1948), July, Nr. 3, S. 379–423. `http://dx.doi.org/10.1002/j.1538-7305.1948.tb01338.x`. – DOI 10.1002/j.1538–7305.1948.tb01338.x. – ISSN 0005–8580

[SJJ16] Siadati, Hossein ; Jafarikhah, Sima ; Jakobsson, Markus: Traditional countermeasures to unwanted email. In: *Understanding social engineering based scams*. Springer, 2016, S. 51–62

[Sym17] Symantec Corporation: Internet Security Threat Report - Volume 22, April 2017. (2017)

[Tsy04] Tsymbal, Alexey: The problem of concept drift: definitions and related work. In: *Computer Science Department, Trinity College Dublin* 106 (2004), Nr. 2