Abstract

Data are essential to every decision support system and as more systems become capable of automated decision making, data quality is critical. Data are of good quality if conformant to some standard (verified) and of some semantic meaning (valid). While verification is predominant, validation is not often used because of its complexity. By presenting why a full Verification & Validation (V&V) process is suitable for data quality improvement and by applying this concept to email address quality assurance, it will be demonstrated that email deliverability of Customer Relationship Management or Email Marketing systems can be significantly improved.

Standard email address verification techniques based on regular expression are extended by complex verification techniques that verify all email address parts in multiple steps. Validation techniques are extended by a Support Vector Machine classification of bounce failures, and an n-gram based classification of email address local-part. All traditional and new techniques are integrated into a V&V framework which can lead to the design and development of a component for continuous email address quality assurance.
Acknowledgment

I would like to thank Keith Majkut for all of his support and especially for his help with the annotation of the delivery failures.
Contents

Chapter 1 Introduction ............................................................................................. 1

Chapter 2 Email address quality and email communication ................................. 4
  Quality definition .................................................................................................. 4
  Data quality .......................................................................................................... 4
  Dimensions of data quality .................................................................................. 5
  Quality of a data process ...................................................................................... 8
  Continuous quality improvement ......................................................................... 8
  Data mining and data quality .............................................................................. 9
  Email address quality ......................................................................................... 9
  History of email address ..................................................................................... 10
  Project objectives ............................................................................................... 14

Chapter 3 Methodology ......................................................................................... 15
  Verification & Validation .................................................................................... 15
  Data verification .................................................................................................. 15
  Data validation ................................................................................................... 16
  Email address specification ............................................................................... 16
  Verification & Validation of an email address ..................................................... 18
  Verification & Validation module design ............................................................. 19

Chapter 4 Verification techniques ....................................................................... 20
  Verification of an email address ......................................................................... 20
  Email address structure refinement ..................................................................... 20
  Verification techniques - TLD ............................................................................ 24
  Verification techniques - Domain ....................................................................... 24
  Verification techniques - Local-part ................................................................... 25
  Verification techniques - Email address ............................................................. 25
  Email Verifier .................................................................................................... 26

Chapter 5 Validation – Bounce classification ..................................................... 27
  Evaluation of validation techniques .................................................................... 27
  Collecting bounce data ...................................................................................... 29
  Bounce data ....................................................................................................... 30
  Structures to process bounces - Part 1 ................................................................. 33
  Bounces classes ................................................................................................. 36
  Bounce Failure Annotation ................................................................................ 38
  Text processing .................................................................................................. 42
  Term document matrix (TDM) .......................................................................... 45
  Latent semantic analysis (LSA) .......................................................................... 46
  Classifying algorithms ....................................................................................... 47
Chapter 1
Introduction

Data quality
In the past, simply knowing something that others wanted to know was an advantage. And although the desire to gain an informational advantage persists, the way that we treat the information is changing. Some think [20] that historians may classify our time as "The Golden Age of Information", but we shall not indulge in that much wishful thinking until we are there. A great deal of effort is still needed to automate the knowledge discovery that will start such a flourishing time. So far we are only adept at automating observations and data collection. According to [21], "Every two days now we create as much information as we did from the dawn of civilization up until 2003". So far, we are still unskilled at transforming data into useful information, nor can we build machines with faculty to reason.

With respect as to what we are able to do with data, it is nothing in comparison to what some seers predict [21]. Our data repositories are now similar to large topsy-turvy scrap yards, where it is difficult to differentiate what can be useful and what is waste. This analogy may sound a bit pessimistic, but it at least assures we will have a sufficient supply of raw material to process in the future. And as the invention of effective coal transformation into steam power could not have happened without a large quantity of plant matter dumped onto one place, it is hoped that in the future some invention of information processing will truly trigger our Golden Age. I believe that this data collection epoch will lay the foundation for the true "The Golden Age of Information".

Why data quality is important
Before we can start developing machines that could truly use the information potential of data, it is important to learn how to separate the data waste from the high quality pieces. Clearly, some data are more important than other data and their importance is related to their usage. As the quality of data is determined by its potential to fulfill our goals, the data quality aspects that need to be evaluated will only emerge if we review them from the perspective of the objective.

Quality of a final product depends on the quality of its components [30], and there is no way to achieve good quality components without using the right material and the right process. It is reasonable to assume that any information or knowledge based on bad quality data or a bad data process will not be of a high quality. Without good quality of data it is not possible neither to build an automated reasoning systems nor to make it reason well.

As data complexity advances, it is essential to take a top-down look at the data quality, and link it to the higher-level corporate goals. Those who can understand how data quality affects their business goals will advance.

The current deficiency of data quality could be minimized if the data processing industry, sometimes called business intelligence, would pay more attention to how quality is addressed in other engineering disciplines. A similar knowledge transfer happened in Software Engineering.
and the maturity of quality techniques is visible when comparing the amount of publications with “software quality” and “data quality” terms over five decades.

![Figure 1.1 Occurrence of software quality and data quality n-grams in publications](http://ngrams.googlelabs.com)

**Objectives**

The ambition is to elaborate the basic principles of quality assurance and improvements used in the mass production and software engineering industries that could be adapted for data purposes. The suitable data entity to demonstrate the transition of such techniques is the email address. It was chosen because the impact of its bad quality is easily measured (low message delivery, low response rate, etc.) and its purpose is well known without any deep explanations. Understanding the quality of email address is also important because these data are traded as any other commodity, and their price relates to their quality.

Although the quality attributes of email address are well defined, there are not many validation techniques that could be used to measure their quality. This is why most systems only verify an email address but do not validate it. My intention is to use data mining algorithms to develop validation techniques that will be able to evaluate email address quality prior and post mailing.

**Outline**

The first chapter of this thesis establishes a view on how data quality is defined and measured. Various approaches that define quality attributes of data are discussed. As low level data quality techniques are proved to be insufficient, the need for new approaches is needed and their requirements described. As the objective of this work is to develop new quality assurance techniques, it is desirable to test such on a real entity, so basic background about email messaging and email address evolution is given.

In the second chapter, after clarifying what is expected from a data quality assurance system, a Verification & Validation (V & V) process adapted from Software Engineering is presented. It is also suggested how V&V techniques could be integrated into automated systems for continuous quality assurance and requirements of V&V techniques used for email address quality.

The other three chapters are devoted to concrete V&V techniques that can be used for email address quality assurance. Verification techniques, which are methods of specification conformance assurance, are presented. As those require unambiguous specification, email address and messaging standards are elaborated and transformed into a new specification.
A new approach is suggested for email address validation and monitoring of processes that depend on its quality. As it might be difficult to transform high level needs into lower level requirements, the entire problem is hidden behind a data mining model, which predicts the quality of an individual email address. Two models are developed, one to predict the email address quality based on historical bounce reasons, and the second to calculate the email address quality based on prior probabilities of failure distributions.

Along both sections, comments are made for each particular technique, and various implementation suggestions are made. This thesis shall provide a sufficient base for development of a Verification & Validation component that could be used for continuous quality assurance of email address quality. It shall be as well an inspiration for development of techniques that validate other data points.

In the final chapter, the entire contribution is reviewed and some additional ideas for future work and improvements are suggested.
Chapter 2
Email address quality and email communication

Quality definition
Quality, as a generic term, is defined by various standards. Unfortunately, each definition is unique and encompasses the subjective perception of the individual defining it. For instance ISO 8042 defines quality as

ISO 8042: "The totality of features and characteristics of a product, process or service that bear on its ability to satisfy stated or implied needs."

ISO 9001: "Quality is a degree to which a set of inherent characteristics fulfills requirements."

Data quality
To understand data quality it is beneficial to look at some real world facts and how they are represented in data. The world around us is complex, and it is not possible to capture its state entirely, thus anything we capture from the real world is not complete, and is missing something. How completely or correctly the real world entity is described is part of the quality. Quality is hidden in sampling frequency for audio, in image resolutions etc. It is however a false assumption that a higher sampling frequency or a higher resolution always equals higher quality. Although it is possible to resize a large image, a lower resolution image is better suited to meet the time to load requirement of a web page, but not a large format printout.

The important aspect here is that image requirements can be transformed into quality measures and in this case time to load is related to image resolution. Once we state our needs we shall convert them into quality measures and allow anyone to objectively evaluate it. In this case it is very simple to measure the image resolution and to define the threshold when to reject or accept the image.

So far just one dimension, resolution, of image quality was reviewed, but to complete the full assessment we must evaluate other attributes as well. Assume for now, that the picture is meeting our expectation in terms of the image resolution and is suitable for the web page. If the web page is documenting the history of a city, then there is no value to see people on it. The quality of the picture is only assessed by the amount of historical buildings depicted on it. The quality threshold is very simple - if a building is on the image, then it is of high quality and if no building, then it is of zero quality.

A quality defined with a measure and decision threshold is needed. Of course, we have a precise rule, but is it always possible to measure it precisely? For example, someone might argue that there is a building on the picture while someone else might not see it (too small or fuzzy). In the end, even a precise rule cannot assure sufficient quality evaluation.
Assume further that the majority of people agree that a building is in the picture, so the next step is to evaluate this information. For instance, if the picture depicts a destroyed building, it would be of high quality, but if it depicts a building which did not change over time, the value would be low (equal to the value of a contemporary picture). In any case we need to identify the building and compare it with reality, which might be simpler if we know the picture location, and the time it was taken. We can as well visit the location or use some other knowledge, which resides outside of the picture. If we identify the city, our knowledge of that area can help us to infer its state. For example, if the picture was taken in Dresden in 1930, it is likely the building was demolished in World War II. The quality of the picture in this case depends on information that is external to it and our ability to use it. The quality measure and even the threshold are not clear, so only those who can identify the building and have enough knowledge about it, can determine the quality of the picture, for the above stated need.

The last example I would like to mention in this introduction to data quality is related to time. On the picture of Postplatz, Dresden in 1930 [33] we see that Robert C. Kunde has a shop for steel ware, and it is right at the stop of the tram line number two. This information might be of some value for a person who wanted to buy a knife 80 years ago, but not now since the building was destroyed. Although data and its quality is the same, the information aged and lost its quality. So far we described a few obstacles that one could face when building quality measures for evaluation of data or information. In our example, a trained human will be able to achieve this task to a certain degree, but for a machine it is a challenge.

The title of this section is called data quality, however in some places we jumped from data to information or even knowledge. These three levels are distinct, however related [6]. The ability to extract information and knowledge from data is indeed a property of data. If we are not able to extract the information or knowledge from the data, then such data are for us of low quality. It of course does not necessarily mean the information is not in the data, and it is more related to the process of extraction, which we will cover in more detail later.

**Dimensions of data quality**

Data, information or knowledge quality can be viewed from various perspectives. These perspectives shall give us a coherent understanding of the product and allow us to compare it with other products. A perspective can be defined by a single measurable attribute or by a complex set of attributes. If we omit the complexity of the inner structure of the perspective, we can simply see it as a vector. These vectors form a multi-dimensional evaluation space which comprises all possible perspectives. The dimensionality of the space depends on the complexity of the product of which quality we aim to evaluate and on the amount of all possible subjective perspectives.

Any concrete evaluation of a product is executed on a subspace of all possible dimensions, and this subspace is often unique and matches the needs of a single person doing the evaluation. It is impossible to objectively agree on the quality of a product, if different subspaces are used. To assure objectivity the same dimensions will need to be used, which can be seen within the mass production industry. One reason why we are able to evaluate the quality of a complex industrial product is because they are made of components and each of these components is formed of
other components which can be measured and tested. Unfortunately, reductionism is hardly applicable to information or knowledge, because of its holistic complexity.

A goal driven approach for data or information evaluation is difficult because it requires a clear understanding of the product and can be used to evaluate only a quality of a single product. If our goal was to evaluate a larger set of data usable for various purposes, we would need to have the whole set of all dimensions and evaluate them all.

Therefore, the idea is to find important dimensions, which is challenging even when evaluating more tangible products than information or knowledge. A software product is also immaterial and in many cases unique, so we still do not have a perfect solution how to deal with its complexity. For example, when evaluating software quality, we are still not able to tell when to stop testing [22],[23] because the amount of dimensions that need to be evaluated is huge and we constantly deal with the risk of missing some requirement.

One approach to minimize the risk is to use a check list of all possible quality attributes. For instance, [7],[8] we define generally applicable concepts to evaluate or at least to gain the understanding of data quality. It is important to note that these are defining quality attributes of data quality, not the specific attributes of one single data entity. For example, [7] is defining ten attributes (dimensions) of data quality.

<table>
<thead>
<tr>
<th>dimension</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accessibility</td>
<td>Is it available or reachable for use?</td>
</tr>
<tr>
<td>Accuracy</td>
<td>Degree to which it reflects the underlying reality</td>
</tr>
<tr>
<td>Coherence</td>
<td>Degree to which it reflects the underlying reality</td>
</tr>
<tr>
<td>Compatibility</td>
<td>Could it be combined with other information and extended?</td>
</tr>
<tr>
<td>Completeness</td>
<td>Is it covering all needs?</td>
</tr>
<tr>
<td>Format</td>
<td>The method it is presented</td>
</tr>
<tr>
<td>Relevance</td>
<td>Is it addressing the customer needs?</td>
</tr>
<tr>
<td>Security</td>
<td>Is it safe from being destroyed or stolen?</td>
</tr>
<tr>
<td>Timeliness</td>
<td>Is it up to date?</td>
</tr>
<tr>
<td>Validity</td>
<td>Is resultant of all other dimensions?</td>
</tr>
</tbody>
</table>

Figure 2.1 Quality dimensions taken from [7]

In [8], the attributes are slightly different, and others who wanted to create something generic end up with their specific lists. This is not surprising, because each author possibly came from a different background or had a different need. It is just another fact that quality is subjective and a universal list does not exist.

For instance, authors who are covering areas of design and implementation of quality assurance (database design, data warehouses, ETL ...) are more interested in data quality measurements which can be implemented at the code level. Only some of the generic attributes mentioned earlier can be traced to that level. In [5],[10] data quality measures are even rejective. They look at the data quality from the failure perspective (How many times data was rejected, how many times the load failed ...). More interestingly, the data quality in operational systems is calculated on whole sets of data, as it is believed that a single datum can be either correct or incorrect (binary quality), and quality of the whole can be defined as the ratio between incorrect and correct values. On such lists, quality attributes are reduced and have more exact interpretation.
Engineers need to produce a working product and face exact constraints (textual value cannot be stored in a numeric column), so this is why only hard measures with exact outcomes are implemented. Deterministic hard techniques are superseding other soft data quality metrics. Systems where hard techniques predominate satisfy needs of maximally one single subject. A debatable example can be found in enterprise solutions that blindly adopt a now very popular design principle called "Single Source of Truth. Hard techniques allow only one objective interpretation, and should be implemented only on the lowest level of the information. If used on a higher level, it will inevitably cause collisions because different people do have different needs. To allow subjective quality in enterprise solutions, we either need to weaken the assessment criteria or separate the higher level information data into multiple places [16].

<table>
<thead>
<tr>
<th>dimension</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete</td>
<td>Individuals are not null, aggregates are not losing anything</td>
</tr>
<tr>
<td>Consistent</td>
<td>One constant notation is used</td>
</tr>
<tr>
<td>Correct</td>
<td>Is the associated object described truthfully and faithfully?</td>
</tr>
<tr>
<td>Unambiguous</td>
<td>Does it have only one meaning?</td>
</tr>
</tbody>
</table>

Figure 2.2 Quality dimensions taken from [10]

The problem with hard measures can be demonstrated on an example taken from [9], where accuracy refers to whether the data values stored for an object are the correct values. The author states that if his birth date is December 13, 1941, and in the database there is 12/14/1941 (US format), then it is inaccurate because it is a wrong value. However when somebody will ask about the year when the author was born, then 1941 is correct. There is a tendency to purge inaccurate information, and according to the first rule the birth date is inaccurate, so a system that maximizes the subjective needs must manage the data differently.

One solution is not to store the birth date as a single value, because a single entity can be either correct or incorrect. Anybody who is interested in the year of the author's birth date (which is impossible to automatically assert) will not use the day part which can then be purged. We are able to meet these two needs at the expense of the system complexity. Unfortunately such solution is not stable if the needs change. If we insist on having only one objective measure, then it is better to build some evaluation model, which combines all subjective needs (for instance in this case 50% of its intended usage is correct) or combination of quality measure of each component (the date component (day) is not correct, so the date it is accurate to 2/3 of its maximum accuracy).

<table>
<thead>
<tr>
<th>independent point of view (objective)</th>
<th>enterprise integration point of view (subjective)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comprehensiveness</td>
<td>Singularity</td>
</tr>
<tr>
<td>Accuracy</td>
<td>Correspondency</td>
</tr>
<tr>
<td>Completeness</td>
<td>Reusability</td>
</tr>
<tr>
<td>Up to date</td>
<td>...</td>
</tr>
<tr>
<td>Applicability</td>
<td>...</td>
</tr>
</tbody>
</table>

Figure 2.3 Grouping quality dimensions into objective and subjective buckets taken from [19]

It is dangerous to measure quality on isolated dimensions, because the quality as a whole as well as the individual quality attributes depend on other quality attributes. As stated in [7], validity is resultant of all other dimensions, so if one essential dimension is not meeting its requirements, it does not matter how good the others are, since the whole validity and hence the quality is bad.
Validity in this sense is identical to quality, which is helpful, however the portion of subjectivity is still unresolved and a different concept is needed.

In [19] it is suggested to add two more data quality dimensions - depth and width. Depth is identical to what we called objective quality, and width reflects the subjective needs. Quality attributes are then separated into these two groups.

Quality of a data process
Another important view on data quality is through the process perspective. Product quality depends on process quality [30],[31],[32], so if a camera "process" produces lower resolution images than what is needed, we can either change the configuration of the camera (adjust the process), or replace it. In this case without changing the process we could never reach the desired quality. This is why it is suitable to define some of the data quality through the process quality attributes. It is important to know the expected quality of data before we start creating the system that will produce it. This is very often not the case, and data quality is reviewed and addressed after the system is ready. It should be noted that it is not easy to foresee data issues before the data is available, which is not an excuse for trying to address them. Data production is closer to assembly line production, than to software production. And although there is sufficient literature covering product quality and process quality (among many [17][18]), materials for data process quality are rare [31].

Continuous quality improvement
In manufacturing or software engineering there is a focus on constant improvement of quality. The driving factor is consumer unhappiness with the current quality. In over fifty years of quality assurance development in the manufacturing industry, many methodologies were implemented [18],[30],[31]. Unfortunately, there is nothing mature yet in the data quality assurance domain and even the founder of Total Information Quality Management (TIQM), nearly 60 years later discovered that there is a potential to change his methodology with concepts from other industries, "I have made some changes, mostly how to use proven techniques from Manufacturing Quality systems..." [35].

As we are now searching for inspiration in what others are doing, it’s beneficial to quickly summarize the ideas from the manufacturing industry, which were ported into software engineering.

PDCA cycle
The basic model used in production industry is Plan, Do, Check, Act (PDCA), sometimes called Deming or Shewhart Cycle [30],[31]. There are many variations of it, but the basic version has four stages.

- **Plan** - Recognize the opportunity and plan the change. In this first stage of the model, we need to define the goal and develop a plan to get there. An important part of this process is to define how to test the deviation from the target.
- **Do** - Once done with the planning, change the process.
- **Check** - Analyze the results and identify what you have learned. After the change is done, we start measuring how the change affected the outcome. Are we near to the target and if not, how far away?
- **Act** - Change the whole process to yield better results. Act on the change and if successful, go step by step to further increase our quality demands. If we are still not near the target, then change the process again.

![Figure 2.4 Activity model of the PDCA cycle](image)

This improvement model can be easily ported into a data quality improvement process and because the idea is simple, some often follow this principle. It is possible to identify its steps on data management systems, where the lowest level of quality is constantly improved. It is important not confuse this proactive approach, with a reactive Fail-Do model, where we act after the quality issue is inhibiting data processing operations.

We learned how to maintain the lowest level of data quality, mostly because we were forced to do it by the constraints of the database systems. Unfortunately, many of those constraints are just simple syntactical or lookup value tests, which is not enough for the higher level of data quality requirements. To improve the higher level of data quality, we need to better understand how the data are used. In the case of an email address, it is not enough to check if it is syntactically correct. This will not guarantee we could deliver a message to it, which is the real need of email address data. To evaluate such level of quality better methods are needed.

**Data mining and data quality**

It is easier to get support for data quality improvement projects, if we are able to demonstrate how these projects can contribute to the corporate goals. It is almost impossible for executives to trace the impact on low level data dimensions from a detailed list of issues. Ideas on how to demonstrate the effect of many attributes to a single question is the domain of data mining. Instead of showing a Customer Relationship Manager low level data issues of an email address, it is more suitable to transform the problem into a single question, whose answer could be calculated by a statistical model. Such a question could be "If we email this amount of users, how many will receive the message?"

**Email address quality**

At this stage we should already have a solid understanding of what data quality means, what are the quality attributes and how, quality can be improved. It was also said that data quality in
current systems is not measured sufficiently and improvements in quality can be hardy made without new, more sophisticated, measures. As the goal is to develop such quality measures, it is suitable to select a data entity where this approach can be demonstrated.

A suitable data entity is the email address. It is an important asset for companies who run businesses in the information society. It is traded as commodity, it can be monetized, its structure is formalized, humans can create it, machines use it, it is bounded to a real object (person) and is changing over time. Although email address is widely familiar and its purpose does not require much explanation, its quality is not deeply understood and it is not trivial to assess or improve it.

If an email address of low quality is used than we can immediately feel the impact, and thus any effort to improve its quality might get better support in the organization than any other data entity. There are many ways how an email address can be used, and its usage is as well restricted or limited by legislation.

What are quality attributes of an email address? Again, this depends on what are our needs. Most people would probably say it is the deliverability or the possibility to reach the intended owner of the email address. Others might say that to reach the owner is not enough and it is the action people take after receiving the message. Email marketers might be interested in the open or click response to sent emails, while sales people are interested in the value of purchased goods. System administrators consider a bad quality email address as one which causes server issues, while database administrators consider one which throws an exception on update or insert. A data steward will classify an email address as bad quality simply if it does not match their metadata standards.

Whatever the consumer needs, we will need to define them, because without them, it would not be possible to build a system that will assure its quality.

**History of email address**

Messaging protocols and email address or mailbox structures evolved during the last 40 years and are now an essential part of the Internet infrastructure. Although there were standards that defined messaging frameworks before the Internet was widespread used, like RFC 561 (1973), RFC 680 (1975) or RFC 733 (1977) the first widely accepted “Internet” standard of the Internet text messages was RFC 822 (1982). This standard was factored into many system designs, and was used for nearly 20 years. In the last decade, while the Internet flourished, email communications became widely used. The complexity and ambiguity of email address definition caused a standardization schism. As RFC 822 was generally obeyed, specific interpretation or proprietary refinements were developed. RFC 2822 (2001) reflected some of those, and more are addressed in the currently valid standard RFC 5322 (2008).

As an email address is just a static locator of a mailbox, there are protocols that define how it should be used and how messages shall be routed. Analogically to the email address standards as well protocols were specified RFC 780, RFC 788, RFC 821, RFC 2821, and the latest RFC 5321. Although there is now one predominant Internet mail protocol SMTP (Simple Mail Transfer Protocol) it does not assure that an email transfer is a simple and error-less task.
Email communication
Before we can start building a system for automated email address quality assurance it is important to understand how email exchange communication and delivery failures work.

A user who wants to send a message uses his local machine client or a web client connected to a Mail Transfer Agent (MTA). Once a message is submitted for processing the MTA server is looking for a Mail Exchanger (MX) record to route the message. The MTA server establishes a connection to the appropriate server using SMTP and sends the message. An example of a single tier configuration is depicted in Figure 2.5.

More often emails do not reach the final destination directly and are routed through other intermediate SMTP servers Figure 2.6. Such communication is called email relay. A Mail Submission Agent (MSA), which is the internal equivalent of an MTA, may be used as well. The company “example.com” is using an MSA “one.example.com” for its internal email and another MTA “bulk.example.com” for its external email. Clients within the company connect to the MSA, which routes message outside of the company through the MTA.

Failures
In the simple case when the SMTP client fails to deliver a message it is their responsibility to decide what to do. It can either inform the sender “jack” or try again. In the relay scenario or in the case when the destination “foo.bar” server is not able to detect the problem in the transaction with the sender, a different approach needs to be taken. The destination SMTP server or the relay
server sends a failure message back, in our case to "one.example.com", to inform "jack" that some problem occurred. Such a message is called a Non Delivery Report (bounce).

**Bounces - validation of email address**

RFC 5321 (section 6.2) permits, under certain conditions, to drop (ignore/lose) the bounce messages, so the server is not vulnerable to an address book or spam attack (sending messages to random mailboxes and hoping to see a match). According to the test conducted in [15], it is not unusual when bounce messages are lost, even under situations that cannot be classified as an attack or threatening the stability or security of the SMTP servers. It is even proved that 5% of the servers are configured to never send a bounce back, while some respond sporadically. Moreover, 73% of the tested servers always responded. It is difficult to attribute the sporadic response of some servers. Further, it was noted [15] that in most cases the delay between the time when an email was sent and the bounce response was collected was in minutes, but in a small amount of cases it was in days and the largest delay was 50 days. It is important to factor the possibility of not receiving the bounces or to receiving them with some delay into the methods that will determine the quality of email addresses.

**Failure codes**

The idea of RFC 5321 and its predecessors is to use three digit reply codes to describe the state of the SMTP transaction. If there is a problem, servers respond with a failure code. To help resolve the problem, either automatically by the server or later manually by an administrator, the standard hierarchy organizes the failures into certain groups and uses these codes to identify them.

As stated in (section 4.2.1) "An unsophisticated SMTP client, or one that receives an unexpected code, will be able to determine its next action (proceed as planned, redo, retrench, etc.) by examining this first digit. An SMTP client that wants to know approximately what kind of error occurred (e.g., mail system error, command syntax error) might examine the second digit. The third digit and any supplemental information that may be present is reserved for the finest gradation of information."

It must be noted that, besides unsophisticated clients, there are also unsophisticated servers which very often report misleading failure codes. One attempt to resolve this deficiency was by creating new extended failure code structures. For instance RFC 3463 (Enhanced Mail System Status Codes) defines how such code should be structured. The analysis proved that this is even less respected and it is not even possible to have a registry of all the possible failures and their corresponding failure codes. In addition, technology evolves further and new problems occur. For instance spam messages did not even have their own failure in RFC 822.

**RFC5321 suggestions**
- 550 That is a user name, not a mailing list
- 550 Mailbox not found
- 550 Access Denied to You

**RFC3463 suggestions**
- 5.1.1 Bad destination mailbox address
- 5.1.2 Bad destination system address
- 5.1.3 Bad destination mailbox address syntax
Failure description

Fortunately, the SMTP protocol addresses the shortcoming of the failure status codes by extending them with a text description. In rare cases, some servers are not verbose and provide only the status code. In general, the text description contains the most reliable information about the failure, and it is useful for the human administrator who needs to understand its cause.

Bounce processing

For CRM (Customer Relationship Management) or email marketing systems, where email communication is one of the critical functionalities, bounce management can help prevent mailing invalid email addresses. This is desired if the goal is to keep a good reputation and delivery rate of the mailing servers.

There are various approaches on bounce processing to assure it will not occur again. The simplest idea is to remove the email address from future mailings after a bounce is observed. Unfortunately, any such concept assumes that the problem is solely on the email address level, and disabling it will solve the bounce issue. This is a false assumption, since an improperly configured sending server or a server with a bad reputation can also trigger bounces, and might falsely disable many email addresses. Any system that determines the quality of email addresses solely on a bounce count not only generates false positives but it oversimplifies the underlying problem, as it tries to tackle it with a binary measure.

In marketing literature bounces are usually classified into two groups, hard bounces and soft bounces [13],[24]. The assumption is that after a hard bounce is received, there is no chance to deliver any new message to the same email address (no quality), where after the soft bounce some chance remains (low quality). Although this concept is interesting and supports the idea of defining the quality on the level of business needs, its implementation relies on the failure codes which are not reliable. RFC3463, RFC5321 and its predecessors, classify failures into two categories as transient or permanent. In the failure below you can see that “permanent failures” (those starting with 5) do not always describe a “hard bouncing” email address.

550 Service unavailable; Client host [xxx.xxx.xxx.xxx] blocked using dnsbl.njabl.org; spam source
550 MAILBOX NOT FOUND 550 <john@foo.bar>... User unknown
550 5.7.2 This smells like Spam
550 <john@foo.de>. Benutzer hat zu viele Mails auf dem Server.
554 Delivery error Sorry your message to joe@foo.bar cannot be delivered. This account has been disabled or discontinued [#102]

Using common sense we can see that the probability of a future failure depends on the bounce type. It is reasonable to assume that after a soft bounce the probability is lower in contrast to the hard bounce. Further, we must not classify a hard bounce failure caused by the sender or the message, as these are not related to the email address and if corrected can result in the message delivery.

Anti bounce assurance shall disable the bouncing email address, but the question is when to do it. A simple approach might be to do it after three bounces or after a combination of hard and soft bounces. In [24], a more relaxed disabling schema is suggested (five soft bounces). It is interesting to note that various disabling processes admire the "Rule of Thumb".
The most sophisticated system that was reviewed used a manually developed decision tree based on a keyword search. This solution uses failure codes and significant words that occur in the failure description. Bounce reasons are classified into various classes and each class is assigned a bouncing weight. If the email address reaches a predefined threshold it is disabled. I was at first impressed by this approach; however after validating it with a training set, it was wrong in many cases. A disabling model corresponds with our definition of email address quality. If the quality is below the threshold, the system will disable it.

**Project objectives**

On our way to the Golden Age of Information, we first need to develop a systems that will be capable to maintain and improve the quality of data or information we capture and store. Without understanding the quality of data we own, it will not be possible to improve it. In previous sections we identified problems that data quality improvement systems need to cope with and suggested how these could be addressed.

The two most critical problems that can be seen are the inability to describe the data quality in the wider context, and the lack of techniques to measure the quality at the top level.

Good inspiration on how to achieve this first objective might be found in the manufacturing and software industries, where this problem is elaborated to a finer degree. My idea is to adapt a requirements analysis process to define data specifications and develop methods for a Verification & Validation process.

The needs → requirements → specification realization hierarchy is complex for data structures and it is debatable how low level data quality measures can contribute to high level quality needs. The use of data mining and statistical methods is suggested to hide this complexity and address the second deficiency.

To demonstrate how these ideas are used together in praxis, an elaboration of email address requirements and defining techniques that could be used for verification and validation will be presented.

Further, a suggestion of how these techniques can be implemented into a data V&V component will be given. This component can be plugged into any system that depends on email address quality, which could be a CRM or Email Marketing solution.
Chapter 3
Methodology

Verification & Validation
In software engineering, the Verification & Validation process is used to assure the quality of all products within various stages of the software development lifecycle. Although verification and validation describe different procedures [IEEE 1012-2004], [ISO/IEC 12207], it is not unusual to see these two terms confused.

ISO/IEC 12207 VALIDATION: "Confirmation by examination and provisions of objective evidence that the particular requirements for a specific intended use are fulfilled."

ISO/IEC 12207 VERIFICATION: "Confirmation by examination and provisions of objective evidence that specified requirements have been fulfilled."

Validation is a process that assures "we built the right thing", and verification is one that assures "we built it right". In software engineering, verification is done on a lower level and can be very formal while validation is done at a higher level with less formality. Verification confirms whether a product complies with the specification while validation with the needs.

These principles can also be applied to data quality assurance. Data are verified if they comply with its specification and valid if they match the purpose. Unfortunately, we often process data per-se, without any goal or objective. This implies that such data cannot be validated, and it raises the question, how such were verified, if the specification is not result of a requirements refinement. To apply these principles correctly, we shall first define the data needs to drive a concrete specification and then once the data are verified, we will validate it.

Data verification
The initial goal is to produce a formal specification that can be tested. From all the various quality attributes discussed in the previous chapter, we can clearly test data format, data values or maybe even data rules. The more vague a data quality dimension, the more complicated it is for us to find a method to test it. It is suggested that we focus only on the perfectly testable attributes, which can be fully and unambiguously specified. Quality dimensions, which might be partially tested with verification techniques, are data completeness, recency or duplicity.

Sometimes data will need to respect requirements that are not driven solely by our needs, for instance if shared with others. In this case we need to develop verification techniques to satisfy the other requirements or adopt techniques used by others. As long as we stay on the lowest level, and are able to define methods or measures with undisputable results, we are talking about data verification. Simply put, a property of a verification method is that we can be certain about its result. Verified data either matches or does not match the specification. We can further see verification as a syntactic test. This is why it is reasonable to verify the data first before we validate them. It is unlikely that incorrect data will meet our needs.
Data validation

The purpose of validation is to assure that data fits our needs. We are testing if the data objectives are satisfied. We are assuring if the data are helping us to infer the knowledge or reason about the subject they describe. Validation techniques shall test high level quality attributes, like coherence, accuracy, relevance etc. How can we test for coherence? The suggestion is that we try to define a measure related to the objective we have. For instance, if our goal is to send a personalized email to an individual person, we need to assure that all data points used in the message match the real person. We can compare the data to other data registers, or internally develop an algorithm to check the data semantics or recency. Such algorithms are mostly probabilistic and cannot obviously produce a binary result. If validating data we have to expect uncertainty, and accept the fact we might be even wrong.

The data quality dimension into verification or validation buckets has intentionality not been separated. There is no sense to do it, because this varies for each individual data entity. What is important, when building any data quality management model, is to understand to what degree the technique is covering the test, and how accurate is the result. This is called the potential of Verification & Validation methods [18]. When building validation techniques for data quality we will quickly realize that a validation result, even slightly better than a random guess, can improve the output of the process dramatically.

![Figure 3.1 V&V potential expressed as degree of uncertainty of quality assurance](image)

Email address specification

When defining quality attributes of a data entity such as an email address, we have to understand what needs are driving the quality attributes. Two systems, for example, Email Marketing (EM) and Customer Relationship Management (CRM), use email address to deliver messages, but their objectives might be different. An EM tool can be used to generate revenue, and the quality of an email address is defined as a potential to do it. On the other hand for CRM purposes it might be enough that the user read the message.

As mentioned earlier, an email address has a formalized structure and thus it should be testable. However, having a formally verified email address does not assure it is used by someone and thus valid. The disproportion between a verified and valid email address is so massive, that if the whole known universe was occupied by atoms representing syntactically correct email addresses, then a minute fraction of one atom will be occupied by valid email addresses.
If we focus on the deliverability attribute of an email address, then a set of not working, but syntactically correct email addresses $E_{invalid}$ is of low quality and worth nothing. The quality of a list comprised of valid working email addresses $E_{valid}$ might be of some value, but because of certain functionality constraints (mailbox full, SMTP issue, unsolicited content,..), even valid email addresses do not assure deliverability. Therefore, it might be for our marketing purposes more suitable to define $E_{reachable}$. We can go even further and define a set of email addresses of users who are going to buy some product $E_{buyers}$ which might be of great value and thus of high quality to marketing or sales people.

To specify what constitutes a verified email address is not that complicated, but what is more interesting is to how to define a valid one. We can surely say that

$$E_{valid} \supseteq E_{reachable} \supseteq E_{buyers}$$

Unfortunately, these are theoretical sets, not existing in the real world. In a system that stores email addresses, we will have a real $E_{stored}$ which is unique for each data repository, and our first goal is to assure that

$$E_{stored} \subseteq E_{verified}$$

Of course one might find “unverified email address” in some systems, but for our analysis these are just textual values labeled as email addresses, and should not be a big problem to address.

By cleaning our list of email addresses, and removing the unwanted data $E_{invalid}$, we can improve the entire list quality, and have

$$E_{stored} = E_{valid}$$

If we fail to constantly clean our list, it can degrade to

$$E_{stored} = E_{invalid}$$

In a real system the situation is somewhere in between, and we can thus define a probability that we will select a valid email address out of the stored list as

$$\pi = \frac{\#(E_{valid} \cap E_{stored})}{\#E_{stored}}$$
Estimate of $\pi$ might be calculated and used to describe the quality of the list of email addresses (in this case the objective is to have a list of valid email addresses).

When planning for constant improvement we need to factor into our model the fact that although $E_{stored}$ will remain fixed, its quality will change as email addresses die out. In fact the probability $\pi$ is non-increasing in time.

$$\lim_{t \to \infty} \pi(t) = 0 \quad (3.9)$$

Verification & Validation of an email address

Business requirements for data usage define quality attributes which can be validated and technical specifications define quality attributes that can be verified. As a collection of valid email addresses is a subset of verified email addresses (3.2), and knowing that verification shall happen prior to validation, we are now able to model Validation & Verification states of an email address, as in Figure 3.2.

![Figure 3.2 Model of V&V states of an email address](image)

Because the valid state is modeled as a sub-state of verified, we exclude the possibility that email addresses can be valid, but not verified. Although I see this as a correct approach, it is not implemented in this manner in many systems. This is mostly because the verification techniques are either not fully testing the specification or the specification is not fully defining the properties of the data entity. This is the case when only regular expression matching is used to verify an email address. This should be considered a deficiency of the verification process rather than refutation of the model presented here.

Another interesting aspect of our model is that we allow a transition between the Valid and Invalid states. This is because existing email addresses can become invalid if not used. In equation (3.9), we defined this transition as a function of time. For our V&V process, it means that we will need to remember the date of validation, and revalidate the email address if the probability of transition gets high. Analogically we can do the same for the transition from the Invalid to Valid state, but since this is very unlikely to happen we can ignore it. When designing the V&V framework for a specific entity we will need to remember that Transitions between Valid and Invalid states are often the property of many data entities.
Verification & Validation module design

In the PDCA section it was mentioned that any production process shall be part of a constant improvement cycle. Considering our production process as a system that sends messages, our objective should be to constantly improve its delivery ratio.

\[
\text{delivery ratio} = \frac{\text{delivered}}{\text{sent}}
\]  \hspace{1cm} (3.10)

Delivery ratio is related to the quality of an email address. The module that verifies and validates an email address will play an essential role in the continuous deliverability improvement. Quality of an email address, although important is not the only attribute that influences the deliverability. When we plan to improve the messaging process, other attributes need to be considered. For instance, if the mailing system is not connected to the Internet, none of the messages will get delivered, irrelevantly of how many valid email addresses are used. All attributes that have an effect on the deliverability will need to be identified and this cause-and-effect analysis is what was suggested earlier. We start from the top objective and break it down to the lower levels. If we are not able to meet the quality expectations of the higher level, it is because quality of the lower level component is not sufficient. This way we search for the components that cause the delivery problems. For an email marketing manager, it is not important how the SMTP system is configured, as he judges performance of the system as whole, but for those who want to understand and improve the process it is a must. Although our primary focus is to develop techniques for email address validation, it would not be possible without having insight into what else is affecting delivery. This is why we need to classify the bounce failures beyond the hard and soft bounces and to distinguish between failures caused by invalid email address and everything else.

A system that gives us this insight (and will need to be built before we can start developing email validation techniques) is called a Bounce Collector. Its purpose is to collect failure messages and classify them according to the cause of the failure. Based on the discovered bouncing patterns that can be attributed to invalid email addresses, it will be possible to train the Probabilistic Email Address Quality Reasoner. This will become the core component of the Validation Module and could be used to reason about the quality of the messaging process as well.

![Figure 3.3 Schema of V&V module](image)

Because the Bounce Collector is a robust system, in a situation where we would need to minimize the system footprint, it should be possible to deploy the V&V module without it. In such configuration the Validator will depend only on the trained Probabilistic Reasoner and will not be able to learn further information. It will also lose the potential to remember invalid email address and will give up other historical information about mail delivery.
Chapter 4
Verification techniques

Verification of an email address
Data stored in a digital format, are by default verified to the level of the storage compatibility - a
data type level (numerical, textual, etc.). If we decide to store an email address, we usually use a
textual, character based type. Considering a relational database as the target storage of the email
address, we need to define the length of the data type. This is the first verification rule that
applies to data. It would beneficial if that would not remain the only quality rule, as it is far from
the perfect verification technique. Since verification is the test of specification conformance, we
must have the specification before any verification technique can be developed. For email
addresses, this will not be that complicated, since besides our requirements, we can use the
Internet standards that define SMTP communication and message format. The genesis of email
address related technologies over the last few decades required many specifications and there is
no single specification that defines all of its properties. An interesting exercise is to find out the
length of an email address. Simply searching on "email address length" yields diverse results and
references to many different RFCs.

The first objective is to reconcile all the available specifications that specify email address
properties and present a consolidated structure to use for the development of email address
verification techniques. Suitable language for the syntax definition of an email address structure,
used in the latest standard for SMTP communication, is the Augmented Backus–Naur Form
(ABNF) [RFC 2234]. To remain consistent, we will also use it for our specification.

Email address structure refinement

Email address
Internet Message Format RFC 5322 specifies an email address (precisely address) as an individual
mailbox or group of mailboxes.

\[
\text{address} = \text{mailbox} / \text{group} \\
\text{group} = (\text{mailbox} * (",", \text{mailbox}))
\]  

Although there is a possibility to send one identical message to a group of recipients, it is
impractical to do it in situations where a single user behavior needs to be monitored. If a
traceable behavior snippet is sent to a group, then we will observe behavior of the entire group
instead of the single recipient. As this is undesirable functionality, we will remove the group from
our specification, to assure that one email address is related to a single recipient. In certain cases
even a single mailbox can be shared between physical users in the case of an alias or when a
mailbox is shared in the household. By removing the group we simplify the email address
structure and can say that address is equivalent to a mailbox.
address = mailbox \hfill (4.2)

Mailbox is further defined as

\[
\begin{align*}
\text{mailbox} &= \text{name-addr} / \text{addr-spec} \hfill (4.3) \\
\text{name-addr} &= \text{[any text] angle-addr} \\
\text{angle-addr} &= \text{[CFWS] "<" addr-spec ">" [CFWS]}
\end{align*}
\]

RFC 5322

In general name-addr form is helpful if one need to keep more information about the mailbox. For instance John Smith <js@example.com> is more descriptive than a simple addr-spec. As name-addr part has no effect on email deliverability, it could be thrown away and we can further rewrite the definition mailbox to

\[
\begin{align*}
\text{address} &= \text{mailbox} \\
\text{mailbox} &= \text{addr-spec} \\
\text{addr-spec} &= \text{local-part "@" domain}
\end{align*}
\]

and

\[
\text{address} = \text{local-part "@" domain} \hfill (4.5)
\]

Local-part

As the domain component is not well covered in RFC 5322, we continue with local-part and will try to refine domain later, based on other RFCs.

\[
\text{local-part} = \text{dot-atom} / \text{quoted-string} / \text{obs-local-part} \hfill (4.6) \text{ RFC 5322}
\]

Although RFC 5322 still allows the obsolete form of the local-part of an email address obs-local-part, it is stricter in its new structure. Since the obsolete definition was not that restrictive and allowed characters many current systems reject as syntactically incorrect, we can exclude it from our specification. Furthermore, the usage of quoted-string is discouraged by the RFC 5321, so after the reduction

\[
\text{local-part} = \text{dot-atom} \hfill (4.7)
\]

And elaborating the dot-atom further

\[
\begin{align*}
\text{dot-atom} &= \text{[CFWS] dot-atom-text [CFWS]} \\
\text{dot-atom-text} &= 1*\text{atext}(\.1*\text{atext}) \\
\text{atext} &= \text{ALPHA / DIGIT / "} / ";" / ";" / ";" / ";" / ";" / ";" / ";" / ";" / ";" / ";" / ";" / ";" / ";" / ";" / ";" / ";" / ";" / ";" / ";" / ";" / ";" / ";" / ";" / ";" / ";"
\end{align*}
\]

RFC 5322

CFWS, as a comment structure, has as well no effect on deliverability, and can be thus excluded. The simplified specification of local-part for our email marketing system is now

\[
\text{local-part} = 1*\text{atext}(\.1*\text{atext}) \hfill (4.9)
\]
The minimum length of a local-part is one character, and this is the maximum about local-part we can elicit from the RFC 5322. Continuing with the RFC 5321 we see that the maximum length of the local-part is 64 characters, and

"The local-part of a mailbox MUST BE treated as case sensitive… However, exploiting the case sensitivity of mailbox local-parts impedes interoperability and is discouraged." It is hard to imagine that any email service or mailing system will force strictly case sensitive mailboxes, especially if this is not required for the domain part. For our marketing system purposes, we will ignore the case sensitiveness as well.

Besides the case insensitiveness, some other platforms define the behavior of special characters that goes beyond the RFC specification. For instance jane.doe+test@gmail.com is identical to janedoe@gmail.com [50],[51]. Eliciting such system specific rules, might further improve our specification completeness and thus the verification engine, unfortunately these are not easy to collect, so we will not restrict our system with them.

**Domain**

RFC 5322 gives "A liberal syntax for the domain portion of addr-spec…"

\[
\text{domain} \; = \; \text{dot-atom} / \text{domain-literal} / \text{obs-domain}
\]

(4.10)

RFC 5322

We can again exclude the obs-domain and make our system more restrictive. In the form of the dot-atom the domain is an Internet domain name, and the domain-literal is an Internet protocol address of the host. In a list of email address which was used in this study for the algorithm evaluation only about 10k email address out of 150 million were using the domain-literal form. And since this is a fraction of the entire list, it is safe for our email marketing purposes to ignore it, and simply rewrite the domain definition to

\[
\text{domain} \; = \; \text{dot-atom}
\]

(4.11)

The other reason why defining the domain as the domain-literal is impractical, is because any change of the host Internet address invalidates the email address. The definition (4.8), of dot-atom syntactically allows the Internet address. As this is not desirable it needs to be refined.

\[
\begin{align*}
\text{Domain} &= \text{sub-domain} * (\text{"."} \text{ sub-domain}) \\
\text{sub-domain} &= \text{ld-str} [\text{ldh-str}] \\
\text{ld-str} &= \text{ALPHA} / \text{DIGIT} \\
\text{ldh-str} &= * (\text{ALPHA} / \text{DIGIT} / \text{"."}) \text{ ld-str} \\
\text{ALPHA} &= \%x61-7a ; a-z \\
\text{DIGIT} &= \%x30-39 ; 0-9
\end{align*}
\]

(4.12)

The purpose of an Internet domain is to hide the addressing behind a user friendly name, which the SMTP server must locate in the DNS. It is thus reasonable to factor into our definition requirements from DNS related RFCs (especially RFC 1034 and RFC 1035), and finalize the domain definition to the form shown in equation (4.12).

In RFC 5321, the length of the sub-domain is restricted to 63 octets, and the whole domain to 255 octets. RFC 1035 defines the domain length to 255 characters, however it states that "...every domain name ends with the null label of the root,..." and RFC 1034 states "...labels are separated by dots (".").
Since a complete domain name ends with the root label, this leads to a printed form which ends in a dot. Although the trailing dot as impractical was omitted from the RFC 5321 specification, the same length of 255 characters remain, which might lead to a false assumption that the maximal domain length if stored in DNS can be 257 characters. As this is beyond the allowed limit of DNS, the real maximum length of the domain part must be only 253 characters.

I was not able to locate anything about the minimum length of the domain, however assuming that besides the arpa domain, all domains must have a second level sub-domain. We can calculate the minimum domain length as 4 characters = tld (two characters) + "." + one character for the second level domain.

**Address length**

Having the length of a domain and the local-part defined, we can theoretically calculate the maximum length of an email address. RFC 3696 (informational) defines its length to 320 characters. local-part + @ + domain → 64 + 1 + 255 = 320 characters.

Ignoring the ambiguity of the domain length mentioned earlier, this is not consistent with the RFC 5321, which defines maximum length of a path to 256 octets including the punctuation and element separators.

\[
\text{path} = "<" \text{mailbox} ">" \tag{4.13}
\]

RFC 5321

The real length of a mailbox is between 6 and 254 characters. Having this information we shall further refine our domain length. If counting one for the local-part and the other for the @ sign, than given the path limit, the domain must not exceed 252 characters.

**Other standards**

The right most sub-domain of the domain specification is called the Top Level Domain (TLD) and cannot be a random combination of the ALPHA and DIGIT characters, as the specification might suggest. TLD is a list of about 300 entries, maintained by the Internet Assigned Numbers Authority IANA [53]. As this is different to other sub-domains it is worthy to separate it.

\[
\text{domain} = 1*126(\text{sub-domain ".") tld} \tag{4.14}
\]

\[
\text{tld} = 1*63(\text{ALPHA})
\]

It should be as well noted that Internet Corporation for Assigned Names and Numbers (ICANN) [54], currently works on internationalisation of Internet domains (domains in native languages). As this is still not widely accepted standard, it is not factored into this specification at all.

Further, the formalized syntax of an email address in the form of ABNF grammar is still not a full specification. For instance, the length constraints and composite rules could not be easily documented by this language and were thus omitted. From the final ABNF definition of an email address see Appendix A.
Verification techniques - TLD

Lookup verification
TLDs can be grouped into more categories, country specific [ISO 3166-2] (except for UK), generic top level domains (com, info, …), sponsored top level domains (gov, int,..) and arpa. We ignore international test domains. The verification of a TLD is a very simple technique since we only have to assure the TLD exists on one of the three lists. If the TLD is not on the list, then the entire email address is not correct.

All three lists can change over time, so it is desirable to track the changes. Until now the changes occurred sporadically, but this can change in the future as ICANN [53] plans to introduce a new concept to generic TLDs.

Verification techniques - Domain

Lookup verification
Although it might be possible to have a list of all second level domains, it would be hard to maintain it. According to [54] there are over 130 million domains and 300k daily changes for the top 5 TLDs. Building a list of third and higher level domains would be almost impossible. As our requirement of verification techniques is that it shall provide a reliable binary result, it would be hardly achievable with such volatile lists. It should be considered that if such verification error is acceptable or if such list technique shall not be treated as a validation technique with probabilistic results.

Exceptions include countries where second level domains are regulated. In this case the verification problem is just pushed to the third level domain. It is worthy to include such exceptional sub-domains on the TLD list.

Grammar verification
It is possible to either generate a parser or a regular expression based on the ABNF grammar. For our simplified version of the domain structure, this shall be a suitable verification technique. Keeping in mind the performance of such parsers, it is more efficient to test the entire email address, and not only its domain part.

Length verification
Any sub-domain must not be longer than 63 characters, and there could be a maximum of 126 sub-domains excluding the TLD. The minimum length of domain is 4 characters and the maximum is 252 characters.

Additional rules
In addition to the second level TLDs, the Network Information Center (NIC) for each country controls the structure of the sub-domain. This is often an extension to the RFCs. For instance the Nominet (NIC) of the UK, in their Rules of regulation [55] (among many), limit the domain length to 64 characters. They also do not allow single character third level domains and regulate the sequence of ALPHA and DIGIT characters. Again, this information is very beneficial to know and very complicated to maintain.
Verification techniques - Local-part

Lookup verification
Local-part, besides the length and allowed characters, is not regulated. Except for the few restricted mailbox names [RFC 2142], verification against a list of all entries is not possible, because such list does not exist.

Length verification
Local-part length must be between 1 and 64 characters.

Grammar verification
Besides testing that only atext characters are used, the only grammar rule is to assure the dot is surrounded by an atext character.

Verification techniques - Email address

Lookup verification
Having all subcomponents of the email address verified, we can do mailbox existence tests. Of course there is no list of all verified email addresses and only SMTP servers know about their own mailboxes. RFC 5321 still proposes VRFY and EXPN commands to allow verification of email address against the SMTP address book. These are relics from the earlier times, when no one expected "spammers" would exploit them. Although it was nice idea, it is unusable at present time. Another dirty verification technique is to establish an SMTP connection and send the email address for verification with an RCPT command and disconnect after the server response. Some servers test the email address existence immediately and reject it if not found in the address book. To avoid exploiting this feature a majority of servers accept anything that is syntactically correct.

Length verification
As the length of local-part and domain is now verified, we must only assure that the sum of both is bellow 264 characters.

Grammar verification
There is no reason to test an email address if the local-part and domain are verified. An email address is a simple concatenation of both with the @ sign in the middle. The only reason to test the grammar at this point could be performance, as testing the whole email address structure can be as costly as to test its single components.

Delivery test
Some systems send a verification email with a token after the email address is acquired, and the user must provide the token back to the system (for instance by post back). This is suitable only for the initial acquisition of the email address, when the user is interacting with the system. The verification potential of this method is disappears once the user stops communicating. It is an unsuitable technique for verifying an external list of email addresses where the motivation of people to respond is low. Testing the quality of the email addresses by sending a message to it
and determining its quality based on the bounce failure, is also an unreliable technique and will be covered in the validation section.

Email Verifier
A verification component provides a simple interface that accepts email address as the only parameter and returns true if the email address matches the syntax or returns false if not.

![Figure 4.1 Schema of email address verifier, with its interface](image)

What is happening behind the scenes is depicted on the activity diagram Appendix B. The first step is to verify the syntax with a grammar verification engine. This could be achieved by the grammar parser or a regular expression which is generated based on the ABNF grammar Appendix A. The process further separates the domain and the local-part. Although the verification of these components is depicted as a parallel process, it can be sequential as well. In case any of the verification steps fail, it is obvious that the email address will not work, and its syntax is bad. The process terminates the verification immediately.
Chapter 5
Validation – Bounce classification

Evaluation of validation techniques
Validation techniques shall assure that data fits our needs. This might be a particularly difficult if techniques used are not precise or our needs are not easily convertible to measures. In most cases validation techniques are facing both deficiencies, and that is why validation methods cannot give a definite answer and end up yielding only a probabilistic assurance of the data quality. As it is possible to make a wrong statement about the data quality, it is necessary to understand how likely such mistakes do occur. These classification mistakes are twofold, invalid data are treated as valid and valid data are treated as invalid. Analogically to a hypothesis testing it is possible to define the probability of misclassification, and use a similar measure to the Type I Error $\alpha$ \eqref{eq:type1} and the Type II Error $\beta$ \eqref{eq:type2}.

\[
H_0: \text{the email address is valid} \quad (5.1)
\]
\[
H_1: \text{the email address is invalid}
\]

\begin{figure}[h]
\centering
\begin{tabular}{|c|c|}
\hline
\thead{H_0 is true} & \thead{H_0 is false} \\
\hline
fail to reject $H_0$ & no error \hfill type II error \\
\hline
reject $H_0$ & type I error \hfill no error \\
\hline
\end{tabular}
\caption{Matrix of Type I and Type II Errors}
\end{figure}

\[
\alpha = P(E_{\text{disabled}}|E_{\text{valid}}) \quad (5.2)
\]
\[
\beta = P(E_{\text{enabled}}|E_{\text{invalid}}) \quad (5.3)
\]

For each data entity, the impact of using bad quality data is different and changes per the objective. It is thus not possible to apply existing single evaluation model to all data needs blindly without proper weighting of the misclassification impact. Having a valid email address classified as invalid will, in our case, exclude it from any future mailing, and the opportunity gain of mailing to it, will disappear.

\[
c_{\alpha}, c_{\beta}, \quad c_{\alpha} = c_{\beta} = -1 \quad (5.4)
\]
\[
c = \alpha c_{\alpha} + \beta c_{\beta} \quad (5.5)
\]

Mailing an invalid email address will affect the sender reputation and also have impact on the network and SMTP server utilization. Some techniques are better at avoiding false-positive errors and others are better at false-negatives, so it is necessary to compare both parameters. It is common that the weight of each error type is different, and generally the Type I Error is treated as more important. However, for the sake of simplicity in our model both weights \eqref{eq:weights} will be
treated as equal. Having both weights defined, we can use a simple cost function (5.5) to evaluate each individual validation technique, where the one with a minimal loss wins.

**Measuring**

The most reliable, although not perfect email address deliverability measurement, is to send a message and wait for the bounce failure to occur. If the target server responds with a failure message, it is an indication that there was a delivery problem. If no bounce is received it indicates a good quality email address, unless the bounce was lost. Although a model solely based on bounce detection is simpler, it might be beneficial to extend it with some other kind of direct measurement that would minimize the α error.

![Figure 5.2](image_url) **Schema of two models for email addresses disabling.** The model with only bounce detection is yielding higher α error because \( E_{\text{delivered}} \) contains three hidden sets. Bouncing email addresses with missing bounce failure, deliverable email addresses with missing response and responding email addresses. In the model with bounce and response detection, \( E_{\text{delivered}} \) is reduced to bouncing email addresses with missing bounce failure and deliverable email addresses with missing response.

A suitable direct measurement might be responses, like email opens and clicks. Opens are measured through the request for a traceable unique image which is included in the email body (not working if the email client blocks it), and click works as the URI redirect (the URI points to a measuring application that forwards it to the final destination and collects each requests). A system that invalidates email addresses can reason about the email address quality based on either the bounce potential or the response potential. An email address with low response probability and high bounce potential will be disabled. This behavior can be as well useful for evaluation of the validation technique where true-positives are not easily detectable. If a control email is classified to be disabled and an open or click response is detected after the next mailing, we can classify it as false-negative error. If that same control email is classified as not disabled and a bounce is measured it will be a false-positive error. Analogically to hypothesis testing we can reshape the matrix and compare the truth and the result of the validation algorithm.

![Figure 5.3](image_url) **Matrix of concrete validation mistakes for email address validation**

<table>
<thead>
<tr>
<th></th>
<th>valid email address</th>
<th>invalid email address</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>Right true-positive</td>
<td>Wrong false-positive</td>
</tr>
<tr>
<td>disabled</td>
<td>Wrong false-negative</td>
<td>Right true-negative</td>
</tr>
</tbody>
</table>

Similarly to the likelihood of errors in hypothesis testing α and β, we can define for validation technique analogous metrics.
Evaluation of the validation potential is the same for both the prior (not using historical information about bounces) and post (based on historical bounce patterns) delivery methods.

Collecting bounce data

Bounce collecting architecture

Before we can start using the model with bounce detection it is necessary to start collecting bounces and understand them. In the case of a delivery failure, an exception is raised and the user is notified about the situation. This failure notification is either given to the user immediately while still connected to the server or later as the content of a Non-Delivery Report (the bounce). The method in which electronic messages are currently routed practically eliminates the possibility of synchronous error handling, and most of the failures are sent as bounce messages. This, of course, affects the timeliness of the failure delivery and bears other complications. According to [15], 25% of bounces are lost and 10% are delayed. It is thus important when designing the bounce collecting system to have a solution that will be able to cope with the delayed delivery and missing bounces.

As bounces are also regular messages, anyone who wants to receive them must have a mailbox. If the sender does not have one, it could lead to a peculiar situation where the undeliverable failure message generates another failure message. As some mailing parties (“SPAM”ers) usually do not have one, an SMTP server generally drops undeliverable bounces after a certain number of round trips. Having just one mailbox for all bounces is not practical, because it would not be possible to match the returning failure to the sent messages. It is essential to have an individual mailbox for each message sent. After bounces are collected somebody needs to process them. Administrators responsible for SMTP servers in a large organization do not usually have time to take care of all the bounces collected. And there is no wonder, because if an SMTP server will send one million messages, it will generate [13] 30 000 bounces. This ratio can, of course, vary, however a decent email marketing campaign sent to tens of millions of users will produce bounces that could make the administrator busy for months. Somebody who would like to use bounces as a source of information about the email address quality and for delivery ratio improvement will need use a bounce processing system.

The first step when building a system for bounce collection is to separate human and machine messages, and deploy them through different set of independent servers. It is as well necessary to have the bounce processing system separate from the corporate inbound SMTP servers. Mass marketing systems are designed and tuned to send millions of emails and often consume all of

\[
\text{precision} = \frac{tp}{tp + fp} \quad (5.6)
\]

\[
\text{recall} = \text{sensitivity} = \frac{tp}{tp + fn} \quad (5.7)
\]

\[
\text{accuracy} = \frac{tp + tn}{tp + tn + fp + fn} \quad (5.8)
\]
the available bandwidth. Many people will not be happy when their message is blocked by millions of marketing emails. The topology of such a system might look like in Figure 5.4.

As part of some standard human communication, "jack" wants to invite "john" for an exhibition so he sends a message which is routed through the "gate.example.com" server. As "john" is no longer working for "foo.bar" and his mailbox is not active, the SMTP server "mail.foo.bar" replies with a bounce message to inform "jack" about the delivery failure. The failure is routed back through "gate.example.com" to "jack's" mailbox.

As part of the email marketing communication at "example.com", "john" is identified as a prospective customer. “john” and a million other users are mailed a message containing an offer. Because the automated message generator is used, the message to "john" is routed through the "bulkmailer.example.com". As we know that "john" is no longer working at "foo.bar", the offer will not get delivered. Instead of generating revenue, the SMTP server at "mail.foo.bar" generates a bounce message, and sends it back to the "bouncecollector.example.com" server. The marketing team at "example.com" knows that "john", as well as many others, did not receive the offer. Hopefully they understand that invalid email addresses were used and that they shall exclude "john" from all future mailings.

**Bounce data**

**Selecting bounce data for analysis**

Bounce data is the most important input needed for an automatic email address validation system. Without bounce data we will not be able to build a predictive validation model nor validate already mailed email addresses.
For my analysis, data from a large Internet enterprise were used. This system maintains over 150 million of active email addresses and sends over one billion email messages each year. From the entire data available, two random sets of users who created their accounts between September 7th and October 27th, 2009, were selected. Bounce failures and user responses collected from those users, over a period 18 months, were used to train and validation models. Before any analysis was done, data were cleansed and separated from anomalies, which could have impact on the analysis. (Only users with fresh never contacted email addresses were selected, user who did not changed their email addresses, etc). The entire list contained over 3 million of users who received, in total, about 30 million messages, which caused over 1.5 million bounces. The bounce collecting system used for the data acquisition faced two small outages, which caused a data loss. These losses were minimal and below the estimated natural loss of bounce messages.

Initial bounce data analysis
Before we start using the bounce data, it might be worthy to explore some of its characteristics. And instead of the delivery ratio (3.10), it is now more practical to look at the bounce ratio which is defined as

$$\text{bounce ratio} = \frac{\#\text{bounced}}{\#\text{sent}} = 1 - \text{delivery ratio} \quad (5.9)$$

If we look at Figure 5.5 to see how the bounce ratio changes over the course of time, we see that it starts at 5% and after an initial decline, it increases. After the 11\textsuperscript{th} message it declines again until the 16\textsuperscript{th} message, before a final steady growth. The final growth is assumed to be the effect of email address aging, which is not the most important factor at the beginning. Content of the message or other factors have more effect. This supports the idea (3.9) that the older an email address is the more likely it will become invalid.

![Figure 5.5 Bounce behavior for ten various samples from the training set](image)

The first conclusion we can make about bounce ratio is that it is not static. After the bounce classification algorithm is developed, this behavior can be explored in more detail. Before getting into it we shall continue with our initial data analysis and look at the bounce data from the marketing perspective, where soft and hard bounce classification is used.

The delivery of a message to a hard bouncing email address will constantly fail, whiles delivery to a soft bouncing one will only sporadically fail. If the sent message generates a bounce, we will
code it as 1 and if no bounce then 0 (message is considered to be delivered). After the initial transition from the unknown state is made (this is the prior probability of 5% depicted above) there could be only four possible transitions (0→0, 0→1, 1→0, 1→1) and the whole behavior model can be depicted as in the Figure 5.6.

Although this model is quite simple, it clearly visualizes a few interesting aspects of bounce behavior. The probability of 1→1 (bounce to bounce) transition is higher than 1→0 (bounce to delivered), which leads us to the assumption that there will be more hard bounces than soft bounces. It is actually very important that the 1→0 transition exists, because this proves that soft bounces really exist and such classification makes sense.

### Bounce transition chains

The average probability of the state transition is displayed in the Figure 5.6. It could be interesting to see if it is static or changes over time, respectively over the number of messages sent. There is quite a desire to see it not change, because in that case it would be a simple Markov first-order chain. Without any testing [56] it is evident from Figure 5.7 that this simple model could not be used for the entire life of the email address as 1→1 transition (hard bounces) is after the 16th message increasing. From the data we can also notice that the bounce sequence of transitions is changing from 000000... to ....111111. While analyzing the bounce data, missing bounces were indeed detected, and some email address which were provably invalid, were missing bounce records. Further analysis will need to happen to see if the number would be as high as 25%.
Structures to process bounces - Part 1

In this section we will take a closer look at the functionality of message generator and the bounce processor engine.

Message Generator

The message generator is responsible for the creation of email messages. The only crucial requirement of this system is to stamp the message with a unique id which will be used to identify it in case of a bounce. The message id shall be stored and used to match the bounce in the future. It is possible to extend the stored data for reporting purposes with recipient who was mailed (RCPT_TO), the name of the system that generated the message (MAIL_FROM), etc. The simple message id or the enriched information is as msgInfo provided to the bounce processor immediately after the message is generated.

Bounce Processor

This system is, for our purposes, the most important. It is validating the email addresses after mailing them and also has the most influence on disabling them. It is also providing data for the bounce prediction module and keeps historical information about email addresses and their bounces. An existing bounce processing system was used as a base for this new system, and we might call this one version II. The main reason to redesign the system was the insufficient information it provided about the bounces collected. The new version of the bounce processor is based on multiple modules, where each is responsible for a specific task. The original solution consisted only of the SMTP engine and the Email Parser.

SMTP Engine

The SMTP Engine is responsible for receiving emails (at this moment we cannot call them bounces yet) through the SMTP interface. Although it is as well possible to expose its interface to the Internet, I would recommend having it behind the MTA. This configuration will simplify the implementation logic and will add additional security. Although the SMTP engine looks like a

---

Figure 5.8 Component diagram of a bounce processing system
robust system which maintains millions of mailboxes, it is truly just a FCFS (first-come first-served) queue that makes the data available for the Email Parser.

While estimating the queue capacity, it is important to keep in mind the size of the email campaigns and their deployment schedule. The average size of a bounce message is about 5kB and the queue shall be able to accept bounces for a minimum of three days (due to failure in the post-processing, where the queue will not be emptied). It is as well possible to deploy more instances of the SMTP engine to assure higher availability. And deploy them on an isolated environment, since there is a serious threat of virus infection. If bounces are stored on the file system, one should consider the appropriate file system, for the amount of files managed.

The system starts receiving bounces immediately after the generator starts sending the messages and they continue to be received even after the deployment has finished. On the Figure 5.9 we can see the difference between the rate at which messages are sent and the rate at which bounces are received. In this example 1.5 million messages were sent and 315k bounces were received (an unusually high 21% bounce rate). The maximal deployment threshold was configured to be 50k messages/minute and in 30 minutes it completed. It was seen that 85% of all bounces were received in 60 minutes, and were coming in at average rate of 4.5k bounces/minute (slightly below 10% of the sent rate). 10% of the remaining bounces were collected within the next 60 minutes. This was not the only the deployment of such size on a single day, but it could give an idea how to estimate the size of the queue.

![Figure 5.9](image-url) 

**Figure 5.9** Example of a campaign deployed to business email addresses acquisitioned from a third party company which specializes on Business Contacts and Company Information.

**Email Parser**

My initial thought was that the email parser will not be doing anything special, but I underestimated the variability of all possible bounce failures and the way SMTP servers treat them. The functionality of the email parser is to determine if the message collected is a bounce message and if so to extract the failure description it contains. All other emails are thrown away.

To monitor the quality of the email parsing engine it is suitable to implement various measurements such as processed/deleted message, parsing failures etc. The output of this component is the failure description as one would receive in the direct SMTP connection. For instance, when a server tries to send a message to a non-existing mailbox it will see on the protocol level the following message (messages are anonymized)
MAIL FROM:<jack@example.com>
250 2.1.0 OK
RCPT TO:<john@foo.bar>
550-5.1.1 The email account that you tried to reach does not exist. Please try
550-5.1.1 double-checking the recipient’s email address for typos or
550-5.1.1 unnecessary spaces. Learn more at
550 5.1.1 http://mail.foo.bar/support/bin/answer.py?answer=6596

Where the bounce message generated by QMail [43] MTA looks like this.

Hi. This is the qmail-send program at smtp.example.com.
I’m afraid I wasn’t able to deliver your message to the following addresses.
This is a permanent error; I’ve given up. Sorry it didn’t work out.

<joh@foo.bar>:
xxxxxxx.xxxxx.x does not like recipient.
Remote host said: 550-5.1.1 The email account that you tried to reach does not exist. Please try
550-5.1.1 double-checking the recipient’s email address for typos or
550-5.1.1 unnecessary spaces. Learn more at
550 5.1.1 http://mail.foo.bar/support/bin/answer.py?answer=6596 g13si3120799fax.40
Giving up on xxxxxxxxxxxxx.

Unfortunately each SMTP server is formatting the message differently, and it is not easy to
perfectly locate the content that was given on the protocol level. For instance, on an MS
Exchange server, the same failure looks different and even omits the full message.

Your message did not reach some or all of the intended recipients.
Subject: Some subject
Sent: 2/28/2010 12:08 PM
The following recipient(s) cannot be reached: john@foo.bar on 2/28/2010 12:08 PM
There was a SMTP communication problem with the recipient’s email server. Please contact your
system administrator.
<exchange.example.com #5.5.0 smtp;550-5.1.1 The email account that you tried to reach does not
exist. Please try>

Although the email parser shall locate and extract three pieces of the information (MAIL_FROM,
RCPT_TO and FAILURE) and provide it to the email matcher, it is not always working correctly. The
desired output from the parser should look like this

MAIL_FROM=“jack@example.com"
RCPT_TO=“bad@foo.bar”
FAILURE=“550-5.1.1 The email account that you tried to reach does not exist. Please try 550-5.1.1
double-checking the recipient’s email address for typos or 550-5.1.1 unnecessary spaces. Learn
more at 550 5.1.1 http://mail.foo.bar/support/bin/answer.py?answer=6596”

The email parser was mostly taking only the first line of the failure, and as this limited parsing has
an effect on the bounce messages classification, it is desirable to improve it.

In a later stage of this project, while researching the information value of bounce failures, I
realized that other information, included in the bounce message header and ignored by the parser
is as well valuable. For instance anti-spam scores or message route can add additional
information.
Email Matcher

An email address of the recipient included in the bounce message and detected perfectly by the Email parser, is not sufficient to pair the bounce message with the original email. It was expected to see DoS attacks on the bounce collector, so it was no wonder that the system received many bounce messages that looked like they were sent by the message generator, while they were not. What was more interesting were bounces of messages that originated in the system, but were not sent to the email address it was bouncing from. After detailed analysis it was discovered that those are caused by active forwarding to an inactive account, either on the mailbox or domain level. The only functionality of this component is to keep a log of all sent messages and authenticate the received bounce failures based on the message id and email address.

SMTP Engine, Email Parser and Email Matcher are simple components without any sophisticated logic. Only the improvement of the Email Parser could be a challenging task, but its functionality is sufficient to build a validation engine that can successfully classify the bounce failures.

Bounces classes

Before we further elaborate the components of the Bounce Processing System, it is necessary to mention how the Classification engine was developed, and why it was so important for the future development of the Verification & Validation Engine. In the previous section a bounce transition model was presented which has the potential to predict a probability of the consecutive bounce. As this model has a potential to be used for post mailing email address validation, it is important to determine the cause of bounces more precisely and separate the types that are not caused by an invalid email address. A classification based on a Status Code or Extended Status Code, is unreliable. This is not because the classes are wrong, rather that SMTP servers are not following the RFC specifications or the specification is not matching their need. Sometimes the same failure is coded differently and sometimes different failures are coded the same. See more on this topic in the Email Communication section. In turn, we are not able to use the set of failures defined in RFC 1893 or any other RFCs that cover this matter. Therefore, it is better to follow the idea of the soft/hard bounce concept, and elaborate it into more granular classes.
In search of the ideal classification framework

A new classification framework for bounce failures shall meet two criteria. It should be understandable to people with no technical background, and it shall be detectable by the classification algorithm. As a base for the new classification framework, I used RFC 1893 despite its limited applicability. It was used not to drive the class definitions, but rather as a list to cross check the new classes.

The original assumption was that failures close in terms of the cosine similarity [1], will describe similar problems. After testing various weighing scenarios [1] over the term vector matrix and applying a hierarchical clustering methods [2],[3] some clusters emerged. It was evident that terms have a potential to separate the failures. After manually investigating concrete bounces and their clusters, it was decided that a hierarchical classification structure is definitely correct. For example, if a server is not reachable, it does not matter that the email address is valid. The idea was to look at the email communication from the perspective of components involved and base the classes on those.

This new classification framework identifies two top level components, the sender and the receiver. Problems on either side can prevent email delivery. The level of bounce failure in the hierarchy depends at which stage the problem occurred. If the sender is not able to resolve the IP of the receiver, then it is a host issue. If the server is not able to establish a connection, it is an SMTP issue.

If the sender can connect and still not deliver the message, it could be a problem with the email address (mailbox) or the message or the sender. These are now the other three classes. Each class was then further broken into categories. Figure 5.10 depicts how the classification framework is related to the component model.

Such a set of classes is much simpler than what is defined in the RFC 1893 or others. Of course being influenced by richness of classes some RFCs suggest and my own desire to precisely describe each individual failure case, I experimented at the early stage with other categories as well. The focus was especially placed on defining more granular categories for Domain and Sender classes. It turned out later that such cases are very rare and thus hardly detectable. So rather than having over fitting issues of the classification algorithm, those were grouped under one umbrella. To merge detailed failure groups into one category is as well in line with one of the
requirements for the bounce classification system, which is to simplify the task of an SMTP administrator. The amount of all possible cases that falls into these two classes is only a small portion of all failures and will drastically help the administrator to spot server issues. Figure 5.11 depicts the final four classes and their categories.

**Classification of bounce failures**

<table>
<thead>
<tr>
<th>domain</th>
<th>message</th>
<th>sender</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host problem</td>
<td>Content</td>
<td>Blocked</td>
</tr>
<tr>
<td>Smtp problem</td>
<td>Spam</td>
<td>Configuration</td>
</tr>
<tr>
<td>mailbox</td>
<td>Virus</td>
<td>Volume</td>
</tr>
<tr>
<td>Undeliverable</td>
<td></td>
<td>Reputation</td>
</tr>
<tr>
<td>Inactive</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Invalid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Figure 5.11 Classification schema, used to annotate bounces and to train the bounce classification algorithm, for a detailed description see Appendix D*

**Bounce Failure Annotation**

**Classifying bounces manually**

The objective was to retrieve a set of bounce failures that could be used for manual annotation and that extensively covered all possible bounce failures. The set of data used for general bounce analysis and described earlier, was not old enough, and was further extended by a sample extracted from historical data collected since the beginning of 2005. Another problem arose when unique bounce cases were selected. As we can see in the table of bounce classes Appendix D, bounce failures often contain information about the email address, the server it was sent from, ids, dates, etc. Thus using a simple distinct operator will not yield the desired result unless all these pieces of information are replaced or removed. Since it was desirable to not lose any snippet of information, it was decided to first anonymize the data. The difference between a real bounce failure and an anonymized bounce failure is depicted below.

```
550 5.1.1 <user@example.edu>... User account expired
550 5.1.1 #EMAIL-ADDRESS#... #NAME# account expired
550 5.7.1 Service unavailable; Client host [1.1.1.1] blocked using cbl.abuseat.org; Blocked - see http://cbl.abuseat.org/lookup.cgi?ip=1.1.1.1
550 5.7.1 Service unavailable; Client host #IP# blocked using #DOMAIN#; Blocked - see #URI#
```

The full set of older failures collected since 2005 was larger than 500 million cases. To anonymize that large of a set would be useless, so only a sample of one million cases were selected. As the data reside in an Oracle database, pseudo random generator `DBMS_RANDOM` package with large seed initialization was used. All new failures (1.5 million) from the new set were used. Afterward, both sets were anonymized and sorted according to their occurrence. The top two thousand failures from each set were selected (in total 4k cases).
After the set was ready for manual annotation, it was assigned to two annotators with expert knowledge in email marketing systems architecture and SMTP communication, who were also trained on how to use the classification framework. Each person annotated both sets, without knowing the classification of the other annotator. After both sets were classified by both parties, the accuracy (5.8) and the Cohen's Kappa indicators [26] were calculated. As part of the annotation process, a new "Unknown" class was defined and used when the failure description did not indicate the failure reason (these were mostly caused by parsing issues in the Email Parser, which affected about 5% of all failures).

In Figure 5.14 we can see the result of the annotated match and mutual agreement on the class level and class-category level. The mutual agreement on the new set is much higher than on the old set. Although both sets contained about 2k failures, it was discovered that the newer set was much smaller in terms of unique failure description. The problem lied in the parser which truncated the bounce failures to 200 characters and words, cut in the middle, caused the undesired uniqueness. Older data were then selected from a much larger base so this effect was not that significant. To annotate the new set took about 8 hours, in comparison to the old set which required about 20 hours.
Although Kappa around 0.70 is not excellent, this result of a human to human comparison was set as a goal for a human to machine agreement. If achieved, it would be considered a great success. Before this manually annotated list was used as a golden standard for the automated classification, it was necessary to review the mismatched cases and achieve a 100% agreement. Both annotators discussed on each case that did not match the reason, and agreed on the correct categorization.

Set of old bounces

<table>
<thead>
<tr>
<th>Annotator B</th>
<th>Domain</th>
<th>Mailbox</th>
<th>Message</th>
<th>Sender</th>
<th>Unknown</th>
<th>B case dist.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>72</td>
<td>16</td>
<td>4</td>
<td>5</td>
<td>4.79%</td>
<td></td>
</tr>
<tr>
<td>Mailbox</td>
<td>26</td>
<td>1138</td>
<td>2</td>
<td>15</td>
<td>59.92%</td>
<td></td>
</tr>
<tr>
<td>Message</td>
<td>6</td>
<td>12</td>
<td>109</td>
<td>20</td>
<td>8.00%</td>
<td></td>
</tr>
<tr>
<td>Sender</td>
<td>19</td>
<td>41</td>
<td>11</td>
<td>247</td>
<td>16.73%</td>
<td></td>
</tr>
<tr>
<td>Unknown</td>
<td>36</td>
<td>46</td>
<td>2</td>
<td>128</td>
<td>10.56%</td>
<td></td>
</tr>
<tr>
<td>A case dist.</td>
<td>7.85%</td>
<td>61.85%</td>
<td>6.12%</td>
<td>14.22%</td>
<td>9.97%</td>
<td>2026</td>
</tr>
</tbody>
</table>

Class level agreement

<table>
<thead>
<tr>
<th>Class</th>
<th>Observed agreement</th>
<th>Chance agreement</th>
<th>Kappa</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>83.61%</td>
<td>0.41</td>
<td>0.72</td>
</tr>
</tbody>
</table>

Class-Category level agreement

<table>
<thead>
<tr>
<th>Class</th>
<th>Observed agreement</th>
<th>Chance agreement</th>
<th>Kappa</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>62.35%</td>
<td>0.17</td>
<td>0.54</td>
</tr>
</tbody>
</table>

Figure 5.14a Table containing cross annotator agreements on the old training data set

Set of new bounces

<table>
<thead>
<tr>
<th>Annotator B</th>
<th>Domain</th>
<th>Mailbox</th>
<th>Message</th>
<th>Sender</th>
<th>Unknown</th>
<th>B case dist.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>49</td>
<td>18</td>
<td>0</td>
<td>6</td>
<td>5</td>
<td>3.96%</td>
</tr>
<tr>
<td>Mailbox</td>
<td>44</td>
<td>1130</td>
<td>3</td>
<td>36</td>
<td>94</td>
<td>66.41%</td>
</tr>
<tr>
<td>Message</td>
<td>3</td>
<td>1</td>
<td>159</td>
<td>10</td>
<td>4</td>
<td>8.99%</td>
</tr>
<tr>
<td>Sender</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>222</td>
<td>1</td>
<td>11.69%</td>
</tr>
<tr>
<td>Unknown</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>169</td>
<td>8.94%</td>
</tr>
<tr>
<td>A case dist.</td>
<td>5.23%</td>
<td>58.54%</td>
<td>8.28%</td>
<td>14.08%</td>
<td>13.87%</td>
<td>1968</td>
</tr>
</tbody>
</table>

Class level agreement

<table>
<thead>
<tr>
<th>Class</th>
<th>Observed agreement</th>
<th>Chance agreement</th>
<th>Kappa</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>87.86%</td>
<td>0.43</td>
<td>0.79</td>
</tr>
</tbody>
</table>

Class-Category level agreement

<table>
<thead>
<tr>
<th>Class</th>
<th>Observed agreement</th>
<th>Chance agreement</th>
<th>Kappa</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>77.13%</td>
<td>0.24</td>
<td>0.70</td>
</tr>
</tbody>
</table>

Figure 5.14b Tables containing cross annotator agreements on the new training data set

Most of the problems were caused by the tiredness of the annotators, where obvious failure was misclassified. Others were due to disagreement on the classification framework. For instance one annotator classified failures under the Mailbox-Invalid category only if it was clear from the failure description that the mailbox did not exist and if this was not specific it was categorized as Mailbox-Undeliverable. The other annotator categorized failures as undeliverable only if it was clear from the text that the mailbox exists, but because of some restriction cannot be reached.
Yet another problem why the classification differed was caused by ambiguous description of the bounce failure.

550 sorry, no mailbox here by that name or inactive user
550 5.7.1 Your mail server has been marked as sending spam or IP name possibly forged

For instance, in this example above the first case is ambiguous because it is not clear whether it belongs to the Invalid or Inactive category (was reclassified as generic problem under Undeliverable category). In the second case it is either a Reputation or Configuration issue (reclassified as Blocked). As in such case it is not possible to classify the bounce failure correctly, so it was assumed that even the automated algorithm will have difficulty and it was desirable to quantify this ambiguity, and define a more realistic threshold for the automated classification.

A third annotation exercise was conducted and the two annotators again classified a sample of cases from already annotated and a reconciled list of failures. The final agreement statistics were taken as the maximum theoretical agreement that could be achieved by the classification algorithm.

<table>
<thead>
<tr>
<th>Class level agreement</th>
<th>Class-Category level agreement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observed agreement</td>
<td>93.76%</td>
</tr>
<tr>
<td>Chance agreement</td>
<td>0.44</td>
</tr>
<tr>
<td>Kappa</td>
<td>0.89</td>
</tr>
<tr>
<td>Observed agreement</td>
<td>86.89%</td>
</tr>
<tr>
<td>Chance agreement</td>
<td>0.26</td>
</tr>
<tr>
<td>Kappa</td>
<td>0.82</td>
</tr>
</tbody>
</table>

Figure 5.15 Table with human to human classification agreement on reconciled list

Proportion of bounce failures

As expected from the hard bounce estimate, the predominant failure is Mailbox-Invalid. Some other failures, like Message-Content or Message-Virus, are very low and maybe too granular. A virus failure is very easy to detect as it always contains the term "virus" in the failure description, so I decided to keep it although its value is disputable. The Content failure is a generic category for any problem with the message. Although there are real failures that distinguish between the Spam and the Content category, it might be better to merge the two, because the corrective action in both cases is alteration of the content.
Text processing
This section describes the core functionality of the Natural Language Processing (NLP) engine and how it is used by the Bounce Classifier. NLP might be too noble of a name for what the component is doing. Techniques used by this component do not differ much from real NLP solutions, however the complexity of the bounce failure domain, and its semantic space is much smaller. If we imagine that our solution can understand only thirteen various meanings from a limited corpus, it is in terms of complexity different than what the real NLP is about. The amount of various meanings this component can understand was on purpose limited to only thirteen cases, however if trained on a larger scale it could detect more failures than what RFC 1893 defines.

Another reason why we are not talking about a real NLP, is because bounce failures, although aimed for humans, are semi-generated by machines and are lacking the ambiguity of natural language.

In the previous section we mentioned that the text was anonymized, and concrete objects were replaced by their class names. This process was not described in detail, although it was not a simple task. The inspiration of how to automatically anonymize the data can be attributed to the GATE [27] tool. The initial annotation of 4k failure descriptions for manual classification was fully accomplished by GATE, however for my purposes of automated annotation of millions of failures it was not scalable. Rather than spending time on integration issues, I decided to develop an ORACLE PLSQL package which did the task. It was not complicated as the corpus of bounce failures is very limited.

Tokenizer
In ANNIE [27], the tokenizer is the first component in the pipeline of many connected packages. In our case its responsibility is to process the failure description (outcome of the Email Parser) and break it down to tokens. It was originally planned to use different tokenizers for each language and experiments with the N-Gram based language detection algorithms [11] were
conducted. This was later omitted because of the time it would require to develop a tokenizer for each language and focused was made only on the English language as it predominates.

The standard NLP tokenizer is detecting the token types. Because the bounce problem domain is simple, detection of additional types (this is usually accomplished by other modules) were incorporated into its logic. For instance, the GATE tokenizer is distinguishing between Word, Number, Symbol, Punctuation and Space tokens. The bounce failure tokenizer is in addition to those detecting Date, Time, Email address, Domain, URI, IP and Failure Status Code.

![Table with tokens for a single bounce failure](image)

<table>
<thead>
<tr>
<th>id</th>
<th>text</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>553</td>
<td>CODE</td>
</tr>
<tr>
<td>2</td>
<td>5.3.0</td>
<td>CODE</td>
</tr>
<tr>
<td>3</td>
<td><a href="mailto:mailbox@domain.com">mailbox@domain.com</a></td>
<td>EMAIL_ADDRESS</td>
</tr>
<tr>
<td>4</td>
<td>Message</td>
<td>WORD</td>
</tr>
<tr>
<td>5</td>
<td>from</td>
<td>WORD</td>
</tr>
<tr>
<td>6</td>
<td>73.112.170.20</td>
<td>IP</td>
</tr>
<tr>
<td>7</td>
<td>rejected</td>
<td>WORD</td>
</tr>
<tr>
<td>8</td>
<td>see</td>
<td>WORD</td>
</tr>
<tr>
<td>9</td>
<td><a href="http://njabl.org/">http://njabl.org/</a></td>
<td>URI</td>
</tr>
</tbody>
</table>

![List of most frequent terms in the corpus (stop words are excluded)](image)

<table>
<thead>
<tr>
<th>id</th>
<th>term</th>
<th>id</th>
<th>term</th>
<th>id</th>
<th>term</th>
<th>id</th>
<th>term</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>550</td>
<td>11</td>
<td>error</td>
<td>21</td>
<td>5.7.1</td>
<td>31</td>
<td>host</td>
</tr>
<tr>
<td>2</td>
<td>user</td>
<td>12</td>
<td>554</td>
<td>22</td>
<td>name</td>
<td>32</td>
<td>accepting</td>
</tr>
<tr>
<td>3</td>
<td>mailbox</td>
<td>13</td>
<td>reply</td>
<td>23</td>
<td>5.1.1</td>
<td>33</td>
<td>following</td>
</tr>
<tr>
<td>4</td>
<td>5,2.2</td>
<td>14</td>
<td>unknown</td>
<td>24</td>
<td>doesn’t</td>
<td>34</td>
<td>local</td>
</tr>
<tr>
<td>5</td>
<td>status</td>
<td>15</td>
<td>rctp</td>
<td>25</td>
<td>original</td>
<td>35</td>
<td>recipients</td>
</tr>
<tr>
<td>6</td>
<td>full</td>
<td>16</td>
<td>command</td>
<td>26</td>
<td>recognized</td>
<td>36</td>
<td>sender</td>
</tr>
<tr>
<td>7</td>
<td>552</td>
<td>17</td>
<td>rejected</td>
<td>27</td>
<td>5.2.1</td>
<td>37</td>
<td>unavailable</td>
</tr>
<tr>
<td>8</td>
<td>recipient</td>
<td>18</td>
<td>delivery</td>
<td>28</td>
<td>failed</td>
<td>38</td>
<td>permanent</td>
</tr>
<tr>
<td>9</td>
<td>address</td>
<td>19</td>
<td>mail</td>
<td>29</td>
<td>such</td>
<td>39</td>
<td>smtp</td>
</tr>
<tr>
<td>10</td>
<td>message</td>
<td>20</td>
<td>account</td>
<td>30</td>
<td>please</td>
<td>40</td>
<td>again</td>
</tr>
</tbody>
</table>

**Stemmer, Lemmatizer**

In some cases it is suitable to reduce the variability of tokens, especially for languages where terms vary. In our case the number of observed cases is so large that variation of terms shall not cause trouble. These two processes are not used.

**Transducer, POS (Part of Speech) Tagger**

The purpose of these modules in traditional NLP is to do lexical analysis of a sentence. As my approach is to use conventional statistical methods, lexical analysis is not needed.

**Gazetteer**

The purpose of this module is to lookup terms in various lists and to add additional information to them. My initial idea was to use an Ontological gazetteer which can search terms within ontology. A documented knowledge about the infrastructure of Email marketing systems could be very beneficial, especially when trying to identify the cause of the failure. For instance, if a new SMTP server is added, it will cause reputation failures because its IP is not yet trusted. If such knowledge is stored in and added to the ontology, an inference engine can discover that this...
change caused the increased rate of bounces. Although the development of Ontological gazetteer sounds challenging, it was realized that this approach could drastically complicate the whole assignment and decided to use a conventional Gazetteer with a standard set of lists.

- The first list contained stop words. At this point it was not sure if those will be used for the classification or not, because some of the recent trends [4], advocate not excluding them.
- The second list contained people names, and as these sometimes occurred in the bounce failures it was desirable to remove them.
- The third list contained internal and external IP addresses of all the SMTP servers used in the system. This turned out to be very significant for Sender-Reputation failures.
- The fourth, and last, list contained commands used in the SMTP protocol.

<table>
<thead>
<tr>
<th>id</th>
<th>text</th>
<th>type</th>
<th>tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>553</td>
<td>CODE</td>
<td>FSC</td>
</tr>
<tr>
<td>2</td>
<td>5.3.0</td>
<td>CODE</td>
<td>EFSC</td>
</tr>
<tr>
<td>3</td>
<td><a href="mailto:mailbox@domain.com">mailbox@domain.com</a></td>
<td>EMAIL_ADDRESS</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Message</td>
<td>WORD</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>from</td>
<td>WORD</td>
<td>STOP</td>
</tr>
<tr>
<td>6</td>
<td>73.112.170.20</td>
<td>IP</td>
<td>OUR_EXTERNAL</td>
</tr>
<tr>
<td>7</td>
<td>rejected</td>
<td>WORD</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>see</td>
<td>WORD</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td><a href="http://njabl.org/">http://njabl.org/</a></td>
<td>URI</td>
<td>-</td>
</tr>
</tbody>
</table>

*Figure 5.20 Output of the gazetteer with tagged terms*

**Anonymizer**

The last element in the bounce failure processing pipeline is replacing specific terms with its type or tag, (the logic is in Figure 5.21). Elements that are unique in the corpus, if not removed, increase the vector space. This could have impact on the classification algorithm, and is as well computationally ineffective. The reason why those are replaced rather than removed is to keep the failure linguistically correct, and in the right order. Keeping the word order is not needed for the bag-of-words model, however at the time this component was design it was not certain what algorithm will be used, and the desire was to not lose their order. Another reason to keep the generalized terms, instead of removing them, was that they still contain some useful information.

```plaintext
450 4.0.0 #EMAIL_ADDRESS#... No such user in this domain
421 4.0.0 Intrusion prevention active for #IP_OUR_EXTERNAL#
```

For instance, in the first failure the problematic subject is the email address, and we can infer that the problem was with the mailbox. In the other case, the failure was caused by our server reputation.
After the anonymization, the order of the most frequently used terms is different and the EMAIL_ADDRESS, DOMAIN and URI types make it into the top 40 terms. These anonymized failures were then used to build the term matrix of bounce failures.

**Term document matrix (TDM)**

The Term Document Matrix is a rectangular structure where each column represents one term and row represents a document (in our case the failure). Term matrix defines the entire space, in which our failure vectors exist.

\[
Terms = \{term_1, \ldots, term_k\} \\
Failure \in (\mathbb{R}^+)^k \\
TDM = [Failure_1, \ldots, Failure]^T
\]

Bounce failures are tiny, just one or two sentences long and not very poetic. The number of terms used (even over all languages) is very small. In our case, the cardinality of terms is much smaller than the cardinality of the bounce failures (the corpus size). In the sample consisting of one million bounce failures there were over 120k terms used, however only two thousand terms occurred in more than five failures. As the term distribution over corpus does have a heavy-tail, and operations on high dimensional matrix are computationally problematic, we need to find a solution how to select the right terms into the matrix. It is important to reduce the matrix sparseness but keep the failure dissimilarity. As the plan is to use a supervised learning technique, it is not worthy to go beyond the list of terms contained in the training set of 4k failures.

**Controlled dictionary**

One technique that limits the number of terms in the TDM is called controlled dictionary. It is a list of terms that can be used to filter the terms coming from bounce failures. To build this dictionary the terms from the training set were ordered based on their \( \text{idf} \) (inverse document frequency) [1], which shows the potential of the term to separate documents. After reviewing the top two thousand terms, which occurred more often than five times, 560 terms was selected into the control dictionary. The other terms were mostly parsing errors. As it was unclear if such a small number could produce any reliable results, it was decided to continue with two matrixes. One matrix was limited to 560 terms TMD and the second to all 58k terms TDM\(_{full}\).
Term document matrix weighting
The matrix dimensions are set. The next step is to focus on the values it will contain.

\[ Failure_j = b_j f_j x, \quad j = 1 \ldots k \] (5.11)

There are various approaches as to how the values shall be calculated and the most referenced one is the \( tf-idf \) weighting. Collection frequency component \( idf \), mentioned earlier, describes the discriminative potential of the term. A term which occurs on each failure has very low potential and consequently low \( idf \). The second component, \( tf \) (term frequency), determines how important the term is for the document. In our case, where the document is very short, it is unusual that the term will occur more frequently than once, so if it occurs it will be 1 and if not it will be 0. Sometimes a normalization length component is used. In our case the failure lengths do not vary that much, so it is ignored.

**Term frequency component**

\[ b_j = \begin{cases} 1 & \text{Failure contains term}_j, \\ 0 & \text{otherwise}. \end{cases} \] (5.12)

**Collection frequency component**

\[ f_j = 1 \] (5.13)

\[ f'_j = \log \left( \frac{l}{\sum_{j=1} b_j} \right) \] (5.14)

**Normalization component**

\[ x_j = 1 \] (5.15)

As it might be interesting to see how the weighting can influence the classification algorithm, two different weighting scenarios will be used for the collection component. The first used was Coordination weighting \( \text{TDM}_{\text{bxx}} \), which used no weighting for the collection frequency (5.13) and the second was a Classical weighting \( \text{TDM}_{\text{bfx}} \), using the \( idf \) version (5.14).

**Latent semantic analysis (LSA)**

Some algorithms do have a problem with large dimensionality of a vector space. One problem might be the sparseness of such matrix and the other the calculation performance. This is not an isolated problem in text mining as it occurs in other data mining domains as well. There are also various approaches on how to reduce the feature space. A compression method suitable for our homogenous corpus is a technique called Singular Value Decomposition (SVD), which is a core fundamental of the Latent Semantic Analysis [4] technique. With decent hardware and a good algorithm, it is possible to decompose the term matrix in reasonable amount of time. It was decided to use the R [28] wrapper for LAPACK implementation of the algorithm [29]. SVD decomposes the transposed term document matrix into three matrices.
Where $U$ and $V$ are unitary matrixes and $\Sigma$ is a diagonal matrix of ordered singular values. The number of singular values selected determines the compression ratio, and removes the noise from the original matrix.

According to tests in [4] the best results were achieved if the $k$ number of largest singular values was between 150 and 200. For our case the first 200 singular values will be used.

The main reason why LSA was favored was because of its potential to work with terms that did not occur in the training set. As our training set is small in comparison to the number of failures that will need to get classified, it will certainly happen than some terms will not make it into the training set, although they could be very good indicators of the bounce failures. This is because of the LSA feature to identify synonyms. For instance, if in the training set only a failure “This mailbox is invalid” will occur and in the entire corpus there will be a semantically similar failure “This user is invalid”, LSA will treat mailbox and user as synonyms and the classification algorithm will not have a problem to classify the user correctly although it did not see it in the training data. Unfortunately, the personal computer (Win XP 32 bit) used to calculate the SVD was limited in memory, and I did not succeed to calculate the SVD for the entire corpus. For the experiment only, the training matrix was decomposed. The potential to discover the hidden semantics will be thus drastically reduced and LSA will work only as a compression algorithm, which can have effect on the performance. Two additional term matrixes on the training data set were calculated $LSA_{bbox}$ and $LSA_{bths}$.

**Classifying algorithms**

**Failure lengths**
Before we describe how the various term vector matrixes were tested and the right model selected, a small reflection on data that were available might be practical. While the failure descriptions are system generated messages, it is not likely that all SMTP servers use an identical
failure description for each concrete failure. On the other hand, it is reasonable to assume that failure descriptions will be similar. This is why the traditional text classification technique, based on similarity detection, was selected.

There are two reasons why a semantically identical bounce failure looks different. The first is the information about concrete entities (email address, IP, domain, etc.) which are included in the failure description and the second is the dialect of the server that generates it. The former was addressed by the anonymization module, while the latter remains unresolved.

The assumption is that that each SMTP engine constructs the same failure description for the same problem. Knowing what engine generated the bounce message can simplify the failure recognition. For instance, the Qmail engine [43], for an invalid mailbox, is generating a 26 character long failure description (excluding the email address). Unless the administrator modifies it, it remains the same. As it is unlikely that other failure description generated by Qmail will have the same length, the failure classification could be based solely on the failure length and the SMTP engine.

The few large free email providers use proprietary SMTP engines and could be identified based on the email address domain, but information about what SMTP engine is used by smaller domain owners is hard to find. In the Figure 5.23, we can see that the length of failure description is indeed separating the bounce failures. For instance, thin spikes are failures of the same length describing the same failure type with no concrete values and are, most likely generated by the same engine (see Hotmail). Other failure descriptions that evoke a normal distribution shape are again covering one topic, however enriched by names of concrete entities involved in the failure (see Yahoo).

**Testing the classification algorithm**

An algorithm that can satisfy the classification needs must be fast enough and yield a good match result. Classification performance is simply measured as the number of bounces classified over a period of time, so it needs to match the number of bounces coming in. Correctness of the classification is a bit more challenging. For measurement of the cross-annotator match a Kappa index [26] was used. It is thus practical to use it again to compare the human x machine versus human x human results. Traditionally, for information retrieval and classification algorithms,
precision (5.6) and recall (5.7) are favored methods. Along with F-measure [37] which combines both. As the classes are not well balanced a macro weighted version of these measures will be used.

\[ P_{\text{macro}} = \frac{1}{|C|} \sum_{i=1}^{|C|} \frac{TP_i}{TP_i + FP_i} \]  

Precision, ratio of a correctly classified failures from all the failures classified into the category, averaged over all classes

\[ R_{\text{macro}} = \frac{1}{|C|} \sum_{i=1}^{|C|} \frac{TP_i}{TP_i + FN_i} \]  

Recall, ratio of classified failures from all that should be classified into the category

**Algorithms evaluation**

Three classification algorithms were chosen for comparison, SMO [38],[39], Naïve Bayes [40] and C4.5 [41]. The data mining engine used was WEKA [42] where each individual algorithm was implemented in the following classes

weka.classifiers.functions.SMO
weka.classifiers.bayes.NaiveBayesMultinomial

After the SMO algorithm was trained and applied to the annotated data, all misclassified failures were reviewed to understand why they were not classified correctly. It was a big surprise that about one third of all the differences were problems on the training data. From 403 mismatches, 128 were just badly annotated failures. So even though the training set was double annotated over 5% of errors remained undetected.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>TDM-type</th>
<th>Agreement</th>
<th>Kappa</th>
<th>P-macro</th>
<th>R-macro</th>
<th>F-macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMO</td>
<td>tdm-bxx-full</td>
<td>84.44%</td>
<td>0.7916</td>
<td>0.6967</td>
<td>0.6108</td>
<td>0.6509</td>
</tr>
<tr>
<td>SMO</td>
<td>tdm-bxx</td>
<td>83.61%</td>
<td>0.7838</td>
<td>0.6851</td>
<td>0.6499</td>
<td>0.6670</td>
</tr>
<tr>
<td>SMO</td>
<td>tdm-bfx</td>
<td>83.61%</td>
<td>0.7838</td>
<td>0.6851</td>
<td>0.6499</td>
<td>0.6670</td>
</tr>
<tr>
<td>SMO</td>
<td>lsa-bxx-full</td>
<td>80.95%</td>
<td>0.7455</td>
<td>0.6225</td>
<td>0.6071</td>
<td>0.6147</td>
</tr>
<tr>
<td>SMO</td>
<td>lsa-bxx</td>
<td>80.33%</td>
<td>0.7387</td>
<td>0.6333</td>
<td>0.5722</td>
<td>0.6012</td>
</tr>
<tr>
<td>SMO</td>
<td>lsa-bfx</td>
<td>82.82%</td>
<td>0.7729</td>
<td>0.6764</td>
<td>0.6358</td>
<td>0.6554</td>
</tr>
<tr>
<td>Naïve Bayes</td>
<td>tdm-bxx</td>
<td>77.78%</td>
<td>0.7035</td>
<td>0.5943</td>
<td>0.5043</td>
<td>0.5456</td>
</tr>
<tr>
<td>Naïve Bayes</td>
<td>tdm-bfx</td>
<td>78.63%</td>
<td>0.7225</td>
<td>0.6009</td>
<td>0.6161</td>
<td>0.6084</td>
</tr>
<tr>
<td>C4.5</td>
<td>tdm-bxx</td>
<td>77.88%</td>
<td>0.7185</td>
<td>0.6090</td>
<td>0.5455</td>
<td>0.5755</td>
</tr>
<tr>
<td>C4.5</td>
<td>tdm-bfx</td>
<td>76.44%</td>
<td>0.6832</td>
<td>0.5313</td>
<td>0.5507</td>
<td>0.5408</td>
</tr>
</tbody>
</table>

**Figure 5.24 Table of best performing algorithm over term matrix combinations**

As the training data set is not very large, a ten-fold cross validation was executed for each term vector matrix and classification algorithm combination. The selection of the algorithm was simple as SMO outperformed the others on all matrices and is considered the best algorithm for basic tm structure by others [44],[45],[46]. No time was devoted to tune the algorithm, and the default configuration with a polynomial kernel was used.
Selecting the classification algorithm
The best Kappa performance was achieved on the full, unweighted matrix $TDM_{bxx-full}$ on the other side the macro recall was lower. Message-Content was misclassified into Message-Spam and Sender-Volume into Sender-Configuration. This tells us that some terms that were excluded from the controlled dictionary were having a negative effect on the class separation and that the classes are maybe more granular than needed. Another bottleneck was the performance and memory consumption (full matrix was 58k terms, the limited only 560 terms). The most suitable matrix structure was the basic $TDM_{bxx}$ matrix.

A large expectation was placed on the LSA transformed matrixes, but since it was not calculated over the full corpus, its effect was minimized to the calculation performance.

The last evaluated parameter was the term matrix weighting. For the SMO algorithm, there was no difference between the weighted and unweighted matrix. The Weka engine was converting the weighted type into the binary form. The effect of the idf weighting was beneficial only for the Naïve Bayes algorithm. The overall performance of the three algorithms over various classes on the $TDM_{bxx}$ matrix is depicted in Figure 5.26.
Final results
The most suitable algorithm for the bounce classification engine is the SMO with a polynomial kernel and $\text{TDM}_{\text{ber}}$ matrix with binary weighting for the term component and no weighting for the collection component.

The final machine x human classification agreement results, based on the Kappa measure is depicted below. The result is almost identical to the human x human cross match.

<table>
<thead>
<tr>
<th>Class level agreement</th>
<th>Class-Category level agreement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observed agreement</td>
<td>Observed agreement</td>
</tr>
<tr>
<td>92.65%</td>
<td>83.61%</td>
</tr>
<tr>
<td>Chance agreement</td>
<td>Chance agreement</td>
</tr>
<tr>
<td>0.54</td>
<td>0.25</td>
</tr>
<tr>
<td>Kappa</td>
<td>Kappa</td>
</tr>
<tr>
<td>0.84</td>
<td>0.78</td>
</tr>
</tbody>
</table>

Figure 5.27 Table comparing Kappa results between the human and the SMO classification algorithm

Although it looks almost perfect, there is one caveat. Bounce failures that were not classified, because of a parsing issue or any other unknown reason (Unknown class), were excluded from the training set. As the SMO algorithm was not trained on them, such could not be predicted. It is thus possible that the classification agreement would be lower if those were included.

Structures to process bounces - Part 2
The Bounce classifier wraps the remaining part of the bounce failure processing environment. It is connected to the other elements through the failure and incident interface. The bounce classifier failure interface provides the Email Matcher with failureInfo. The bounce classifier incident interface is used for messaging purposes, through which systems can subscribe to notifications about sender blocks, spam related or other issues.

Figure 5.28 Component diagram of the Bounce classifier
Controller
The NLP Engine that was covered earlier calculates the termVector for each bounce failure. The Controller applies the weights to it and sends it into the Classifier. After the failure is classified, it is saved with other information into the database. For the testing environment, Controller is implemented as a package in the statistical software R [28]. The benefit of R is its rich set of mathematical libraries for linear algebra, which is needed for term matrix transformations. Since it was later proved that the simple bxx weighting is sufficient, there is no need to use it in the production environment.

Incident Evaluation Engine
This component monitors the delivery ratio. If problematic behavior is detected, it informs other systems. The other systems could be the Message generator, which would stop sending the marketing campaign so that too many spam related complains are not received. Its incident detecting potential can be based on simple statistical evaluation of variance or a more sophisticated algorithm (Evaluation Model).

Database
All modules discussed so far were purely functional, and for each input a result was returned. In the validation section we mentioned two approaches for email address validation. One approach was based on the bounce failures post-mailing validation and the other based on pre-mailing validation. As both approaches are based on data provided by the bounce collector system, the database module is the source of such information. The Email Validator can be retrained on the aggregated structures of the historical data or it can use the history of individual email addresses. The minimum data that shall be stored are about the messages sent and failures collected. For a simplified version of the database model see Appendix E.
Chapter 6
Validation techniques

Processed bounce data
In the early stage of the bounce processing system, we discussed a simple model Figure 5.6, that visualized bounce transition between two states (bounced and delivered). This model raised more questions than answers, and its unpredictable behavior triggered the idea of creating an entirely new bounce classification framework. With the system now operational and able to classify bounce failures, it is possible to look at the bounce behavior of each individual component and how they evolved over time.

Figure 6.1 Model of bounce transitions, visualizing only the prior probabilities and probability to stay in the same failure state. Other transitions are hidden.

The new model distinguishes between four bounce classes with thirteen bounce categories and two delivery states. In the Figure 6.1, we see the initial distribution of probabilities after the first message is sent and the probabilities of remaining in the initial state (the full set of all transitions is much larger and is hidden from this model). The difference between the Message, Sender and Domain, Mailbox states is interesting. A bounce is more likely to leave the earlier two and stay in the later states. It is not visible in the figure, but the average transition from Message -> Delivered is 73% with a standard deviation (SD) of 12.6% and the Sender->Delivered is 71% with SD of 1.7%. This supports the idea that the Message and Sender failures are independent of an email address and are related to the content sent.
On the other hand, the Mailbox type failures are a strong indication of invalid email addresses. The likelihood to remain in the bounced state is not 100% because Mailbox-Full → Responded or Mailbox-Inactive → Responded transitions do occur (the other reason is missing bounces). Let us consider the bounce sequence of a single email address that starts and terminates with a Mailbox-Invalid. Since the transition from Mailbox-Invalid → Delivered is illogical, any missing bounce within that sequence will still be Mailbox-Invalid. When analyzing such sequences an average missing rate of 4% was estimated. Since sequences which do not start or terminate with Mailbox-Invalid were ignored, this estimate is only an optimistic missing bounce rate. Further, as the bounce transition from Mailbox-Invalid → Mailbox-Invalid is stable and oscillates around 92% with SD of 2.6%, there is no room for such a high missing bounce ratio as presented in [15], where 25% of missing bounces were claimed.

Bounce transition Domain → Domain in Figure 6.3 is not flat as one would expect for a fatal bounce failure and changes with SD of 5.5%. This suggests that there is a mixture of multiple problems mixed together there. This will require further analysis and will be covered on detail domain analysis part.
Validation techniques - TLD

Verification techniques assure that a Top Level Domain (TLD) exists, because such are compared to the limited list published by ICANN. Since the likelihood that TLD routing will not work is minimal, the only aspect that shall be assured is that the TLD used in the email address is the right one. If we know that john works for "foo.bar" then his email address must not be from the .net domain. Thus, john@foo.bar is the valid email address and john@foo.net is invalid since it is using a TLD of a different entity. This, of course, is not the case for companies which own identical sub-domains within different TLD hierarchies, and allow the routing to the right mailbox. An invalid TLD can occur intentionally, but most of them just typos. For instance, an email address having the Oman (.om) or the Cameroon (.cm) TLDs exhibit unusually high bounce ratios. There is no reason why the bounce behavior of these particular TLDs shall be that much different to other TLDs so it must be caused by a typo. A similar issue is detectable for the .net domain where .et is used instead.

<table>
<thead>
<tr>
<th>tld</th>
<th>bounce ratio</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>.om</td>
<td>86.64%</td>
<td>0.00%</td>
</tr>
<tr>
<td>.et</td>
<td>84.57%</td>
<td>0.00%</td>
</tr>
<tr>
<td>.co</td>
<td>50.51%</td>
<td>29.61%</td>
</tr>
<tr>
<td>.cm</td>
<td>27.00%</td>
<td>48.01%</td>
</tr>
</tbody>
</table>

Figure 6.4 Table of mistyped TLDs, which are causing an unusually high host problem bounces.

Since the median bounce ratio of a domain type over all TLDs is 1.54%, it is reasonable that any domain with a bounce ratio of 80%, as depicted in the Figure 6.4, is most likely invalid. It is also notable that .co or .cm TLDs have an SMTP failure type. This is because some of the most popular domains from the .com TLD are registered in these two countries and the host servers respond. In the real global system such typos are a hundred times more frequent than valid email addresses of actual African (.co or .cm) users. This is drastically disfavoring valid email address from these countries, because the validation algorithm will almost certainly disable them.

Validation techniques - Domain

There is an abundance of problems that could occur on the full domain level (sub-domain with TLD). It is not a problem to store the domain information and measure bounce properties for a system that maintains hundreds of millions of email addresses. The prior-probability of bounce failure calculated on TLDs in the previous section can be thus handy only for new or unknown domains or for a validation which is not utilizing any historical data. In the real system used for the analysis, 150 million email addresses were only from 2.5 million domains and 90% of all email addresses were in the top 10 domains. To assure the domain is valid we could use operational or computational techniques. Operational techniques do use internet protocols and are very well known, so those will be covered briefly. The main focus will be put on computational techniques that use statistical methods.
Operational techniques
A domain, as an identifier of the organization, is responsible for the mailbox maintenance. Anyone who wants to receive an email message to the mailbox at a specific domain, must define what servers are going to process them. These servers are located in the DNS as mail exchanger (MX) records. The first operational test is to query the DNS for the MX records. After MX records are retrieved, usually in a form of a hostnames, the second test is to resolve their IPs and try to connect to the SMTP server. If that fails, another MX record is used. If at least one MX can be contacted, it means the domain is valid.

These tests look almost perfect. However, one issue is that failing to connect to the SMTP engine associated with the domain is not a strong indication of failure. Connectivity issues can be considered fatal only if they are permanent. A second issue is that, the inability to connect to the SMTP engine is not always caused by the target server, since connectivity problems can also be on the sender side. A third issue of these techniques is the performance. It is easily possible to consume all of the server connections when validating a list of dummy email address, while waiting for a response from the nonexistent servers. If these techniques are used, it is helpful to store the historical information about domains queried and use that data for a future reference.

A single attempt to connect to a domain does not tell anything about the quality of the domain. Let us consider the host issues recorded for one of the largest free email providers in the Czech Republic. Over the fourth quarter in 2010, five large Domain-Host incidents were recorded, Figure 6.5. If this domain would only be validated by an operational technique, on a day when the Host Problem occurred, many email addresses would be incorrectly disabled. It is interesting that such connectivity issues happen because Domain failures were originally considered as fatal. In this particular case it is unknown why the SMTP servers were not able to resolve the host, but it is evident that such incidents have no indication about the domain quality, since they occur on semi regular basis.

<table>
<thead>
<tr>
<th>sent at</th>
<th>bounce ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010-09-29</td>
<td>34.62%</td>
</tr>
<tr>
<td>2010-10-05</td>
<td>5.77%</td>
</tr>
<tr>
<td>2010-10-07</td>
<td>36.59%</td>
</tr>
<tr>
<td>2010-10-29</td>
<td>26.26%</td>
</tr>
<tr>
<td>2010-11-11</td>
<td>8.22%</td>
</tr>
<tr>
<td>2010-11-23</td>
<td>41.51%</td>
</tr>
</tbody>
</table>

Figure 6.5 Domain-Host delivery failures on a free email provider seznam.cz.
As messages were routed from Nord America to Europe, it is disputable what caused the connectivity issue.

Another reason why operational techniques are superseded by the Bounce collector is that they were needed to run over a longer period of time to minimize the random domain failures. A system that sends millions of email messages a day, may process bounces from about 50 thousand domains, which provides better statistics than what could be achieved with operational tests.
Computational techniques

Bounce patterns
Since there are both invalid domains (a permanent operational issue) and domains with temporary operational issues, it is essential to find a method to differentiate them. Domains with a fatal deliverability issue are mostly caused by typos, but since some try to exploit them using Typosquatting [12], it often happens that even invalid domains are operational. If a mistyped domain is not registered and someone registers it later, then Domain-Host failures changes to Mailbox-Invalid failures. Registered mistyped domains can be detected because the ratio of Mailbox failures is unusually high. Domains that work as a redirect or are accepting everything are suspicious too, because there are no bounces at all. In Figure 6.6, the red line represents the bounce ratio of the Domain type bounces. The first 100 domains are guaranteed to be mistyped domains, where the SMTP server is not active. The Mailbox line shows the ratio for mistyped domains where the SMTP is active, but not accepting the messages. There are about 50 of such domains.

![Chart showing bounce behavior of mistyped domains, ordered in decreasing order according to their bounce ratio of given Bounce class.](image)

The Sender bounce failures depicted on the chart are only for demonstrational purposes to accent the difference between the bounce types. As mentioned earlier, there is no doubt that there are also typo-squatted domains which accept all messages and may use them for some purpose. Although being absolutely perfect might under certain situations look weird, there are many real SMTP servers which do not respond if failures happen. It is thus hard to distinguish these two from each other.

Domain similarity
Another technique to detect mistyped domains is to use a similarity detection algorithm. The edit distance (Levenshtein distance) [58] is a simple algorithm that calculates the distance between two strings based on the minimum number of operations needed to transform the original string into the other one. As most of the domain typos exhibit a character mismatch or its omittance, the edit distance is pretty small, and the mistyped domains are easy to find, given the original domain. Calculating the similarity between all domains will require a huge matrix, so only domains with domain type bounce ratio over 40% were selected for detailed analysis. In the Figure 6.7 mistyped domains for yahoo, gmail, and hotmail are shown.
An assumption is that business and personal mailboxes have a different behavior over time. One difference is that a business mailbox is tied to the employment of the individual person. So if you change your employer, your mailbox is deleted. The same applies to student mailboxes assigned by universities. It would be thus interesting to elaborate such behavior and build a model that will use the domain type as a quality evaluation attribute. As there is no free register that has the information about domain types, such analysis will require domain crawling and domain classification. This endeavor was originally started, however it turned out that it is not that simple task as there are more domain types besides just corporate or free based domains. There are domains acting as email aggregators or email redirects, personal domains owned by a single person and many others, as yet unknown. As this turned out to be a time consuming project, it was not completed and a simpler approach was searched. A suitable indirect definition of the domain classification is the domain size (number of mailboxes registered). Free email servers usually have millions of more email addresses than businesses do have employees. It is thus possible to say that the more mailboxes the domain owns the more likely it is a free web server. Although the fluctuation of mailboxes is a property of a domain, it is in fact measured as failure of the local-part (Mailbox-Invalid failure). Since it was proved that the bounce ratio of Mailbox-Invalid failures decreases with the domain size (Figure 6.8 below), we can assume that the die out behavior of an email address will be lower on the free email servers.
Mistyped domains and the domain size
The size of a domain was calculated identically as in the previous paragraph. The actual number of a domain size corresponds to the amount of mailboxes from the test data (see section about bounce collection) and not the full database. The threshold will thus need to be adjusted accordingly to the full set. This technique operates with the fact that mistyped domains will not be of a large size and will have an unusually high bounce ratio. The red dots in Figure 6.9 are proved invalid domains.

![Figure 6.9 Chart showing Valid and Invalid domains and their bounce ratio for domain failures. Domain size is expressed in number of mailboxes associated to the domain in the analyzed data set.](image)

Domain length
It was assumed that the domain length might have an effect on the domain bounce behavior, but this assumption was not supported by the test data.

Historical data
Analogically to the TLD, we can calculate the overall probability of a domain to cause a bounce failure. This could be more practical for incident detection rather than for email address validation. Knowing that email campaigns sent to yahoo.com yield an average bounce ratio of 0.5%, a sudden spike above this threshold means an incident

Validation techniques - Local-part
The local-part is the last element that can have an effect on the validity of an email address. It identifies the individual user in the domain. The structure of the local-part varies based on what domain it is created. Some users are restricted and do not have control about how their local-part will look. This is mostly the case of corporate domains, where the local-part is based on the people’s names. Various construction rules or structures are used, such as "firstname.lastname", "lastname" or the first character of the first name and last name. There are also generated local-parts, which are useless for marketing purposes, since they are not used by human users. How various attributes influence the quality of the local-part is visualized on a set of conditional plots. A valid email address has no bounce recorded after the first message and an invalid email address has a bounce of Mailbox-Invalid type.
Local-part length
This simple technique tells us how the local-part quality changes with its length. In general, longer local-parts are of worse quality. The reason is unknown but might be related to the fact that typing or remembering a longer email address is more complicated. Although the length has an influence on the bounce ratio, it is not widely applicable because too long local-parts are very rare.

![Figure 6.10 Effect of the local-part length to the email address validity](image)

Similarity to first/last name (FLS)
As mentioned earlier, corporate mailboxes shall exhibit a different die out pattern than the free ones. And as free email local-parts are not that restrictive, it might be possible to determine the mailbox type based on the similarity of the people names to their local-part. The Jaro-Winkler distance measure [57], calculates the similarity between two texts and standardizes the outcome on a scale between 0 and 1, where 0 indicates dissimilar and 1 indicates identical. As the process of the local-part construction is unknown, the similarity was calculated as the maximum of three various local_part constructions. (first x localpart, last x localpart, firstlast x localpart)

![Figure 6.11 Effect of the local-part similarity to the email address validity](image)

It is evident from the Figure 6.12 that the average similarity changes for various domain types, and is higher for educational domains. The conditional plot on Figure 6.11 shows that the higher the FLS, the less likely the email address is invalid. This somehow contradicts the assumed theory that local-parts with high FLS (corporate domain mailboxes) shall die out earlier than those where the FLS is lower (free emails).
<table>
<thead>
<tr>
<th>edu domains</th>
<th>free email domains</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>fls</td>
</tr>
<tr>
<td>delta.edu</td>
<td>99.94%</td>
</tr>
<tr>
<td>mail.adelphi.edu</td>
<td>99.67%</td>
</tr>
<tr>
<td>kings.edu</td>
<td>99.57%</td>
</tr>
<tr>
<td>students.mc.edu</td>
<td>99.36%</td>
</tr>
<tr>
<td>wagner.edu</td>
<td>98.70%</td>
</tr>
<tr>
<td>ecats.gcsu.edu</td>
<td>98.56%</td>
</tr>
<tr>
<td>mu.edu</td>
<td>98.56%</td>
</tr>
<tr>
<td>go.tarleton.edu</td>
<td>98.52%</td>
</tr>
<tr>
<td>mymail.champlain.edu</td>
<td>98.45%</td>
</tr>
</tbody>
</table>

Figure 6.12 Table showing FLS for educational and free email domains

Because local-parts with lower FLS are generating higher bounce rates immediately after the first message is sent, it could be that these were mistyped, and not describing the general bounce pattern of corporate domains. Another use for this technique could be the validation of an email address on a form where First Name and Last Name is provided. Such validation can warn the user if the FLS will be close but not 100% similar. Of course it will need to be more robust than the simple calculation used for the die out analysis. The local-part construction rules will need to be used, suffixes like number omitted, etc. Unfortunately, as the local-part is limited to ASCII characters (4.8), this techniques has no practical use on international systems where people names are provided in their native language.

<table>
<thead>
<tr>
<th>first name</th>
<th>last name</th>
<th>local-part</th>
<th>fls</th>
</tr>
</thead>
<tbody>
<tr>
<td>Юлия</td>
<td>Полякова</td>
<td>loontik</td>
<td>0.00%</td>
</tr>
<tr>
<td>Татьяна</td>
<td>Цакоева</td>
<td>tsakoeva</td>
<td>0.00%</td>
</tr>
<tr>
<td>Miluše</td>
<td>Žaloudková</td>
<td>mila.zaloudkova</td>
<td>79.71%</td>
</tr>
<tr>
<td>Aarón</td>
<td>González Orozco</td>
<td>aaronorozco_glez</td>
<td>79.67%</td>
</tr>
<tr>
<td>Jessica</td>
<td>Artibee</td>
<td>jartibee</td>
<td>95.83%</td>
</tr>
<tr>
<td>Leighton</td>
<td>Jeffrey</td>
<td>ljeffrey</td>
<td>95.83%</td>
</tr>
<tr>
<td>John</td>
<td>Smith</td>
<td>johnsmith</td>
<td>97.00%</td>
</tr>
<tr>
<td>Faye</td>
<td>Johnston</td>
<td>fayejohnston</td>
<td>100.00%</td>
</tr>
</tbody>
</table>

Figure 6.13 Effects of non ASCII characters in people names to FLS

N-Gram approach

Many web forms allow entry of an email address which is not verified by sending the user a message (see delivery test from verification techniques). Users, who want to quickly complete the form and do not care about the future email notification, usually provide an invalid email address. Typing valid data requires concentration and that is why invalid email addresses are just a sequence of random characters with the @ sign. As the sequence of mechanically typed characters do not follow the probability of a real text sequence, it is possible to find it. Decomposing the local-part to a sequence of characters, ngrams [11] and calculating how likely the characters in the sequence occur, adjacent unlikely characters indicates the unreal text. A sequence of repeating characters is not easily detectable with unigram and typing the same key is the easiest way a bigram will be used (trigram or higher n-grams might be longer than the entire local-part so there is no benefit to use them).

\[ \text{ng-local-part} = "\text{" local-part "}" \]  (6.1)

Since the local-part is not real text, it is suitable to build the sequence of probabilities on a valid email addresses rather than on a textual corpus. For the bi-gram calculation only email address
with a response to the second message sent were used. A lowercased version of local-part uses 40 characters (4.8), (4.12). To identify the leading and trailing characters, the local part was surrounded by the chevron brackets (6.1).

\[ \text{Chars} = \text{dot atom text} \]  \hspace{1cm} (6.2)

\[ Q_{i,k} = (q_{i,k}), \quad i = 1 ... \text{Bigrams}, \quad k = 1 ... \#\text{Chars} \]  \hspace{1cm} (6.3)

\[ \text{unlikely transition}_{i,k} = \begin{cases} 1 & q_{i,k} \leq 0.05, \\ 0 & \text{otherwise.} \end{cases} \]  \hspace{1cm} (6.4)

\[ \text{unlikely transitions(localpart)} = \sum_{i=1}^{\text{length(localpart)}} \text{unlikely transition}_{i,k} \]  \hspace{1cm} (6.5)

Over 100 000 valid local-parts were analyzed, and the result formed a rectangular transition matrix of $41^2$ Bigrams (rows) and 41 columns. Not all transitions occurred in the training data, and some bi-grams occurred only in few instances. Bi-grams which did not cover at least ten transitions were removed. Empty cells in the matrix are assumed to be unlikely and set to 0.05. Instead of calculating the entropy of the local-part, each unlikely transition (6.4) was counted and the sum of all unlikely transitions (6.5) was the desired measure. The maximum of invalid transitions was as high as the length of the local-part. Thus, the higher the count of invalid transitions meant that the email address was more likely to bounce.

![Diagram](conditional_density_plot.png)

**Figure 6.14** Effect of the sum of unlikely transitions to the email address validity

Since the data used for the bi-gram calculation, came from a source where users would like to receive notifications, the volume of unlikely local-parts is very low. The potential of this technique will be higher on email addresses collected from less trustworthy sources.

<table>
<thead>
<tr>
<th>local-part</th>
<th>unlikely transitions</th>
<th>length in characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>daniel</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>edie.morena</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>nina.messers</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>yankeekeo</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>ciraco.cinzia</td>
<td>4</td>
<td>13</td>
</tr>
<tr>
<td>elpricipiyoelfin46</td>
<td>9</td>
<td>18</td>
</tr>
<tr>
<td>pashwiniremugade.r_ar</td>
<td>9</td>
<td>20</td>
</tr>
<tr>
<td>priscilla.fernandez.mq17</td>
<td>10</td>
<td>24</td>
</tr>
<tr>
<td>wkh6n95f1901208</td>
<td>12</td>
<td>16</td>
</tr>
</tbody>
</table>

![Table](example_local_parts.png)

**Figure 6.15** Example of few local-parts ad their unlikely bi-gram transitions scores
Validation techniques - Email address

For the final validation of an email address two computational models, which utilize techniques mentioned earlier, will be developed. One model is suitable to test the email address prior the mailing, when no historical data about an individual email address is available. The second model will be used to disable the email address based on its bouncing history. For the evaluation purposes we define a valid email address as such where in the consecutive mailing no bounce is detected, and invalid where bounce is detected. Only the fatal bounces of either Mailbox or Domain type are considered.

Pre-mailing models

The main benefit of pre-mailing models is that they can validate email addresses even without the mailbox history. Since no bounce collector is required, the deployment is a single assembly. Further, since there is no need to retrieve the bouncing history from the database, the performance is limited only to CPU speed. On the other hand, since the bouncing information about TLDs, Domains and textual properties of the local-part can never yield accurate results as models with history, their validation potential is expected to be lower. A suitable application of these models is when no history is available, such as on web forms or with purchased email address datasets. Two versions of this model are introduced, one utilizing prior bounce probability of the TLD component and the second of the Domain component. The potential to detect an invalid email address is determined by the amount of classification errors created, Figure 5.3, and measured by (5.6), (5.7), (5.8).

Because the impact of both error types is, for simplification purposes treated equally (5.4), it is reasonable to balance the test list to have valid and invalid email addresses represented uniformly. A list consisting of 2000 valid and 2000 invalid email address was sampled from the original set of bounce data, and the email address validity was determined based on the bounces collected after the initial mailing. Since the idea is to demonstrate how validation techniques can be built, rather than to find the best validation technique, a simple C4.5 tree [41] with the default configuration will be used (weka.classifiers.trees.J48).

![Figure 6.16 Receiver Operating Characteristic and Cost Curve of TLD and Domain based models](image-url)
The performance of both pre-mailing models is obviously not outstanding, because email addresses used for the analysis come from a higher quality source, where users have a great desire to be contacted. This list contains mostly only mistyped email addresses. As there were no other lists of a reasonable size available for quality comparison, it is hard to determine the quality change between various email address sources.

**Post-mailing model**

A significant improvement of the local-part quality detection can be achieved with historical bounce data. Knowing what state the email address is at right now, and knowing how likely it will change states (6.9), it is possible to predict what will happen after the next message is sent. The purpose of this algorithm is to detect dead email addresses and disable them. To minimize the effect of missing bounces, it is necessary to calculate the state transitions on a set of email addresses that were provably valid (at least one response was detected, Figure 5.2). To avoid any seasonal patterns and to simulate more realistic scenarios, where email addresses are of a different age, a random historical length up to four messages into the past is retrieved. A history of four messages is sufficient because the prediction algorithm will utilize only up to the fourth order probability chain.

\[
\text{Delivered} = \{\text{Responded, Not responded}\} \quad (6.6)
\]

\[
\text{Bounced} = \{..., \text{list of all bounce failures, ...}\} \quad (6.7)
\]

\[
n = \#(\text{Delivered } \cup \text{ Bounced}) \quad (6.8)
\]

\[
P_{1st \text{ order chain}} = \begin{pmatrix} p_{i,j} \end{pmatrix}, \quad i, j = 1 \ldots n \quad (6.9)
\]

\[
P_{2nd \text{ order chain}} = \begin{pmatrix} p_{i,j;k} \end{pmatrix}, \quad (i, j) = 1 \ldots n^2, \quad k = 1 \ldots n \quad (6.10)
\]

Before the final model is presented it is worthy to show how the length of the history chain contributes to the future prediction. For this test, four random samples of a variable email address history chains were selected. The 1st order chain sample contained the result after the first message was sent and the 4th order chain sample contained only email addresses with a four message history. For all the samples, the next state was retrieved and treated as a control state.

**Figure 6.17 Validation potential of a simple pre-mailing model**

**Figure 6.18 Classification potential of the C4.5 tree based on the email address history.**

For an email address with a single message in the history the 1st order transition probabilities were used and so forth up to the four message history.
From Figure 6.18, it is clear that the longer the chain used, the better the results are predicted. On the other side, the difference is not that large and given the complexity to calculate and use the higher order chains, a simple 1st order chain should be sufficient. In addition to the information about the state transitions, the post-mailing validation model will use the same attributes as used in the pre-mailing model. For the model evaluation, a balanced set of randomly selected valid and invalid email address, with a total size of 4000 was used. The definition of valid and invalid remains the same, as for the pre-mailing models and again C4.5 decision tree is used.

![Figure 6.19 Receiver Operating Characteristic and Cost Curve of the post mailing model](image)

**Table 6.20 Validation potential of the post-mailing model**

<table>
<thead>
<tr>
<th>Chain Type</th>
<th>Kappa</th>
<th>Precision</th>
<th>Recall</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Final model</td>
<td>0.8806</td>
<td>0.8892</td>
<td>0.9997</td>
<td>0.9403</td>
</tr>
</tbody>
</table>

This model yields much better results than the pre-mailing version. The prediction based on the most frequent bounce failure Mailbox-Invalid is perfect, however some other bounce types do not have that much potential. A good example might be the Message-Virus or Message-Content bounce types. Since these are rare, their transition statistics are unreliable. Since these two bounce types and some others were already questioned earlier, it might be worthy to either extend the training base, or to simply ignore them.

**Email Validator**

Analogically to the verification techniques, validation techniques are implemented in the Email Validator. This component provides similar functionality, where for an input string (in this case verified email address) yields a real number between 0 and 1 which identifies the degree of email address validity.

![Figure 6.21 Schema of email address validator and its interface](image)
The result of this function depends on the type of deployment, where a validator with no history yields a lesser result than one which uses it. As it was originally decided that the impact of both error types Figure 5.3 will be treated equally, there is still room to change this. Validation models described earlier were invalidating the email address if the probability of being invalid was equal to or above 0.5. By adjusting this threshold one could influence the impact of misclassification. The higher the threshold, the less false positives will appear the lower the more false negatives. The detailed behavior of this module is described in an activity diagram which is in the Appendix C.
Chapter 7
Conclusion

With the exponential growth of data collected, the desire for automated information processing is gaining importance. This places subsequent significance on data quality assurance, because the outcome of this processing depends on the quality of the data and techniques used. To assist with the description of data quality and its implications, various data quality dimensions were presented. These dimensions often have ambiguous interpretations, and an attempt to clarify and organize them was made. A simple framework with two perspectives was applied and dimensions were categorized based on how subjective or objective they are and how precisely or imprecisely they can be assured. The precisely measurable quality dimensions were tested by Verification techniques and imprecisely measurable quality dimension by Validation techniques. Verification techniques are widely used in data quality assurance, while Validation techniques are not. Since an imprecise description of data quality by Validation techniques is better than nothing at all, it is advocated that Validation techniques are needed. Further suggestions were made as to how such techniques could be developed.

A practical example of Verification & Validation techniques is demonstrated on email address quality assurance. Standards covering email address specification are reconciled and adapted for the business needs of the email marketing system. This new specification allows for the creation of Verification techniques that utilize the syntax grammar and email address constraints. Because the rules of email address and domain construction are constantly changing and are dependent on local authorities, it is not possible to develop and maintain an all-encompassing specification. However, suggestions are made how to extend these rules and achieve a better and more comprehensive set of Verification techniques.

An innovative approach to data quality validation was proposed. It was also demonstrated how multidimensional classification algorithms can predict email address quality. Various attributes of an email address, domain or top level domain were evaluated and their contribution to validity was presented. The list of presented email address attributes is by no means complete, and its purpose is purely inspirational. Textual properties of email address local-part were researched and its length, characters used, bi-gram transitions and first name / last name similarities were evaluated. Similar techniques were used for domain evaluation. All of these data attributes, along with the bounce transition probabilities, were used to calculate email address quality. If an email address is predicted to be of high quality, it is treated as valid and kept, otherwise it is invalidated and deleted.

As with any prediction model there is room for improvement. Some of the models presented here have a weak prediction potential, while others are more promising. Besides the prediction algorithms that could be replaced, improvement can be achieved by better measurement techniques of email address messaging history or by utilizing other data that could be linked to an email address. For instance, it is believed that other domain properties will have a significant impact on email address quality prediction. Since the domain classification framework was not
completed due to the lack of available information and complexity, this effect remains unknown. Another uncertainty is that only email addresses from a single source were evaluated, and it would be interesting to see how the email address collection phase contributes to the email address quality.

It would not be possible to develop any of the email validation techniques without reliable information about message delivery failures. The major focus was therefore placed on the development of a robust and reliable bounce classification engine. This module uses the textual description of delivery failures and puts them into various classes and categories by using a Support Vector Machine algorithm. Although its prediction reliability beat expectations and was almost as good as a trained SMTP administrator, it can be still better. It is important to note that the parsing engine was omitting multiline failure descriptions (which affected about 5% of all failures), and the classification schema was overly granular in some categories.

The detection of Domain or Mailbox failures, which are related to the quality of an email address, was used to train the validation models and to disable invalid email addresses. Other bounce failures caused by the Sender or Messages, although detected and classified, were not used. As the email marketing engine is treated as a production process with the delivery ratio (3.10) quality measure, removing invalid email addresses has a positive effect on it. Improving the process by minimizing the Sender or Message bounces is not sufficiently elaborated and any improvement still depends on a human expert review of those two failure types. The classification engine does drastically reduce the human effort needed, since only these two types need manual review. It is surely possible to make these improvements even more sophisticated. An idea is that the system shall, based on the detected failures, either change the process automatically or suggest how the change could be accomplished. Few experiments were made with a failure ontology, which if connected to the inference engine, can propose solutions or describe the problem in context of recent changes. Despite the challenge of this idea, it was omitted because of complexity and time requirements.

The Verification & Validation framework is presented in a manner suitable for anyone else who would like to use it on their data entities or as part of a larger quality assurance effort on data warehouses or business intelligence systems. The accompanying technical documentation was provided to give an idea of how a system can be built and to demonstrate the technical obstacles that were resolved while developing verification and validation techniques for email address quality assurance.

It is hoped that this small contribution to the data quality assurance techniques will encourage others to start developing their own modules. If data validation, as presented here, will be used more actively, we will truly start moving towards our Golden Age of Information.
Bibliography


[4] Thomas K. Landauer (Editor), Danielle S. McNamara (Editor), Simon Dennis (Editor), Walter Kintsch (Editor) (2007); Handbook of Latent Semantic Analysis; University of Colorado Institute of Cognitive Science Series


[16] V1 Magazine (2007); Source of Truth, V1 Magazine, Promoting spatial design for a Sustainable Tomorrow


[19] Ryu Kyung-Seok, Park Joc-Seok, Park Jae-Hong; A data quality management maturity model


[21] E. Schmidt (2010); Techonomy conference, Lake Tahoe, CA

[22] E. Petrova, N. Malevris (1991); Rules and criteria when to stop testing a piece of software; Microelectronics Reliability 31(6); p.1211-1227


[25] KM. Schneider, Techniques For Improving the Performance of Naïve Bayes for Text Classification (2005); Lecture Notes in Computer Science, Volume 3406/2005, p.682-693

[26] Cohen, Jacob (1960); A coefficient of agreement for nominal scales, Educational and Psychological Measurement


[31] W. Shewhart (1986); Statistical Method from the Viewpoint of Quality Control: New York: Dover Publications 69


[52] Internet Assigned Numbers Authority (IANA); http://www.iana.org/

[53] Internet Corporation for Assigned Names and Numbers (ICANN); http://www.icann.org


[55] Rules of Registration and Use of Domain Names; Nominet;
http://www.nominet.org.uk/registrants/aboutdomainnames/rules/


[57] W. Winkler (1985); Preprocessing of Lists and String Comparison, in Record Linkage Techniques; Internal Revenue Service, Publication 1299 (2-86); p.181-187

[58] V. Levenshtein (1966); Binary codes capable of correcting deletions, insertions, and reversals; Soviet Physics Doklady 10: p.707–710
Appendices

Appendix A – ABNF Syntax of email address
ABNF syntax of an email address as required by the Email Marketing System. Verification parser or Regular expression verification formula can be generated based on this grammar.

\[
\begin{align*}
\text{address} & = \text{local-part} \ "@" \ \text{domain} \quad \text{;between 6 and 254 chars} \\
\text{local-part} & = 1^*\text{atext} \ "\".\" \ 1^*\text{atext}) \quad \text{;between 1 and 64 chars} \\
\text{atext} & = \text{ALPHA} / \text{DIGIT} / \"!\" / \"\#\" / \"\$\" / \"%\" / \"&\" \\
& / \"'\" / \"*\" / \"+\" / \"-\" / \".\" / \"/\" / \"=\" / \"?\" / \"^\" / \"_\" / \"`\" / \"{\" / \"|\" / \"}\" / \"~\" \\
\text{domain} & = 1^*126(\text{sub-domain } ",\" ) \ \text{tld} \quad \text{;between 4 and 252 chars} \\
\text{sub-domain} & = \text{ld-str} [\text{ldh-str}] \quad \text{;between 1 and 63 chars} \\
\text{tld} & = 1^*63(\text{ALPHA}) \quad \text{;controlled list} \\
\text{ld-str} & = \text{ALPHA} / \text{DIGIT} \\
\text{ldh-str} & = *( \text{ALPHA} / \text{DIGIT} / \".\" ) \ \text{ld-str} \\
\text{ALPHA} & = %x41-5A / %x61-7A \quad \text{; a-z, lower case only} \\
\text{DIGIT} & = %x30-39 \quad \text{; 0-9}
\end{align*}
\]
Appendix B – Email address verification (Activity diagram)
Activity diagram depicting steps in the email address verification process.
Appendix C – Email address validation (Activity diagram)

Activity diagram depicting steps in the email address validation process.

Email Address Validation

1. Validate
2. Break email-address into components
3. Do Operational Test
4. Use another MX record
5. Failed
6. Connect to the SMTP server
7. Save result
8. Computational tests
   - Calculate domain attributes
   - Retrieve domain history
   - Calculate local-part attributes
   - Retrieve email address history
9. Merge all attributes
10. Run validation model
11. Disable email address
### Appendix D – Bounce Classes

<table>
<thead>
<tr>
<th>failure class</th>
<th>failure category</th>
<th>failure description and some examples</th>
</tr>
</thead>
</table>
| Mailbox       | Undeliverable    | From the failure description it is not possible to determine what problem caused the delivery issue, or it is a specific problem that is not worthy to have its own category. It might be a mailbox level block or even invalid mailbox. This is a generic class for mailbox level problems.  
451 4.3.0 Message held for human verification before permitting delivery. For help, please quote incident ID 39489196. I'm not going to try again; this message has been in the queue too long.  
550 Blocked: Your email account has been blocked by this user. If you feel you've been blocked by mistake, contact the user you were sending to by an alternate method.  
550 Address rejected (#5.1.1) |
| Mailbox       | Inactive         | Mailbox was not yet activated or was already deactivated. There is still a theoretical chance it might work again in the future. The major reason why this class is separate is because there is evidence that it was once valid.  
550 <matt@retsnom.net>: Account Deactivated  
550 5.1.1 <keith@example.edu>: Keith account expired  
550 El usuario esta en estado: inactivo |
| Mailbox       | Invalid          | Mailbox is deleted or never existed on the domain. This is a clear hard bounce.  
511 5.1.1 unknown address or alias: lea@foo.bar  
550 <joyce@foo.com>: Recipient address rejected: User unknown in virtual mailbox table  
450 4.0.0 <bill@something.org>: No such user in this domain. |
| Mailbox       | Full             | Mailbox or the system is over its quota. Technically when someone cleans it can again receive messages.  
452 4.2.2 Over Quota Giving up on 1.1.1.1. I'm not going to try again; this message has been in the queue too long.  
550 <puneet@testing.com>: quota exceeded  
550 <dinesh@hello.de> Benutzer hat zu viele Mails auf dem Server |
| Domain        | Host problem     | This is a DNS or MX level error, the sending server is not able to find a target to open a connection to  
544 Unable to route to domain. Giving up on 1.1.1.1.  
550 Domain does not exist.  
452 We are sorry but our server is no longer accepting mail sent to the ourserver.com email domain. |
| Domain        | Smtp problem     | SMTP connection is established, but it is terminated or there is a problem in the communication, although the problem could be on either side it is classified under the target component.  
451 Temporary local problem - please try later  
550 Too many retries.  
550 Protocol violation |
| Message       | Content          | There is something wrong with the message, it is not correctly formatted or it is rejected by the server, because of unknown reasons  
571 Message Refused  
550 Message too large  
554 5.7.1 Blocked by policy: blacklisted URL in mail |
| Message       | Spam             | Message is classified as spam  
550 5.7.1 Blocked by SpamAssassin  
550 5.0.0 Your message may be spam. A copy was sent to quarantine for review.  
554 Sorry, message looks like SPAM to me :-( |
| Message       | Virus            | Messages contains a virus  
554 Rejected : virus found in mail  
554 Your email was rejected because it contains the Phishing.Heuristics.Email.SpoofedDomain virus  
550 Virus Detected; Content Rejected |
<table>
<thead>
<tr>
<th>Sender</th>
<th>Blocked</th>
<th>Server is not able to deliver the message, because of an unknown reason</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>550 Mail from route.monster.com is denied from host 1.1.1.1 SPF</td>
<td></td>
</tr>
<tr>
<td></td>
<td>550 Permanent Failure: banned</td>
<td></td>
</tr>
<tr>
<td></td>
<td>421 4.0.0 Intrusion prevention active for [1.1.1.1]</td>
<td></td>
</tr>
<tr>
<td>Sender</td>
<td>Configuration</td>
<td>Server is not correctly configured. There are actually more types in</td>
</tr>
<tr>
<td></td>
<td></td>
<td>this category, and it might very well be that someone else is trying</td>
</tr>
<tr>
<td></td>
<td></td>
<td>to impersonate the server and his message is rejected due to invalid</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DNS.</td>
</tr>
<tr>
<td></td>
<td>451 4.1.8 DNSerr Relaying temporarily denied. IP name forged for 1.1.1.1 (PTR and A records mismatch).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>550 authentication required for relay (#5.7.1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>550 Access Denied</td>
<td></td>
</tr>
<tr>
<td>Sender</td>
<td>Volume</td>
<td>Volume of bad emails, number of connections or bad recipients in row</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is reached. After the threshold is reached all messages are rejected.</td>
</tr>
<tr>
<td></td>
<td>550 Too many invalid recipients</td>
<td></td>
</tr>
<tr>
<td></td>
<td>554 Too many connections</td>
<td></td>
</tr>
<tr>
<td></td>
<td>452 Too many recipients received this hour</td>
<td></td>
</tr>
<tr>
<td>Sender</td>
<td>Reputation</td>
<td>Sender server is either rejected by external or local block list. It</td>
</tr>
<tr>
<td></td>
<td></td>
<td>takes time to establish a good reputation for an IP.</td>
</tr>
<tr>
<td></td>
<td>550 5.7.1 Service unavailable; Client host [1.1.1.1] blocked using cblabuseat.org; Blocked - see <a href="http://cblabuseat.org/lookup.cgi?ip=1.1.1.1">http://cblabuseat.org/lookup.cgi?ip=1.1.1.1</a></td>
<td></td>
</tr>
<tr>
<td></td>
<td>550 mail not accepted from blacklisted IP address</td>
<td></td>
</tr>
<tr>
<td></td>
<td>553 sorry, your mailserver is rejected by see <a href="http://spamcop.net">http://spamcop.net</a></td>
<td></td>
</tr>
</tbody>
</table>
Appendix E – Bounce collector tables

A simplified model of tables used for Bounce Collector Engine and its components

class Bounce Collector tables

Processing tables

HOSTS

«column»

HOST: VARCHAR2(255)
IP: VARCHAR2(15)
TLD: VARCHAR2(255)
SLD1: VARCHAR2(255)

WORDS_EN

«column»

TEXT: VARCHAR2(50)
T_TYPE: VARCHAR2(100)
T_TAG: VARCHAR2(100)

BOUNCE_TOKENS

«column»

*F_ID: VARCHAR2(18)
*T_ID: NUMBER(10)
T_TYPE: VARCHAR2(100)
T_TAG: VARCHAR2(100)

BOUNCE_CLASSES_OUT

«column»

*F_ID: VARCHAR2(18)
B_CLASS: VARCHAR2(100)
B_CAT: VARCHAR2(100)

BOUNCE_FAILURES

«column»

*F_ID: VARCHAR2(18)
*BOUNCED_AT: DATE
*RCPT_TO: VARCHAR2(255)
*MAIL_FROM: VARCHAR2(255)
FAILURE: VARCHAR2(2000)

EMAIL_ADDRESSES

DOMAINS

BOUNCE_TERMS

«column»

*TEXT: VARCHAR2(50)
T_TYPE: VARCHAR2(100)
T_TAG: VARCHAR2(100)

MESSAGES

BOUNCE_TERMS_IN

«column»

*F_ID: VARCHAR2(18)
*T_TERM: VARCHAR2(2000)
*T_FREQ: NUMBER(8,2)