INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

Mestrado em Engenharia Informática e de Computadores
(Perfil de Sistemas de Informação)

Learning Techniques for Automatic Email Message Tagging

TONY TAM
(Licenciado em Engenharia Informática e de Computadores)

Dissertação de natureza científica realizada para a obtenção do grau de Mestre em Engenharia Informática e de Computadores

Orientadores:
Mestre Artur Jorge Ferreira
Mestre André Ribeiro Lourenço

Júri:
Presidente: Mestre Vítor Jesus Sousa de Almeida
Vogais:
Doutor Bruno Emanuel da Graça Martins
Mestre Artur Jorge Ferreira
Mestre André Ribeiro Lourenço

Novembro de 2011

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

Mestrado em Engenharia Informática e de Computadores
(Perfil de Sistemas de Informação)

Learning Techniques for Automatic Email Message Tagging

TONY TAM
(Licenciado em Engenharia Informática e de Computadores)

Dissertação de natureza científica realizada para a obtenção do grau de Mestre em Engenharia Informática e de Computadores

Orientadores:
Mestre Artur Jorge Ferreira
Mestre André Ribeiro Lourenço

Júri:
Presidente: Mestre Vítor Jesus Sousa de Almeida
Vogais:
Doutor Bruno Emanuel da Graça Martins
Mestre Artur Jorge Ferreira
Mestre André Ribeiro Lourenço

Novembro de 2011

Acknowledgments

This thesis would not be possible if it were not for my supervisors. André and Artur have proven invaluable with their knowledge and help during the course of this work.

A big acknowledgement also goes to INSTICC, for their availability to turn this into something more.

Resumo

A organização automática de mensagens de correio electrónico é um desafio actual na área da aprendizagem automática. O número excessivo de mensagens afecta cada vez mais utilizadores, especialmente os que usam o correio electrónico como ferramenta de comunicação e trabalho. Esta tese aborda o problema da organização automática de mensagens de correio electrónico propondo uma solução que tem como objectivo a etiquetagem automática de mensagens.

A etiquetagem automática é feita com recurso às pastas de correio electrónico anteriormente criadas pelos utilizadores, tratando-as como etiquetas, e à sugestão de múltiplas etiquetas para cada mensagem (top-N). São estudadas várias técnicas de aprendizagem e os vários campos que compõe uma mensagem de correio electrónico são analisados de forma a determinar a sua adequação como elementos de classificação. O foco deste trabalho recai sobre os campos textuais (o assunto e o corpo das mensagens), estudando-se diferentes formas de representação, selecção de características e algoritmos de classificação. É ainda efectuada a avaliação dos campos de participantes através de algoritmos de classificação que os representam usando o modelo vectorial ou como um grafo. Os vários campos são combinados para classificação utilizando a técnica de combinação de classificadores Votação por Maioria.

Os testes são efectuados com um subconjunto de mensagens de correio electrónico da Enron e um conjunto de dados privados disponibilizados pelo *Institute for Systems and Technologies of Information, Control and Communication* (INSTICC). Estes conjuntos são analisados de forma a perceber as características dos dados. A avaliação do sistema é realizada através da percentagem de acerto dos classificadores. Os resultados obtidos apresentam melhorias significativas em comparação com os trabalhos relacionados.

Palavras Chave

aprendizagem automática, classificação, etiquetagem, mensagens de correio electrónico, categorização de texto

Abstract

Automatic organization of email messages is still a challenge in machine learning. The problem of "email overload", coined in 1998 by Whittaker et al, is presently affecting enterprise and power users. This thesis addresses automatic email organization by proposing a solution based on supervised learning algorithms that automatically labels email messages with tags.

We approach tagging using previously created user-folders as tags and top-N ranking classifier output. Learning techniques are reviewed and the different fields of an email message are analyzed for their suitability for classification. Special attention is given to the textual fields (subject and body), by studying and testing different representations, different feature selection methods and several classification algorithms. The participant fields are analyzed and evaluated using classification algorithms that work with the vector-space model and a graph based representation. The different email fields are combined for classification using the classifier combination technique of Majority Voting.

Experiments are done on a subset of the Enron Corpus and on a private data set from the Institute for Systems and Technologies of Information, Control and Communication (INSTICC). The data sets are extensively analyzed in order to understand the characteristics of the data. The evaluation of the system, using accuracy, shows great promise, with the experimental results presenting a significant improvement over related works.

Keywords

machine learning, text classification, email tagging, email foldering, text categorization, supervised learning

Contents

List of Figures

1

Introduction

Email is one of the oldest and most popular forms of electronic communication. These days, with the increase of the number of users on the web, email usage has also increased and diversified. While more than 20 years ago, the phenomenon of email overloading was starting to be observed, today, more and more people express their dissatisfaction [70] [71] [83] about the lack of time and tools to process growing volumes of messages.

1.1 Motivation and Context

Ever since the dawn of the web, email was present. And soon after, the first signs of email overload were reported.

In 1982, Peter J. Denning [25], at the time president of the Association for Computing Machinery (ACM), was concerned with the "electronic junk" that started to pile in his inbox. He described his "not unique situation" where he tipically received several dozens of messages per day, that required much of his time to "skim and dispatch". He stated that "It is clear that some attention must be paid to the processes of receiving information, and preventing unwanted reception."
After over a decade, in 1996, Whittaker et al [82] analyzed the problem of excessive messages in the inbox and coined it as "email overload". They described how email is now used for more than communication. The increase in the number of "netizens" (internet citizens) with the added use cases have effectively overloaded the mailboxes. Another important issue is the usage of folders for email archival and organization. They believed that there was no good solution, since folders were useless for retrieving messages and required high maintenance in order to keep the mailbox organized.
Whittaker et al [81] revisited the problems of email in 2005. This new study focused on the usage of email and the different approaches people have towards it. Email overload was and still is a concern.
Fisher et al. [35] also revisited Whittaker's original paper, in 2006. They examined a larger

sample of mailboxes in order to understand how users organize their email in the present as compared to the past. The most significant difference was the number of messages users had in their mailboxes (tenfold increase) and the increased use of folders for archiving.

Asserted the existence of an email overload problem, several solutions have been proposed. The classic solution has been to automatically archive the messages into folders [44]. Foldering is a specific application of the more generic task of classification, where classes are seen as folders. An alternative to foldering is a search-based approach [38]. Some approaches have also tried to classify messages into tasks/activities [26], while others have tried to prioritize the inbox [86].

The advent of Gmail introduced the concept of tags (or labels) into the domain of email. Tags are keywords assigned to an item for later retrieval, sorting or browsing. While they became popularized in the websites of the so called "social web" [24] [36] [77], with gmail, tags became an alternative to folders as a tool for organizing the mailboxes of users.

Gmail has also proven the utility of this feature. In a recent website design [63], they have observed that "the overall number of labels applied has increased substantially". Since Gmail's focus isn't tags, "there are still a substantial number of Gmail users who never use labels (...) because Gmail has always placed a strong emphasis on using search".

In this work, we take special interest in tags, focusing in automatic ways to assign tags to email messages as opposed to the manual way it is commonly done. The classic way of organizing email relies on the use of folders, which can be formalized as (1.1),

$$f : m \mapsto F \tag{1.1}$$

where f is a function that maps the email message m to a folder F. Email message tagging can be formalized as (1.2),

$$g : m \mapsto \{t_1, t_2, \ldots, t_n\} \tag{1.2}$$

where g is a function that maps the email message m to tags $\{t_1, t_2, \ldots, t_n\}$.

Reports on the percentage of people that archive messages in folders greatly vary. While Tang et al [73] report that even though users organize email in different ways, the majority still choses to archive some of their email, Yahoo researchers [48] claim that 70% of the service's users have never defined a single folder.
Folders are far from ideal, with some studies [17] finding some issues in the way they are implemented in email clients. Folders constrain how email can be organized, by allowing a single message to be filed under only one folder. According to Whittaker, users like to leave items in their inbox for task management and find it hard to archive them in folders. Maintaining an organized folder structure is also a time consuming task.

Tags solve these problems [63] because they can work as folders, support multiple labeling and can leave the items in the inbox. Tags also offer several different perspectives on the same item.

The task of assigning tags to messages is still done in a manual way. With this work, we aim to automatize such task, minimizing the needed effort. Our main source of inspiration is the field of text classification.

One of the first use cases of email classification was email spam filtering. The large volumes of spam messages triggered the study of how to deal with spam using automatic methods. For the email overload problem, the focus has been to approach the problem through automatic foldering. The work in the field has been somewhat scarce when compared to traditional text classification [52]. This is mainly attributed to the lack of public data for comparison. In recent years, the Enron email data set has been made available. Yang [46] [47] and Bekkerman [4] have evaluated the suitability of the data for email classification. Cselle [22] made an extensive study in the ways of organizing email, notably automatic foldering and topic detection and tracking. Others have tried to infer usage patterns through the analysis of the social groups. Several more have documented their experiments, that will be described in detail in section 1.3.

Other use cases can be envisioned for automatic tagging such as curating web pages for search [9] or tagging news articles [32]. NIST's topic detection and tracking series [59] tackled similar work.

In figure 1.1 we present the timeline of the related research identifying, in our opinion, the most important articles for our work. It is worth noticing that The research literature increased substantially after the introduction of the Enron Corpus, in 2004.

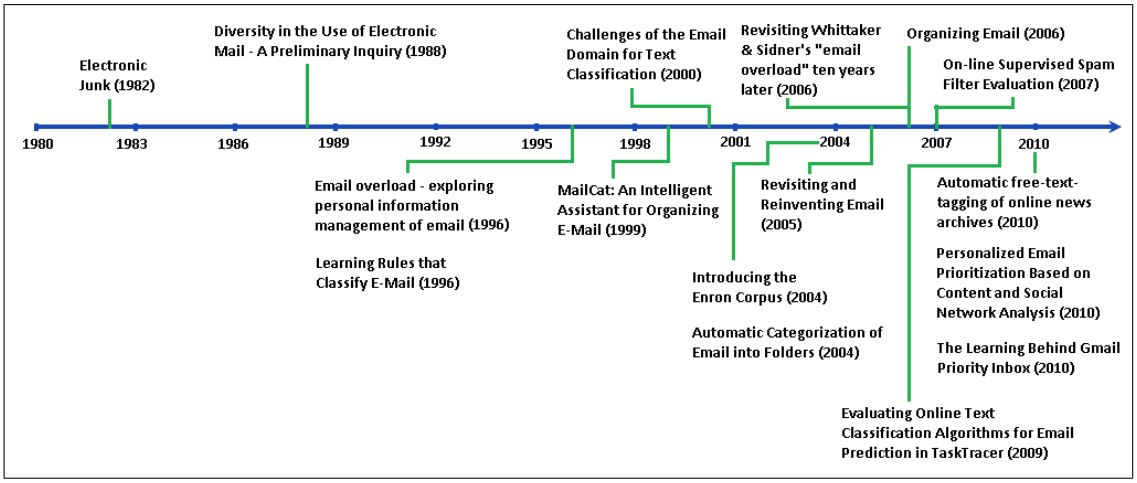

Fig. 1.1. Email research timeline, from 1980, to 2011 (the time of this writing).

1.2 Email Classification: Challenges and Related Work

In this section we present a summary of some of the most significant works in the field of email classification. The most common approach is to do automatic foldering (or some variant of it), using text classification techniques.

Cohen [20] detailed one of the first experiments with machine learning on email. His approach was rather simplistic, comparing a rule based classifier and a traditional information retrieval method for email filtering; both these methods were employed on text. While these results were encouraging, a private data set was used, making it difficult to do comparative studies and presenting a **problem that prevailed for some time** - the **lack of a public email data set**.

Segal & Kephart's Mailcat article [68] did an interesting description on folder suggestion. They developed a simple information retrieval classifier, similar to the one used by Cohen, that suggested up to three folders for filing a message. Their biggest concerns regarding classification were the **dynamic of the environment (folder creation/deletion/rearrangement)** and the **possible change in the meaning and content of folders over time**. Their solution was to apply an adaptive classifier that changed based on usage (feedback from user actions). They report an excellent performance on classification (80% to 90%) on databases that had archives of thousands of messages distributed to up to sixty folders. Since they did not actually file the messages, their system was non intrusive.

Brutlag & Meek [12] were among the first to study the problem of email classification. In their work, they analyze the challenges posed by email for text classification. They take an automatic foldering approach. The **main challenges** they have identified were: the **dynamics of the environment**, the **large number of sparse folders** and the **heterogeneity of the folders**. They have also noted that everything is **very dependent on the user's habits**.

Bekkerman & McCallum [4] present an extensive study on the problem of email foldering using a baseline data set. They have identified as problematic: the **dynamics of email usage (new folder creation and old folder deprecation over time)**, the fact that **folders do not necessarily correspond to simple semantic topics** and that the **topic can shift over time**, **content and foldering habits differ drastically from user to user** and **email arrives in a stream over time**. As seen, many of the problems that they have identified are similar to Brutlag & Meek's.

Cselle [22] revisited the problem of automatic foldering, using a subset of the Enron Corpus [47]. His **results were quite disappointing**, but in conformity with Bekkerman's work. He reported that these results are **strongly dependent on the corpus used for measurements**. He also states that this approach has several drawbacks, such as **ambiguities in folder filing**, the **dynamics of the environment regarding future needs (new folder creation) and**

folder naming.

The **problem of multi-class classification** is implicit in these descriptions [74]. Since users usually have several folders and the desired result is to file messages in one of them, a method of multi-classification must be devised (e.g. one classifier per folder or one global classifier for all folders).
It has also been observed that **folders usually suffer from large variations in the number of messages they contain**. It is frequent for some folders to have a large number of messages, while others have very few. This problem can be formalized as class imbalance [41] and occurs when the number of class examples is not large enough for the classifier to be able to generalize it. When such phenomenon takes place, it is common to have the smaller classes overshadowed by the larger classes.
Regarding the problem of email tagging, while there are some fields that are similar - e.g. topic detection and tracking, email folder classification ranking and folder suggestion - to our knowledge, very few work has been done with the specific goal of automatically tagging email messages.

1.3 State of the Art

To the best of our knowledge, very few work has been published in the exact field of our proposals in this thesis. As such, in this section we review the works of related fields, such as email classification and automatic tagging.

Yang et al [85] tackled the problem of personalized email prioritization. Their approach uses supervised classification, learning the kind of messages that the user considers important, based on the content and on the participants. They introduce the novelty of combining "unsupervised clustering, social network analyis, semisupervised feature induction, and supervised classification to model user priorities". A personal social network (graph) is built for each user of their data set. On the network, they apply the Newman clustering algorithm [18] in order to capture social groups of senders and recipients who might share similar priority judgments over messages. The results of the metrics they apply on the graphs are called the unsupervised social importance features.
Since their data set is not "complete" (in the sense that a large quantity of messages does not have an importance label associated to it), they use semisupervised learning of social importance features based on the importance labels that exist and the email interactions. They use the Level-Sensitive PageRank to propagate the label to non-labeled messages. These become the semisupervised features.
Messages are represented as synthetic vectors, where each one is the result of the concatenation of several groups of features: basic features (tokens in the from, to, cc, subject and body fields), semi-supervised social importance features (induced from both the assigned labels and the user

interactions) and Newman clustering features (features obtained from the clustering of the message's sender).
They apply Support Vector Machines (SVM) [43] (one per importance level) and evaluate using the mean absolute error metric on two types of tests: with and without social features. They claim that performance is significantly better when classification is done using social features.

Aberdeen et al [2] also tackled the problem of personal email prioritization through learning. Their method, Priority Inbox for Gmail, ranks messages by the probability that the user will perform an action on it. A per user statistical model is created in order to predict the importance. They identify features and calculate their values during ranking for later learning. The features are identified as social features (based on the degree of interaction between sender and recipient), content features (based on the content of the message), thread features (if the message belongs to some existing conversation) and label features (if the message was automatically filtered and labels were applied). Simple linear logistic regression models are used for learning and prediction. One global model, based on all existing features, is used in combination with the user model to rank the messages. Evaluation is done using an importance metric. Since messages are ranked, they must be above an importance threshold in order to be considered important. User feedback is used to update the models. Their results are very interesting and their studies point out that Priority Inbox is a valuable feature for gmail.

Farkas et al [32] present a work in the field of automatic tagging on news articles. They extract potential tags from the documents through linguistic analysis and from external sources. In order to limit the number of tags outputted from the system, they apply a filtering procedure to the candidate tags. The potential tags are extracted in a per-document way using named entity recognition and noun-phrases extraction and derivation. They use Wikipedia as an external source for tags. At first, they try to map a set of candidate tags to Wikipedia articles. Then, different methods of validation and assignment are applied on the mapped results, such as treating Wikipedia redirection pages or using the link structure to enrich the results. Potential tags are also extracted through the topic of the article, by analyzing the whole corpus. They call this approach "supervised learning of tags". Using logistic regression, they defined a training task for each of the 243 tags they had collected.
Hand-crafted heuristics based on a tag ranking method are used to select approximately five tags per document, since they had some special constraints to fulfill. The heuristics favored top-ranked named entities.
In order to evaluate the performance of their system, they have compared it to another system and manually graded the results from both systems in terms of F-Measure. Their results were satisfactory but they point out the difficulty on evaluating results from such a task.

Keiser and Dietterich [45] evaluate six different online multiclass text classification algorithms in the domain of email, within their TaskTracer [75] system. Their experiments analyze the performance of classifying messages to exactly one class, the impact of the number of classes and the

impact of class imbalance in the chosen classifiers.
They previously used a hybrid classifier based on SVM but found it unsuitable for online processing. They test Bernoulli Naive Bayes (BNB), Multinomial Naive Bayes (MNB), Transformed Weight-Normalized Complement Naive Bayes (TWCNB), Term Frequency-Inverse Document Frequency Classifier (TFIDF), Online Passive Aggressive Classifier (PA) and Confidence Weighted Linear Classification (CW) on a single-user email collection.
Messages are represented as boolean vectors. Features are defined for each unique sender/recipient address, for each unique set of recipients as a proxy for the "project team" and for each unique word in the subject and body. They divide the data into two sets: NoBody data set with 21827 features and Full data set with 84247 features. They conclude that CW and BNB show the most promising results for email classification, working well with their data set.

Bermejo et al [5] published a very recent study where the problem of class imbalance in email foldering is addressed. They describe how the large number of folders and the different number of messages per folder make email foldering a difficult problem. This problem is named class imbalance and has been overlooked until the recent appearence of large real life data sets.
It is referred how proposed solutions are mostly limited to binary classification problems, while in email foldering the problem is in the multi-class domain. They present a new method to tackle the imbalance of the data sets, based on sampling probability distributions. Their main contribution is a distribution-based balancing algorithm. Their focus is on the naive Bayes classifier and how it's performance can be improved in email foldering using the proposed balancing algorithm. The balancing approach that is proposed seems to improve classification accuracy results.

1.4 Objectives and Original Contributions

A previous work on this field was started in [72]. Even though the goal was the same, the approach was the opposite, using unsupervised learning, i.e. clustering, to try and automatically organize email messages. Tests were done where data was transformed through the usage of an external source of knowledge. The past work only made use of the textual fields of email messages.

With this work, we expect to address and to overcome some of the challenges of email classification and tagging. By taking a supervised learning approach, our plan is to review several techniques that are expected to be useful for our goal. To the best of our knowledge, this work has the following original contributions:

- Extensive analysis of two email collections and the implications for classification (section 4.1);
- Study of the effects of feature weighting (section 4.3.1) and feature selection (section 4.3.2) on the task of classification;
- Comparison of different algorithms in email classification (section 4.3.3);

- Usage of different fields (textual and social) of an email message for classification (section 4.3.3);
- Combination of classifiers for the different fields of email messages for automatic classification and tagging 4.3.5.

We also propose two new classifiers, one based on topic models (section 3.4.3) and another based on the social features of email messages (section 3.4.2).

A paper derived from this thesis has been submited and accepted for the Conference on Electronics, Telecommunications and Computers 2011 [1]. The paper, named Automatic Email Foldering With Supervised Learning - Addressing the Class Imbalance Problem, describes how class imbalance affects email classification and how simple balancing algorithms can help improve classification.

1.5 Document Organization

The remainder of this thesis is organized as follows:

- Chapter 2 describes the theoretical foundations of this work. We present an extensive description on the representation of email message fields, different vector space model variants, feature selection methods and classifiers;
- Chapter 3 shows the details of the proposed solution. The implemented system is presented, first through a general overview of the architecture and later by a description of the algorithms;
- Chapter 4 presents the data set, the experimental setup and the results of the tests. The Enron data set is analyzed in a complementary perspective to the existent descriptions. The setup of the experiments is described in full detail. The results are presented accompanied by a detailed examination;
- Chapter 5 finalizes the thesis with the main conclusions and hints for future work.

[1] `http://www.deetc.isel.pt/cetc11/`

2

Background Concepts and Theory

In this chapter, the theoretical foundations of the work are explored. We begin with a review on the background concepts (section 2.1). We then briefly describe classification and tagging systems and their main components (section 2.2). Then, we detail the different fields of an email message, describing possible representations for each field (section 2.3). Afterwards, we describe each component of a classification system and the main algorithms used at each step (sections 2.4, 2.5, 2.6 and 2.7). We visit topic models (section 2.8) and characterize the task of tagging (section 2.9). The chapter ends with the presentation of evaluation methodologies (section 2.10).

2.1 Background Concepts

The task of classification deals with data in the form of $X = x_1, \ldots, x_n$ where X is known as an **instance** and $x_1, \ldots, x_n$ are the **features** or attributes of the instance. The values of $x_1, \ldots, x_n$ are known as the **weights** and are commonly represented as $w_1, \ldots, w_n$. The goal of classification is to predict the class (or label) c of instance X, using a function f, also known as a classifier. f is inferred through a process of learning, from a set of training instances i.e. instances with a previously known label. This process is known as supervised learning [29].

In practical applications, the input data is standardized to a common representation for the use of a classification system. This is **preprocessing**'s main goal, to transform the input data from its raw format into the chosen representation of the system.

The data can suffer several kinds of transformation before classification, in order to improve the accuracy or performance. **Feature weighting** [66] deals with the transformation of the values of the features. The goal is to make the most relevant features stand out. The goal of **feature selection** [50] is to select a subset of features that best represent the data, removing redundant, irrelevant, and noisy features.

The goal of **classification** is to generalize a model for prediction from a sample of training

data. Given a set of instances $\mathcal{X}$, with previously known labels $C_{\mathcal{X}}$, the model will learn to predict the labels of instances with unknown labels. The training process is formalized as (2.1)

$$train(\{\mathcal{X}, C_{\mathcal{X}}\}) : test \tag{2.1}$$

where *train* is a function with input $\mathcal{X}$ (the set of instances) and $C_{\mathcal{X}}$ (the associated labels), and outputs *test* (the classification function). The function *test* (2.2) takes as input the instances $\mathcal{Y}$

$$C_{\mathcal{Y}} = test(\mathcal{Y}) \tag{2.2}$$

and outputs the predicted labels $C_{\mathcal{Y}}$.

Tagging is the task of assigning a set of labels to an input instance. The function *tagging* outputs a set of labels $\{c_1, c_2, \ldots, c_j\}$, where j is the number of tags returned by the specific tagging function.

2.2 (Email) Classification and Tagging Systems

A classification system takes as input a collection of instances and outputs the labels associated to each instance. This is done by the implementation of a framework that abides to the description of the previous section. In general terms, the architecture of such a system resembles figure 2.1.

Fig. 2.1. Generic framework of a classification system.

A tagging system also takes as input a collection of instances but outputs a set of labels to each input instance. The complete internals of such a system is usually dependent on the type of data it operates on, since tags can be extracted from more than just previously known labels. We present a general architecture of a tagging system in figure 2.2.

Fig. 2.2. Generic framework of a tagging system.

Farkkas et al [32] describe an automatic tagging system for news articles (figure 2.3). They

create a collection of candidate tags from each article through linguistic analysis and semantic knowledge. The system filters the candidate tags with a tag selection process, that ranks tags according to hand-crafted heuristics. The remaining tags are outputted by the system.

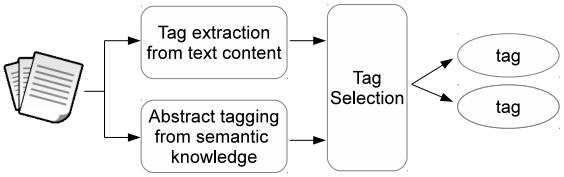

Fig. 2.3. Architecture of a news article tagging system, as described by [32].

Input data can take many forms, such as images, news articles or web pages. In our work, we focus our attention on email messages. Email messages are documents composed of different types of fields: the subject and body contain text whereas the date and the participants (from, to, cc, bcc) can be interpreted as structured fields since they follow a previously known structure.

A system which operates with email messages can take advantage of the additional information being offered, by using the different fields to improve classification. As an example, Gmail's priority inbox is an email classification system that outputs a priority level to an email message. In order to do so, it extracts different features of the messages and builds up a model based on the users of the system. When a new message is received by the system, its features are extracted and processed by the previously learned model, outputting a priority level. Figure 2.4 presents a rough approximation of the (inferred) architecture of the system.

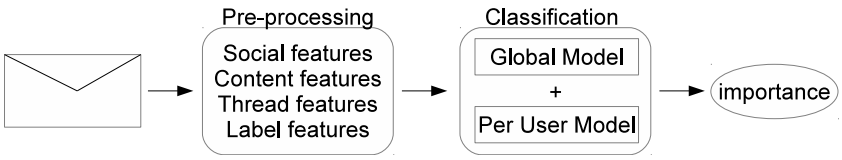

Fig. 2.4. Architecture of an email classification system, as described by [2].

2.3 Representing Email Message Fields

Even though an email message can be treated as a simple stream of text, such an approach totally disregards the structural information, since it is made up from different fields (figure 2.5).

The *textual fields* of messages (**subject and body**) are a natural fit for the vector space model.

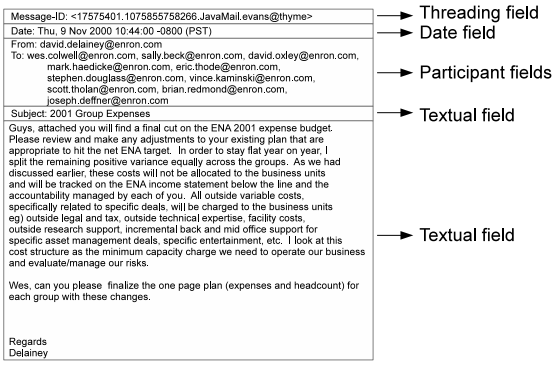

Fig. 2.5. Email message fields.

The vector space model [42] was developed by Salton et al [65] to enable the representation of textual data in a vector-like format. The instance of the data is a vector and its content (e.g. the text) is transformed into features (e.g. words) that represent the dimensions of the vector. When dealing with textual data, features are also known as terms. Formally, a document d is represented by a vector $d = \{w_1, w_2, \ldots, w_t\}$ where $w_1, w_2, \ldots, w_t$ are the weights of the features $T_1, \ldots, T_t$ (figure 2.6).

$$\begin{array}{c|cccc} & T_1 & T_2 & \ldots & T_t \\ D_1 & w_{11} & w_{21} & \ldots & w_{t1} \\ D_2 & w_{12} & w_{22} & \ldots & w_{t2} \\ \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & & \vdots \\ D_n & w_{1n} & w_{2n} & \ldots & w_{tn} \end{array}$$

Fig. 2.6. Term-document matrix from the vector space model.

Klimt and Yang [47] have also treated the *participant fields* (**from, to, cc, bcc**) as textual features while Roth et al [64] have taken inspiration from social network analysis and its graph-based representation.

In social network analysis [3], a graph G is constructed by representing the participants as vertices V and their interactions as edges E. Additional information from the email message can be used to enrich this representation, such as the directions of the edges.
In figure 2.7, a graph is shown. `david.delaney`, `wes.colwell`, `mark.haedicke`, `david.oxley` and `sally.beck` are the nodes while the interactions that are initiated from `david.delaney` are the directed edges.

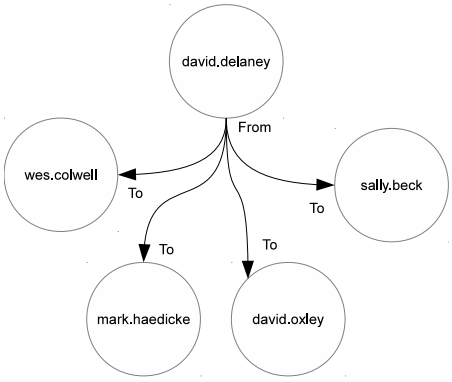

Fig. 2.7. Example graph built from the participants of an email message.

2.4 Feature Weighting

In this section, we present some of the most popular feature weighting schemes, based on the Vector Space Model.

The natural representation for text is the **term frequency**, where the weight w of a feature i is given by the number of occurrences of the feature in the document d_j

$$w(i,j) = tf_{i,j}. \tag{2.3}$$

The **boolean model** (2.4) is a variant where the features are represented with the values 0 or 1. This has the advantage of using less space for representing the weights allowing a more resource-efficient processing. On the other hand, features are treated as equals even if the number of occurrences is very different,

$$w(i,j) = \begin{cases} 1, \ tf_{i,j} > 0 \\ 0, \ \text{otherwise.} \end{cases} \tag{2.4}$$

The major drawback in using plain term occurrences or the boolean model is that every feature is given the same importance when assessing relevancy. That is, the importance of the feature is unlikely to be determined by the mere number of times it appears in a document.
Several approaches to tackle this problem have been proposed, such as **logarithmic term frequency (or sublinear tf scaling)** (2.5), which penalizes high frequencies, and **maximum term frequency normalization** (2.6), which tries to solve the anomaly of high frequencies in long documents.

$$w(i,j) = \begin{cases} 1 + \log(tf_{i,j}) \\ 0, \ \text{otherwise} \end{cases} \tag{2.5}$$

$$w(i,j) = \frac{tf_{i,j}}{max\ (tf_j)} \tag{2.6}$$

TF-IDF (**term frequency - inverse document frequency**) (2.8) is based on the notion that discriminative features in a collection of documents are the ones that can best distinguish a

document from the rest. This value is higher when the term frequency of a feature is high in a small number of documents (low document frequency). If a feature is too common or too rare, its discriminative power is very low.

$$idf(i) = \frac{1}{df(i)} \tag{2.7}$$

$$w(i,j) = tf_{i,j} \times idf(i) \tag{2.8}$$

Some variants of tf-idf have been proposed. The differences are in the combination of term frequency, document frequency and normalization, as shown in figure 2.8. The identification scheme follows a system of mnemonics known as the SMART notation.

Term frequency		Document frequency		Normalization	
n (natural)	$\mathrm{tf}_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1+\log(\mathrm{tf}_{t,d})$	t (idf)	$\log \frac{N}{\mathrm{df}_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2+w_2^2+...+w_M^2}}$
a (augmented)	$0.5+\frac{0.5\times \mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$	p (prob idf)	$\max\{0, \log \frac{N-\mathrm{df}_t}{\mathrm{df}_t}\}$	u (pivoted unique)	$1/u$ (Section 6.4.4)
b (boolean)	$\begin{cases}1 & \text{if } \mathrm{tf}_{t,d}>0\\ 0 & \text{otherwise}\end{cases}$			b (byte size)	$1/CharLength^{\alpha}, \alpha<1$
L (log ave)	$\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t\in d}(\mathrm{tf}_{t,d}))}$				

Fig. 2.8. Feature weighting schemes from the SMART System [52].

2.5 Preprocessing Techniques

Preprocessing has the main goal of transforming the input data from its raw format into a standardized format. For text, this usually implies extracting the data's textual string and transforming it into the vector space model.
Preprocessing can also apply different transformations that aim to remove redundancy and noise from the data or enrich the representation. The common tasks performed in preprocessing text [56] are:

- **tokenization**, which splits the original text string into tokens, such as words or phrases;
- **truecasing** in order to normalize the textual tokens;
- **stopword removal** which removes pre-defined tokens that are considered to have no relevance;
- **stemming** [61] might be applied to identify the common root of tokens, performing a syntax based merging and reducing the overall number;
- **part of speech tagging** is used to tag the tokens as nouns, adjectives, etc;
- **named entity recognition** is used to identify important tokens such as names or places.

2.6 Feature Selection

Feature selection [31] [39] serves several purposes, such as identifying the subset of the most relevant features for classification as well as aggressively reducing the number of features in order to speed up processing and to improve accuracy. Feature selection can be tackled through different approaches that can be combined: supervised vs. unsupervised [51] and filter vs. wrapper vs. hybrid [23]. Supervised feature selection methods make use of the instances' label in order to determine the subset of features, while unsupervised methods try to automatically generate these subsets using the available information, such as global corpus statistics. Filter approaches rank features using a score function and only the top ranked features are kept. Wrapper approaches use a learning algorithm to guide selection. Hybrid methods combine both approaches.

We describe some filter-based feature selection techniques, both supervised and unsupervised.

2.6.1 Unsupervised Feature Selection

Term Weight Filtering is a simple unsupervised filter feature selection method. Features are ranked according to their term weight, usually the term frequency or tf-idf. This is a very fast method but with varying results.

Document Frequency is an unsupervised filter feature selection method. Based on the Information Retrieval scoring scheme, it ranks features according to the document frequency, that is, the number of different documents where the term occurs. This method assumes that rare terms are useless or not influential for class prediction and even maybe prejudicial if terms are noisy. By removing them from the feature space, processing can be improved.

Variance is the second central moment and measures how the values of a given distribution are spread around the mean value of the distribution. Higher variance implies larger spread. In feature selection, when used in an unsupervised way, it determines the distance of a feature to the collection's mean (2.9, where $V(x)$ is the variance of the feature x, E is the expected value and μ_x is the mean of feature x). This represents the importance that the feature has for the whole collection.
Variance can also be used in a supervised fashion. When used in such a way, it is also known as Conditional Variance and determines the importance of the feature in regards to a class.

$$V(x) = E\{(x - \mu_x)^2\} \tag{2.9}$$

ℓ_0-norm methods are filter methods proposed in [34]. The ℓ_0-norm of a feature is the number of non-zero entries of the feature in the collection, also known as the document frequency. The first method is **Method 1**, unsupervised and identical to document frequency but with the added step of removing features that occur in all documents.

2.6.2 Supervised Feature Selection

Information Gain [21] [84] is a supervised filter feature selection method. It is used as a term goodness criterion, measuring the number of bits of information obtained for classification by knowing the presence or absence of a feature. The information gain of a feature x_k is given by (2.10), where $H(c)$ is the entropy of class c and $H(c|x_k)$ is the conditional entropy.

$$IG(x_k) = H(c) - H(c|x_k) \tag{2.10}$$

$$H(c) = -\sum_{i=1}^{m} P(c_i)\ log\ P(c_i) \tag{2.11}$$

$$H(c|x_k) = \sum_{j} P(x_k = v_j)H(c|x_k = v_j) \tag{2.12}$$

The **Fisher Criterion** is a method used in Fisher's linear discriminant [7]. It can be applied to feature selection as a supervised filter method that tries to evaluate how much a feature can discriminate between two classes (2.13, where $F(x_k)$ is the Fisher criterion value of feature x_k, $E(x_k|c)$ is the conditional mean of x_k with respect to class c and $D(x_k|c)$ is the conditional variance of x_k with respect to class c). Wang et al [80] describe an implementation of the Fisher criterion for feature selection.

$$F(x_k) = \frac{(E(x_k|P) - E(x_k|N))^2}{D(x_k|P) + D(x_k|N)} \tag{2.13}$$

$$E(x_k|c) = \frac{1}{n}\sum_{i=1}^{n}\frac{w_{c,i}(x_k)}{V_{c,i}(x_k)} \tag{2.14}$$

$$D(x_k|c) = \frac{1}{n}\sum_{i=1}^{n}(\frac{w_{c,i}(x_k)}{V_{c,i}} - E(x_k|c))^2 \tag{2.15}$$

The result of the computation is a matrix of pairs of class-feature. An analysis on this matrix must be done in order to determine the scalar value of the feature. We propose three scoring methods: the **Minimum Score** selects as value for the feature the minimum value, taking a pessimistic approach; the **Sum Score** is defined as the sum of all the pairs; the **Sum Squared Score** sums all the values of the pairs squared.

ℓ_0-norm **Method 2** is a supervised method, and works by applying Method 1 and afterwards, computing the rank of the remaining features. The rank, r_i of the ith feature, is defined in (2.16) as the sum of the absolute difference of the ℓ_0-norm of the feature among the K classes.

$$r_i = \sum_{l=1}^{K}\sum_{k=1}^{K} \mid l_0^{(i,l)} - l_0^{(i,k)} \mid \tag{2.16}$$

2.7 Classification

In this section, we focus our attention on classification algorithms and the problems that arise with classification. We begin by reviewing several classification algorithms. We then describe the problem of class imbalance, which affects classifiers when the training data is asymmetrically distributed among classes. We end this section with an overview of classifier combinations, describing methods to combine different classifiers to output a single decision.

2.7.1 Classification Algorithms

Decision Trees

Decision Trees [28] [57] refer to a family of algorithms that represent their prediction model as a decision tree. A decision tree (figure 2.9) is a tree structure with decision nodes, that apply tests on features, branches, which specify the possible outcomes of the test, and leaf nodes, which contain the labels.

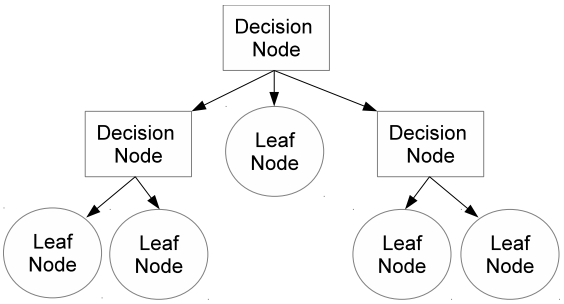

Fig. 2.9. Example of a decision tree.

Classification of an instance follows a recursive process, that starts by testing the feature at the root node of the tree and following the branch corresponding to the resulting output. The process is repeated on the subtree of the new node that lies at the end of the following branch. The process ends when a leaf node is obtained, assigning the correspondent label to the instance.

In the training phase, the decision tree is constructed. This is done by building a decision node, where a feature is selected from the training instances and the branches with the outcomes are inferred. Each leaf node contains the label of the instances associated with the correspondent outcome. The process is generalized by testing the accuracy of the resulting tree. Leaves that are not accurate enough are expanded into decision nodes, built using only the instances associated with this node.

Implementations of decision tree algorithms vary in the way the training is done. ID3 [62] is a popular algorithm for decision tree learning. The basic structure of the algorithm is iterative.

A subset of training instances, called the window, is randomly chosen to build a decision tree that correctly classifies all instances of the window. The tree is then used to classify the remaining instances of the training set. If all instances are not correctly classified, the incorrect instances are selected to be added to the window. The process stops when there are no incorrectly classified instances.

In order to build the tree, it is necessary to determine the features to be chosen as decision nodes. ID3 employs the information gain measure (see section 2.6.2), selecting the feature with the highest gain. Features cannot be repeatedly chosen as decision nodes.

Naive Bayes Classifier

Naive Bayes [28] [57] is a classifier that, given an input instance $d = x_1, x_2 \dots x_t$, assigns to each label $c_k \in \mathcal{C}$ a probability. The label with the highest probability is selected as the predicted label, $\hat{c}$, for the instance. Formally, the classification process is given by

$$\hat{c} = \underset{c_k \in \mathcal{C}}{\arg\max}\, P(c_k|d). \tag{2.17}$$

Using Bayes theorem, we can rewrite the conditional probability $P(c_k|d)$ as

$$P(c_k|d) = \frac{P(d|c_k)\ P(c_k)}{P(d)}, \tag{2.18}$$

where $P(d|c_k)$ is the class conditional and $P(c_k)$ is the class prior. The naive Bayes method assumes that $x_1, x_2 \dots x_t$ are conditionally independent given the class (a naive assumption), which translates into

$$P(d|c_k) = \prod_{i=1}^{n} P(x_i|c_k). \tag{2.19}$$

Since $P(d)$ is constant in respect to c_k, this yields

$$P(c_k|d) = P(c_k) \prod_{i=1}^{n} P(x_i|c_k). \tag{2.20}$$

The training phase of the naive Bayes classifier estimates the values of $P(c_k)$ and $P(d|c_k)$. $P(c_k)$ can be determined by counting the number of instances labeled with c_k in the training data. Computing $P(d|c_k)$ has become simpler with the independence assumption, as it is just the product of the probabilities for the individual features.

Maximum Entropy

In classification, maximum entropy is used as a probability distribution estimation technique, similar to the naive Bayes classifier. The principle of maximum entropy [28] [53] states that the choice of a distribution should fall to the one with the highest entropy, satistying the constraints that are known on the observed data. The rationale is to preserve the maximum uncertainty possible regarding the model. If the decision were to select the model with the least entropy, constraints would be added to the model that might not be correct since they wouldn't be supported by the available data.

Classification is done by estimating the conditional distribution of the class c given the instance d, i.e.

$$P(c|d) = \frac{1}{Z(d)} \exp(\sum_i \lambda_i f_i(d, c)) \tag{2.21}$$

where $f_i(d, c)$ is a maximum entropy feature, λ_i is the weight of the feature (a parameter to be estimated), and $Z(d)$ is the normalizing factor given by

$$Z(d) = \sum_c \exp(\sum_i \lambda_i f_i(d, c)). \tag{2.22}$$

The features in maximum entropy are constraints on the model, derived from the labeled training data. The training of the classifier consists in finding the weights λ_i of the features. The estimation of the parameters of the model is done using iterative optimization techniques.

For text classification [58], we represent a document by a set of word count features. We estimate the expected value of the word counts from the training data, on a class-by-class basis. For each word-class combination a feature is instantiated as:

$$f_{t,c'}(d, c) = \begin{cases} 0 & c \neq c' \\ \frac{N(d,t)}{N(d)} & \text{otherwise} \end{cases} \tag{2.23}$$

where $N(d, t)$ is the number of times word t occurs in document d, and $N(d)$ is the number of words in d. It is expected that features accounting for the number of times a word occurs shall improve classification.

Support Vector Machines

Support Vector Machines (SVMs) [10] [13] [52] [79] are discriminative, vector space based classifiers. With SVMs, we expect to find the maximum-margin hyperplane that separates the instances, in relation to their labels. This can be understood through the geometrical representation in figure 2.10.

Given instances d_i of labels $c_i \in \{-1, +1\}$, SVMs try to find a decision boundary between the

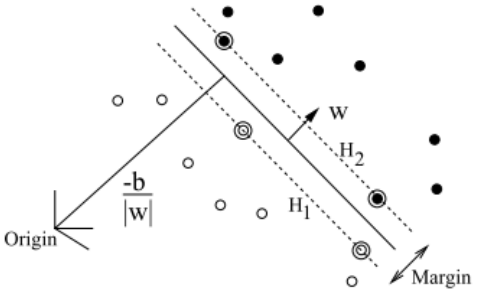

Fig. 2.10. Support Vector Machines [13].

features of d_i. This boundary should be as far from the features as possible (maximum margin).

The points d_i that lie on a hyperplane satisfy $w \cdot d_i + b = 0$ and are called the support vectors, where w is normal to the hyperplane, $\frac{|b|}{||w||}$ is the perpendicular distance from the hyperplane to the origin and $||w||$ is the euclidean norm of w.

Let the shortest distance from the separating hyperplane to the hyperplane of the positive instances $H1$ be given by dst_+ and the distance to the hyperplane of the negative instances $H2$ be dst_-. The **margin** of the separating hyperplane is $dst_+ + dst_-$.

The training of the support vector algorithm looks for the hyperplane with the largest margin. Classification is done by determining on which side of the decision boundary a given test instance d lies and assigning the corresponding class label, i.e. we take the class of d to be $sgn(w \cdot d + b)$. SVMs are inherently two-class classifiers. Different techniques can be used to generalize SVMs to the multiple class case, such as structural SVMs or building $|\mathcal{C}|$ one-versus-rest classifiers (where $|\mathcal{C}|$ is the number of different labels).

Winnow

Winnow [49] is a family of online classification algorithms, meaning that they do not make a clear distinction between the training and classification phase. When given an instance, the classifier makes a prediction. If there is a known label associated to it, i.e. the algorithm is being trained, the classifier uses the information to update its internal state related to its prediction and the true label.

Winnow mantains a set of k weight vectors $\{v_1, \ldots, v_k\}$, one for each label, that is a linear separator for the label. When training, these vectors are updated according to the specific algorithm, in response to the prediction made and the real label.

The Wide-Margin Winnow algorithm [4] is a multi-class implementation of Winnow. It keeps k weight vectors v of $(m+1)$ dimensions, one for every label, all initialized with the value one. m is the number of features. When given a new instance x, the classifier predicts the label to be (2.24):

$$j = \arg\max_{i=1,\ldots,k} \{v_i \cdot x\}. \tag{2.24}$$

In training, instances are presented t times to the classifier and the weight vectors are adjusted when the prediction is incorrect or prediction accuracy was insatisfatory (i.e. the ratio between the largest and second largest predictions is below a given threshold). The adjustment consists in increasing v_{real} and decreasing $v_{predicted}$.

2.7.2 Class Imbalance

A problem arising from the application of classification to practice is the phenomenon of class imbalance [16]. Some data sets are highly unequal in terms of instances per class - some classes can have thousands of documents while others might not reach a dozen. This is an issue to classifiers due to possible suboptimal classification performance - without sufficient data, classifiers cannot make accurate predicitions.

The existing solutions for this problem take two different approaches [41]:

- At data level, by trying to balance the instances through some transformation. Data approaches take the form of undersampling (selecting a subset of instances from a large class), oversampling (generating new instances in order to fill up a smaller class) and hybrid approaches.
- At algorithmic level, with algorithms that are sensitive to the size of the classes. Algorithmic approaches are usually in the form of weighted classifiers, that is, algorithms that give a different weight to each class according to its size. Ensemble methods [16] have also been proposed, such as bagging and boosting, which use a combination of several classifiers to predict the class.

Consider $\mathcal{C} = \{c_1, \ldots, c_k\}$ to be the training set organized by the different classes, and $c_i = \{d_1, \ldots, d_n\}$ one of these classes, where d_j denotes an instance that is contained in the class. In the literature, several approaches using data based transformations have been proposed [41], thus transforming $\mathcal{C}$ into a new balanced set denoted by $\tilde{\mathcal{C}}$. We describe two representative techniques. The problem with data-level algorithms is to select from a large class a subset of instances that characterizes it, or, from a small class obtain instances that can represent current and future status.

Random Sampling (Algorithm 1) is a simple method that randomly selects instances. There

are two different strategies depending if the number of instances of class c_i, denoted $\#c_i$, is greater or less than n_{th}, the target number of balancing instances. **Random Undersampling** is used when $\#c_i > n_{th}$. This consists of effectively removing an instance from the pool of instances so the next time sampling is done, this instance will not be available for selection. **Random Oversampling** is used when $\#c_i < n_{th}$, where instances are randomly selected but not removed from the pool of instances.
Some problems may arise in both approaches: in the first, the larger class might not become completely represented if the undersampling is very severe; in the later, duplication of instances causes classifiers to focus on the specific regions of the minority class which can lead to overfitting.

Algorithm 1 Random Sampling

Input: $\mathcal{C}$ - set of messages organized by folder.
n_{th} - target number of balancing messages.

Output: $\tilde{\mathcal{C}}$ - balanced set of training messages.

```
for c_i ∈ C do
    if (#c_i < n_th) then
        Perform random sampling with reposition (oversampling).
    else
        Perform random sampling without reposition (undersampling).
    end if
end for
```

Synthetic Minority Over Sampling Technique (SMOTE) is an oversampling method developed by Chawla et. al. [15] that works by generating artificial instances through interpolation of existing instances on classes with number of instances lower than the predefined number, n_{th}. This approach is designed to avoid overfitting.

The basis of this technique is that, when oversampling, the decision region of classification for minority classes becomes very specific. The duplication of instances causes classifiers to focus on the specific regions of the minority class. This leads to overfitting.

The key idea of SMOTE is to expand the decision region of classification for the minority classes by creating synthetic instances in a way such that the decision region expands into the area of majority classes. Instead of duplicating instances, the new instances are generated through a combination of features. Samples are generated by first computing the difference between an instance of the minority class and one of its nearest neighbors. Then, the difference is multiplied by a random number between 0 and 1 and added into the minority class' instance. SMOTE effectively causes the decision region to expand since the synthetic samples are generated using instances of the majority class, generalizing the classifier.

Samples are generated by first computing the difference between an instance of the minority class and one of its nearest neighbours. Then, the difference is multipled by a random number between 0 and 1 and added into the minority class's instance.

SMOTE effectively causes the decision region to expand since the synthetic samples are generated using instances of the majority class, generalizing the classifier. In order to use a full sampling approach, SMOTE can be combined with undersampling (Algorithm 2).

Algorithm 2 SMOTE with Undersampling

Input: $\mathcal{C}$ - set of messages organized by folder.
n_{th} - target number of balancing messages.

Output: $\tilde{\mathcal{C}}$ - balanced set of training messages.

for $c_i \in \mathcal{C}$ **do**
  if $(\#c_i < n_{th})$ **then**
    for $d_j \in c_i$ **do**
      Find its $k-$nearest minority neighbors.
      Randomly select m of these neighbors.
      Randomly generate synthetic samples along the lines joining the minority sample and its m selected neighbors (the value of m depends on the amount of oversampling desired)
    end for
  else
    Perform random sampling *without* reposition (undersampling).
  end if
end for

2.7.3 Classifier Combination Methods

The performance of a classification system can often be improved by combining multiple classifiers [7] [76]. Several ways to combine classifiers have been devised and can be combined themselves. We review a few of the most popular methods and their approaches.

Ensemble or committee of classifiers, use a predefined function to combine the outputs of the individual classifiers. Commonly used functions are sum, weighted sum, max, among others. Non-Ensemble combinations try to combine heterogeneous classifiers, where each tackles a different aspect of the problem. Another approach takes the output of the classifiers to be the input of a "meta" classifier. The combination algorithm is learned from the different input classifiers.

Ensemble Classifiers

Bagging [11] or bootstrap aggregating [30] is a method that generates M bootstrap data sets and then uses each to train a separate copy $y_m(d)$ of a predictive model where $m = 1, \ldots, M$. The committee prediction is given by

$$y_{com} = \frac{1}{M} \sum_{m=1}^{M} y_m(d). \tag{2.25}$$

Boosting [37] [40] [67] is a method for combining multiple 'base' classifiers to produce a form of committee whose performance can be significantly better than that of any of the base classifiers. Boosting commonly makes use of classifiers that have weak accuracy performance (known as weak learners).
The classifiers are trained in sequence, each taking as input a weighted form of the data set, where the weights are associated to the performance of the previous classifier. Instances that were incorrectly classified are given a larger weight. Final prediction results from weighted majority voting.

Non-Ensemble Combinations

Majority Voting is a simple voting technique that makes its prediction to be the most frequent class given by the classifiers. Variations of this method have been described, such as weighted majority voting, where each classifier is given a weight.

Borda Count is a rank-based voting technique. The method computes the rank r_i of each class for every classifier. Overall rank r_i of class i is given by $r_i = \sum_{j=1}^{N} r_i^j$, where N is the number of classifiers, and r_i^j the rank of class i in the result of the j-th classifier. The final prediction is given by the best overall rank sum.

2.8 Topic Models

Probabilistic Topic Modeling [8] is a suite of algorithms for discovery and annotation of themes in large data sets of text documents. The algorithms inspect the texts and the words of the documents and make use of statistical methods to discover the themes and interconnections that are present in the data. Topic models are unsupervised methods, and thus do not require any information regarding document labels.

Latent Dirichlet Allocation (LDA) is a statistical model with the intution that documents can contain multiple topics. It is a generative process, that assumes that a topic is a probability distribution over a fixed vocabulary, and that it is specified before the data is generated. The **generative document process** starts by selecting a distribution over the topics, i.e. probabilities of topics. For every word in the document, a topic from the distribution over topics is randomly selected. Then, a word is randomly selected from the corresponding topic.

All the documents in the data set share the same set of topics, but documents exhibit the topics with different proportions.

The topics, the topic distributions for the documents and the document and word assignments are called the hidden structure. The only observable information are the documents.
In order to discover the topical information, the documents are used to infer the hidden topic structure.

The discovery of the topical information is done using probabilistic models, where the observed variables are the words of the documents, the hidden variables are the topic structure and the computational problem of inferring the hidden topic structure from the documents is the problem of computing the posterior distribution.

Topics are $\beta_{1:K}$, where each β_k is a distribution over the vocabulary (the distributions over words), topic proportions for the dth document are θ_d, where $\theta_{d,k}$ is the topic proportion for topic k in document d, topic assignments for the dth document are z_d, where $z_{d,n}$ is the topic assignment for the nth word in document d, observed words for document d are w_d, where $w_{d,n}$ is the nth word in document d, which is an element from the fixed vocabulary.

The generative process for LDA corresponds to the following joint distribution of the hidden and observed variables

$$p(\beta_{1:K}, \theta_{1:D}, z_{1:D}, w_{1:D}) = \prod_{i=1}^{K} p(\beta_i) \prod_{d=1}^{D} p(\theta_d)(\prod_{n=1}^{N} p(z_{d,n}|\theta_d)p(w_{d,n}|\beta_{1:K}, z_{d,n})) \tag{2.26}$$

The distribution specifies a number of dependencies that define LDA. They are encoded in the statistical assumptions behind the generative process.

The computational problem is to calculate the conditional distribution of the topic structure from the observed documents, the posterior, given by

$$p(\beta_{1:K}, \theta_{1:D}, z_{1:D}|w_{1:D}) = \frac{p(\beta_{1:K}, \theta_{1:D}, z_{1:D}, w_{1:D})}{p(w_{1:D})} \tag{2.27}$$

The numerator is the joint distribution of all the random variables. The denominator is the marginal probability of the observations, which is the probability of seeing the observed corpus under any topic model.

Topic modeling algorithms form an approximation to the posterior by forming an alternative distribution over the latent topic structure that is adapted to be close to the true posterior. These algorithms generally fall into one of two categories—sampling-based algorithms and variational algorithms.
Sampling based algorithms attempt to collect samples from the posterior to approximate it with

an empirical distribution. Variational methods are a deterministic alternative. Variational methods assume the existence of a parameterized family of distributions over the hidden structure and then find the member of that family that is closest to the posterior, transforming the inference problem into an optimization problem.

2.9 Tagging

The assignment of tags in an automated way has been a popular research problem in the field of recommendation systems for web 2.0 applications. The goal has been to suggest tags to a user, when the system receives an entity to tag.

Tagging should not be seen as something opposed to classification, but rather an extension. Classification usually deals with the single-labeling of instances (this is specifically true in automatic foldering), but multi-labeling is also possible. As such, classification can be used for tagging. While tags are usually user-generated, it is also common to do tag suggestion through the extraction of features from the instance's content.

Si et al [69] proposed a content-based tag suggestion method called **Feature-Driven Tagging** (FDT). This method presumes the existence of a feature-tag matrix, built during a training phase. The suggestion of tags is done by a scoring model, based on the features.
FDT starts by extracting the features from the instances (e.g. words). Next, feature weighting is applied (e.g. TF-IDF). A matrix Θ stores the association between a feature and tags, where $\theta_{i,j}$ represents the weight of tag t_j to feature f_i. The matrix Θ is of size $|F| \times |T|$, where F is the set of features and T is the set of tags. The matrix is computed using a method like Mutual Information.
When suggesting tags for a new instance, the feature extraction and weighting process is applied. Scores are computed to the tags that are associated to the extracted features and form a weighted list of tags, which is outputted.

Rose et al [6] proposed a keyword extracting method called **Rapid Automatic Keyword Extraction** (RAKE). This is an unsupervised method that operates on individual documents.
RAKE tries to identify relevant keywords through the usage of predefined delimiters. It then assesses the importance of the keywords using frequency statistics, returning the top T number of keywords as the result.
RAKE takes as input the document text, a set of stop words, a set of phrase delimiters and a set of word delimiters. The stop words and the phrase delimiters are used to partition the text into a list of candidate keywords whereas word delimiters are used to split the text into an array of words.
After every candidate keyword is identified, a graph of word co-occurrences is built (words that belong to the same candidate keyword). Then, a score is computed for every word and candidate

keyword. The score of a word can be obtained by using the degree or the frequency of words in the graph.The score of a candidate keyword is defined as the sum of the score of the individual words that belong to the keyword. The top T scoring candidate keywords are selected as keywords for the document.

Dredze et al [27] detailed a set of **methods to generate summary keywords for emails using topic models**. They use the latent representations of the underlying topics in order to find words that best describe the email messages. First, messages are represented as latent concept models. Then, two keyword summary generation methods are presented, one based on query-document similarity and ther other based on word association. On the query-document similarity method, a candidate keyword is represented as an one word query and its similarity is computed with the email message. The most similar candidate keywords are selected. On the word association method, pairs of words are scored in terms of their co-association. The most closely associated words of a message are selected as keywords. The number of keywords to be returned is based on empirical observations.

2.10 Evaluating Results

The evaluation of a classifier can be made in several ways. When assessing simple classification results, four metrics are generally used:

- **true positive (tp)** are instances belonging to a class C and classified as such;
- **true negative (tn)** are instances not belonging to a class C and not classified as such;
- **false positive (fp)** are instances not belonging to a class C but classified as such;
- **false negative (fn)** are instances belonging to class C but not classified as such.

With these metrics, it is possible to compute the **accuracy** of the classifier, defined as

$$Acc = \frac{tp + tn}{tp + tn + fp + fn}, \tag{2.28}$$

which translates as the ratio of correctly classified instances. Sometimes the **error rate**, defined as

$$Err = 1 - accuracy, \tag{2.29}$$

is preferred.

Other metrics, popular in information retrieval, are the **precision**, defined as

$$P = \frac{tp}{tp + fp}, \tag{2.30}$$

and **recall**, defined as

$$R = \frac{tp}{tp + fn}, \tag{2.31}$$

The **F-Measure** [78] combines both precision and recall to measure the test's accuracy and is defined as

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall}. \tag{2.32}$$

Weighted Accuracy has been proposed as a measure to evaluate imbalanced data sets

$$\text{weighted accuracy}(\lambda) = \frac{\lambda tn + tp}{\lambda(tn + fp) + (tp + fn)}, \ \lambda > 0, \tag{2.33}$$

where λ is the specified weight given to the correctly classified instances of the smaller class.

The previous metrics evaluate the best decision of the classifier. A different type of evaluation metric has been used where the evaluation does not fall on the best decision, but rather on the rank of the real label in a ordered list of labels outputted by the classifier. This has been called **top-N accuracy**, where N is the size of the list.

Using the previous metrics to analyze classifier performance can be misleading since the impact of class size is not shown. One tool, commonly used to analyze class imbalance, is the **receiver operating characteristic (ROC) curve** [33] (figure 2.11), which measures the true positive rate versus the false positive rate when varying a discrimination threshold for a binary classification. The threshold can be a value such as the classification probability that is used to determine classification.

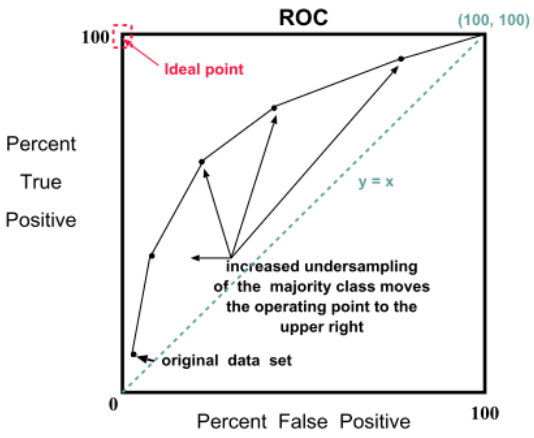

Fig. 2.11. Illustration of sweeping out a ROC curve [16].

The true positive rate (TPR) is given by 2.34

$$TPR = \frac{tp}{tp + fn}, \tag{2.34}$$

while the false positive rate (FPR) is computed as in 2.35

$$FPR = \frac{fp}{fp + tn}. \tag{2.35}$$

The ideal point is located at $(0, 1) \times 100\%$ when $tp = 100\%$ and $fn = 0\%$ for the TPR, and $fp = 0\%$ and $tn = 100\%$ for the FPR, which yields perfect accuracy ($Acc = 1$).
The ROC curve only presents the results of the classification of one class. By using the information regarding the percentage of positive and negative instances correctly classified, the ROC can present the impact classification does to smaller classes.

A **confusion matrix** is a visual tool that displays the relationships between classes and the classification results (tp, tn, fp and fn). It enables the analysis of how the classified instances are distributed (number of correctly classified instances per class and where the incorrect instances are classified in).

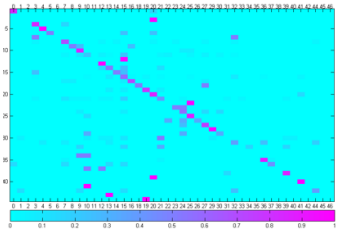

Fig. 2.12. Example of a confusion matrix.

Figure 2.12 shows an example of a confusion matrix. Both columns and rows contain the labels. Rows indicate where the instances of the label have been classified. For a correct classification, only the diagonal of the matrix should be colored.

3

Implementation Details

In this chapter we describe the details of the implementation. A general overview of the implemented system's architecture is given in section 3.1. We dwelve into the implementation details of the subsystems and the algorithms used in the subsequent sections. Section 3.2 deals with the preprocessing, section 3.3 describes the feature weighting and feature selection methods, classification is presented in section 3.4 and tagging is discussed in section 3.5. In section 3.6 we describe how we evaluate the results of the system.

3.1 System Architecture

Our system (figure 3.1) takes the classification approach into tagging. Our rationale is to expand classification, using it to provide predefined tags (in the form of folders). We use the textual and participant fields of email messages to try and improve our solution.
The implemented system follows the generic approach for classification. The input is in the form of email messages in plain text format, from which we extract the target fields (subject, body, from, to, cc, bcc). A preprocessing module for the textual fields formats the input data while feature weighting and feature selection are applied for the classification module. In addition to the textual representation of the subject and body, we also use topical representations on them and classify with the topic model classifier. Participants can be classified using either the peoplefier classifier or standard vector-model classifiers.
The result of each classifier is given to a meta-classifier that, using classifier combination techniques, outputs the results.
In order to accelerate and standardize development, we use an existing open source machine learning framework, named Mallet [55]. This framework is written in Java [60] and specializes in topic models and text classification algorithms.
The analysis of results is done using MATLAB [54]. Scripts are used to parse the output and analyze the results.

Mallet's two main classes are `Instance` and `InstanceList`. The `Instance` class represents a

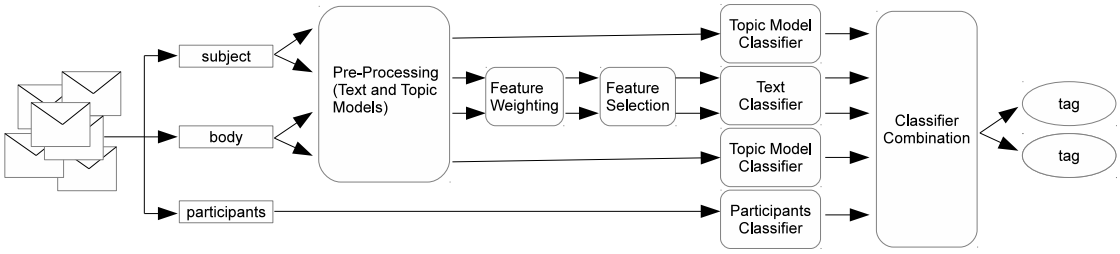

Fig. 3.1. General architecture of the system.

single data source, e.g. an email message or a news article. Its content can vary, depending on its usage. When used to load the original data, the content can be a string while after preprocessing, the content should be a feature vector. The `InstanceList` class represents a collection of `Instance`s. It is generally used to process data in batch mode.

Execution of the system is done by calling the desired classes for processing. Preprocessing is done by classes that extend from `PreProcessor`. Class imbalance is treated by subclasses of `Balancer`. Classes that implement `IWeight` and `IFilter` are responsible for feature weighting and feature selection, respectively. Classification training is done using Mallet's `ClassifierTrainer` subclasses, while prediction uses the equivalent `Classifier` class.

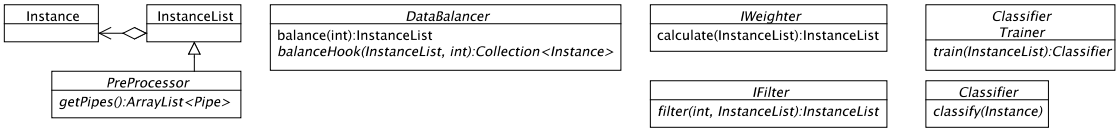

Fig. 3.2. Overview of the implemented architecture's main components.

3.2 Preprocessing

Preprocessing is done using Mallet's preprocessing classes: `Pipe`s. A pipe is a single preprocessing operation (e.g. tokenization). Mallet has several pipes available for preprocessing. Figure 3.3 presents the classes envolved in this step.
We use tokenization (`CharSequence2TokenSequence`), truecasing (`TokenSequenceLowercase`) and stopword removal (`TokenSequenceRemoveStopwords`).
A custom `Pipe` was created in order to use the Porter stemming algorithm. We used the stemming code available in [61] and wrapped it into a `Pipe`.
The result of preprocessing is a feature vector.

The `PreProcessor` abstract class implements a preprocessing pipeline, a sequence of prepro-

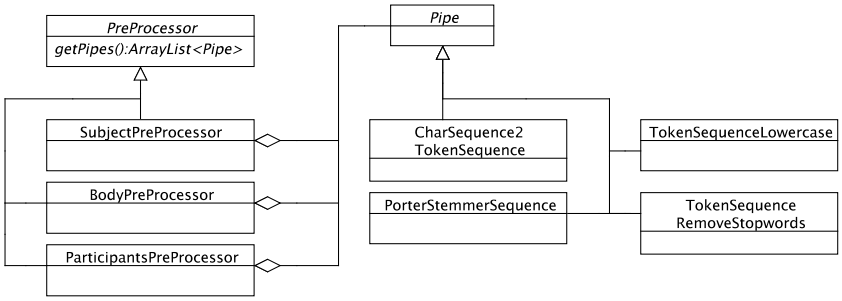

Fig. 3.3. Unified Modeling Language (UML) class diagram of the preprocessing classes.

cessing operations. Preprocessing is done to various fields of an email message by the respective classes that derive from `PreProcessor`: `BodyPreProcessor`, `SubjectPreProcessor`, `ParticipantsPreProcessor`.
After the preprocessor classes are executed, the respective `InstanceList` for each field contains the vectorized instances.

3.3 Feature Transformation

Figure 3.4 presents the two types of feature transformation that are implemented. Feature weighting can be applied by using `FeatureWeighting`, which implements the interface `IWeighter` while feature selection can be done by one of the implementations of `IFilter`.

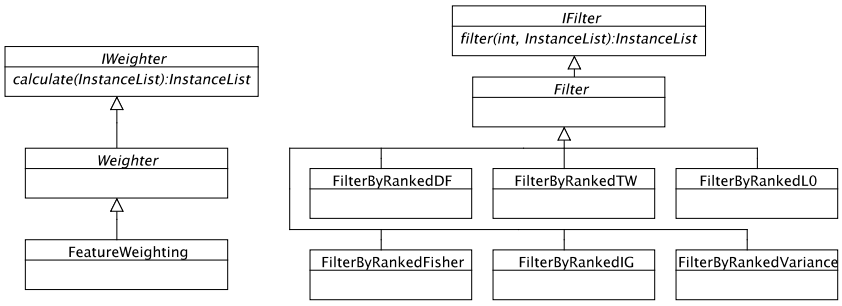

Fig. 3.4. UML diagram of the feature weighting and feature selection classes.

The `FeatureWeighting` class implements three types of feature weighting, allowing any combination between them:

- term weighting;
- document weighting;
- normalization.

The following term weights can be used:

- term frequency;
- maximum normalization term frequency;
- logarithmic term frequency;
- boolean weighting.

In document weighting, inverse document frequency can be applied. Normalization can be done using the cosine normalization.

Eight techniques for feature selection were implemented (for complexity reasons all of them are of the filter type) and previously described in section 2.6: term weight, document frequency, variance, ℓ_0 norm method 1 and method 2, Fisher criterion (with minimum score, score sum and squared score sum). Mallet's implementation of information gain can also be used.

Both these transformations have an `InstanceList` as input and output, while the `Instance`s' data is in the `FeatureVector` format.

3.4 Classification

Due to the large dimension of the data set, it's imbalance, and in order to do extensive testing of the several algorithms, we use and extend Mallet's cross validation classes. The `ExecutionRun` class applies cross validation using the specified classifier and number of folds. The results are in the form of objects of `ExtendedTrial`, which is an extension to Mallet's `Trial` class with the added information of the training instances and test instances that were used in the specified fold. `CombinationUtils` is used to perform the combination of multiple classifiers.

To allow testing of several levels of variation of a method, e.g. for feature selection, the `IteratedExecution` class was created. It takes as arguments a maximum and a variation value, outputting an array of step values. When applied to feature selection, the step values are the number of features that are used in the filter methods.

The `ExecutionResult` class holds all of the information regarding an execution, i.e. one `InstanceList`'s full processing (feature transformation and selection variations plus classification with cross validation). This class supports serialization of the trials and the classifier accuracies. `Trial`s are written into one file per trial, while accuracies are written into one common file.

3.4.1 Classification Algorithms

Classification is done using Mallet's `ClassifierTrainer` class. This is the base class for all classification algorithms. Mallet contains several algorithms: `NaiveBayesTrainer`, `MaxEntTrainer`,

`DecisionTreeTrainer`, `WinnowTrainer` and `BalancedWinnowTrainer`. We have extended the list of classifiers by creating a wrapper for LibLinear [14], an implementation of SVMs, which we named `LibLinearTrainer`. We have also implemented two new classifiers: `PeoplefierTrainer` and `TopicModelTrainer`, described in sections 3.4.2 and 3.4.3, respectively. All classes are shown in figure 3.5.

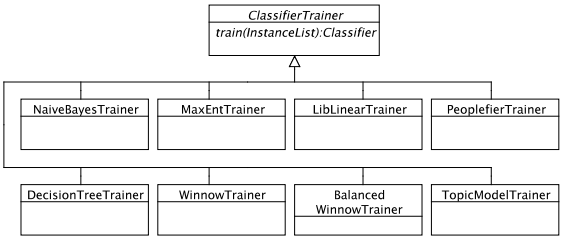

Fig. 3.5. UML diagram of the classification classes.

The Majority Voting algorithm is implemented in `MajorityVotingClassifier`, which performs the prediction using the outputs of several classifiers.

3.4.2 Predicting Classes Based on Email Message Participants

Email participants are an interesting source of information. Although most work has been done in the field of social network analysis, only a few works have used this information for classification.

The most common approach when dealing with the participants for classification has been to **vectorize the participants**, i.e. to treat each participant as a feature and extend the feature vector with this information. This approach is rather simplistic, as it does not take into account the structural information present in the messages, such as the direction of interaction.
This approach is implemented in our work, in the form of the `ParticipantsPreProcessor`, which vectorizes the participants of the messages. Classification is then applied using one of the previously described classifiers.

Behind our approaches is the rationale of trying to capture the co-occurrences of participants. Instead of treating features as single participants, we had in mind enriching the features with something more, since interactions always occur between 2 participants at least. At first, we tried computing the **powerset** of the participants and using each element as a feature. Unfortunately this proved to be too expensive to compute, due to the fact that several messages have a large number of participants. Taking inspiration from the work of Roth et al [64], we developed a graph based classifier for the participants. Using the common social network representation,

each participant is represented by a node and the interactions between them as edges.

This approach, named **Peoplefier**, constructs a graph during the training phase. This graph is directed, taking into account the relationship of the `from`, `to`, `cc`, and `bcc` fields; weighted, using the number of interactions between two participants as the weight of the edge that connects its respective nodes. The graph is also typified, since that the class information differentiates the edges.

Figure 3.6 presents an example of a graph built using the Peoplefier algorithm. User `david.delaney` has several outgoing edges and two incoming edges from `sally.beck`. The incoming edges belong to different classes (`C1` and `C2`), are of the same relationship (`To`) and have different weights. This example represents the case where `david.delaney` has two folders named `C1` and `C2` that contain messages exchanged with `sally.beck`. In folder `C1`, he has two messages, while in folder `C2`, he has kept twenty-one messages.

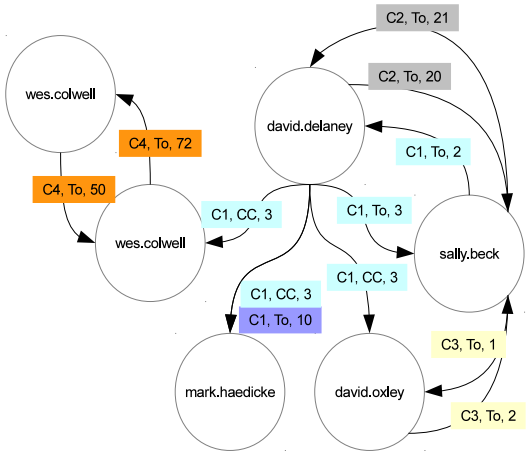

Fig. 3.6. Example graph built by Peoplefier during training.

Prediction occurs using a scoring system. Given a set of participants, $\{p_1, \ldots, p_N\}$, from an email message, we calculate the score of each class, S_c, given by

$$S_c = \sum_{p}^{N} S_p(c) \tag{3.1}$$

where S_p is the participants' score, computed as

$$S_p(c) = \sum_{e}^{edges_c} type_e \cdot weight_e \tag{3.2}$$

For all the edges of the specified class, $edges_c$, we multiply its weight $weight_e$ with a predefined value of the type of the edge, $type_c$. We set *type* to be 1.00 for *from* and *to*, 0.50 for *cc* and 0.25

for *bcc*. After computing the scores, we pick the label with the largest score as the prediction of the classifier. Our goal is to use all of the participants' information for classification. Our belief is that this information can help classification.

A variation of this graph-based approach, which we imaginatively named as `PeoplefierV2`, uses the training graph to construct a feature vector. Here, the rationale is to use common classifiers instead of the scoring system, while retaining all of the graph's information regarding the participants.

In the training phase, the previously described graph is built. Afterwards, the graph is transformed into an `InstanceList`, where each feature, `PeoplefierFeature`, is a directed participant, i.e. a participant derived from a directed edge, and each instance is a labeled participant, i.e. a participant derived from the typified edge. The weights of the features are given by

$$\sum_{e}^{edges(direction,class)} weight_e \tag{3.3}$$

Classification is done by transforming an instance into the previously described representation and then applying a common feature-vector classifier, such as naive Bayes, for instance.

3.4.3 Classifying with Topic Models

Topic modeling tries to discover the hidden structure of text data sets. It would be of great value if this hidden structure were to be similar to classification labels. Unfortunately, this is not guaranteed to happen, and in practice it is very improbable to happen. As such, using topic models for classification should require some kind of transformation on the topics in order to be of practical use.

Our approach is very simplistic and essentially, maps topics to classes. The training phase of our `TopicModelClassifier` starts by estimating the model of the topic distribution for the document at hand. Then, an association between topics and classes is built. This is done by determining the most frequent topic of the document, which becomes associated with the true class of the document. At the end of the training phase, a mapping exists between topics and classes.
Prediction happens by determining a score for each class. This is done by first inferring the model of the document. Afterwards, for every word in the document, its topic is used to retrieve the classes associated to it and the probability of the topic is added to the score of each of the retrieved classes. In the end, the class with the highest score is the predicted class.

3.4.4 Class Imbalance

In order to tackle the class imbalance problem (mentioned in 2.7.2), abstract class `Balancer` was created. `RandomSampler` and `SMOTE` derive `Balancer` and implement the random sampling (with undersampling and oversampling) scheme and SMOTE algorithm, respectively. The balancing classes are presented in figure 3.7.

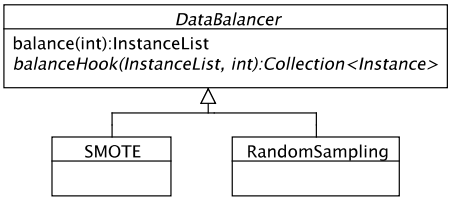

Fig. 3.7. UML diagram of the data balancing classes.

Balancing is applied before classification, preferentially right after preprocessing. The balancing algorithm implementations take as input the class instances and the number of documents that should be obtained. How this is done is dependent on the algorithm.

3.5 Tagging

As presented in our system's architecture, we use different classifiers for the different fields of an email message. Tagging is achieved by combining these different classifiers using a classifier combination method, e.g. `MajorityVotingClassifier`.

The output of this approach are the predefined classes from the folder structure created by the user. We can extend the single-labeling of the meta-classifier by using the top-N approach, where we take as tags the N best classified labels.

In order to add some diversity to the resulting tags, we also implement the RAKE algorithm, defined in section 2.9, which is used to extract relevant keywords, from the documents, that are to be used as tags. Here too, a top-N approach is available, returning the N most relevant keywords.

An alternative to RAKE, but also from content-based tagging, is to use the results from topic models. The words with the highest probability can be chosen as representative for a class.

3.6 Result Evaluation

Results are analyzed using MATLAB scripts that compute the accuracy (`accuracy_viewer`), the confusion matrix (`confusion_matrix`) and the ROC curve (`roc_viewer`). These results are presented as graphics, for a visual analysis. Rank analysis for top-N classification is also available (`rank_evaluator`). All of these measures are defined in section 2.10

In addition, we also have several data set analysis scripts, which provide statistics regarding the data set (`stats_collection`) and its content (`stats_content`).

4

Experimental Evaluation

In this chapter, we describe how experiments were conducted. We begin by characterizing the data sets that were used (section 4.1). Then, the testing methodology is presented (section 4.2). We end this chapter with the results of classification and an analysis of the executions (section 4.3).

4.1 Data Set Analysis

The lack of a common email data set has been considered as one of the biggest obstacles to the development of the field of email classification. Without such a common resource, researchers have had to gather their own private data for tests, making it difficult to compare and share tools and results. In recent times, the Enron email data set has been made available, allowing comparisons between different approaches.

4.1.1 Enron Corpus

The **original Enron corpus** [19] is a large set of email messages gathered during the legal investigation of the Enron corporation. It contains roughly 500 000 email messages from over 150 users.

This is the largest known publicly available data set of email messages. Since it's a "real world" data set, collected from several users, it is very heterogenous. The number of messages and folders varies considerably between the users.

Klimt and Yang [46] [47] were among the first to analyze the Enron corpus. Their goal was to understand if this corpus is suitable for email classification, namely automatic foldering. They made their own corrections to the data set, removing repeated messages and redundant folders (e.g. folders that were automatically created by the email client software). Their final collection contained 200 399 email messages, distributed by 158 users.

Their work concludes that the Enron corpus is indeed suitable for email classification, being very diverse in terms of users and number of messages. One important note was that this data set proves that users really use folders for organizing their email.

Bekkerman, McCallum and Huang [4] made the reference study on automatic foldering using the Enron corpus. One of the biggest legacies of their work was the subset of data they used - tests were done using the data from the seven users with the most messages. They also applied additional steps to clean the data, removing redundant folders, flattening the folder hierarchy and taking out all the folders that contained less than three messages.

Data Set Statistics

Our work is based on the data set introduced by Bekkerman et al. We present some statistics regarding the data of the seven users, in terms of folder organization (e.g. number of folders) and message content (e.g. time distribution of the messages).

Collection Statistics

In order to understand the data, we first analyze the organization of each users' mailbox (table 4.1). The seven Enron users have fairly large mailboxes (all above 1000 messages), with some variation in terms of organization (`beck-s` has 101 folders while `lokay-m` only has 11).

User	Num. folders	Num. messages	Smallest folder	Largest folder	Avg. num. messages/folder
beck-s	101	1971	3	166	19.51
farmer-d	25	3672	5	1192	146.88
kaminski-v	41	4477	3	547	109.2
kitchen-l	47	4015	5	715	85.43
lokay-m	11	2489	6	1159	226.27
sanders-r	30	1188	4	420	39.6
williams-w3	18	2769	3	1398	153.83

Table 4.1. Collection statistics for the Enron data set.

In figures 4.1-4.7, we present the distribution of the messages in the folders, per user. What immediately stands out is the few folders that seem to monopolize the data sets. All users have at least one folder holding a large percentage of total messages. We can also observe the extreme imbalance that all users' mailboxes have, with the folders having large variations in the number of messages. The class imbalance problem can be observed in the difference between the largest and the smallest folder.

Content Statistics

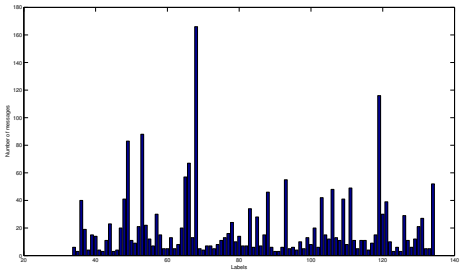

Fig. 4.1. Distribution of the messages over the folders for user beck-s.

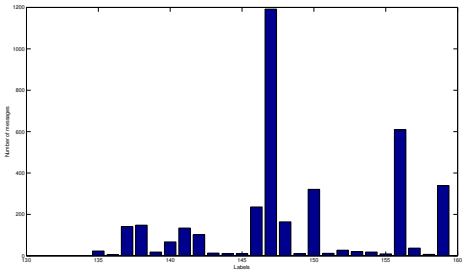

Fig. 4.2. Distribution of the messages over the folders for user farmer-d.

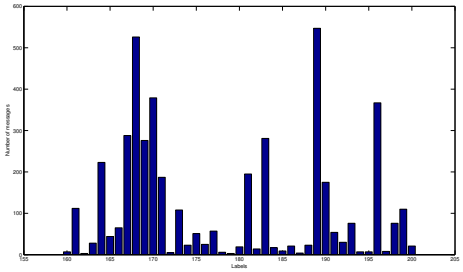

Fig. 4.3. Distribution of the messages over the folders for user kaminski-v.

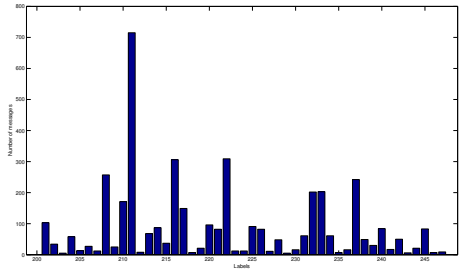

Fig. 4.4. Distribution of the messages over the folders for user kitchen-l.

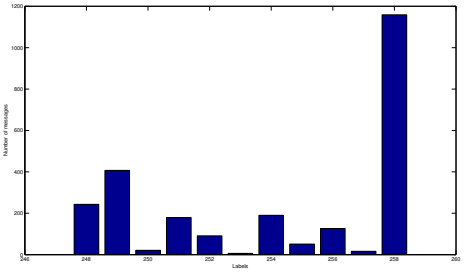

Fig. 4.5. Distribution of the messages over the folders for user lokay-m.

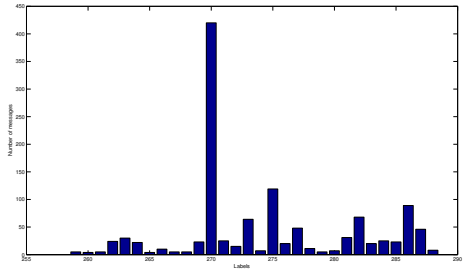

Fig. 4.6. Distribution of the messages over the folders for user sanders-r.

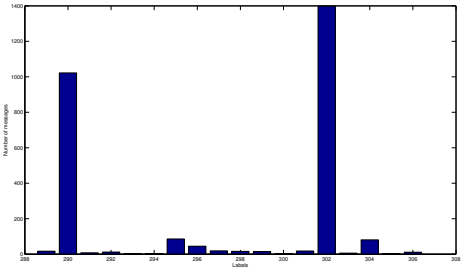

Fig. 4.7. Distribution of the messages over the folders for user williams-w3.

Because email classification could be improved using the several fields of a message, we also analyze the contents of messages. We take a look at the textual fields dates and the participants of the messages.

In figures 4.8-4.14, we present the **time distribution of messages in the classes**. The rows display the folders of the user while the columns are divided by month. The value of the cell contains the number of messages sent in that month that belongs to the respective folder.

What can be observed from these figures is the spreading out of most classes through the several months. This presents a problem for classification using the date field, since that it would be difficult to distinguish the folder of the messages. This pattern occurs in all users of the data set.

While analyzing the **participants**, we came across several **problems with the data**. Several messages are not complete nor coherent in terms of participants. In some messages, there was just one participant (the `From`) in the standard participant fields of the email message headers (there was no `To`, `CC`, `BCC`). The additional participant information was present in non-standard headers (`X-To`, `X-From`, `X-cc` and `X-bcc`), which appear to be generated by the email client. In terms of coherence, in many cases, the same participant is identified in different formats (e.g. Sally Beck@ECT, Beck-S, Sally Beck@ENRON_DEVELOPMENT) and present in both of the headers (standard and non-standard), under different formats. Another common problem are incorrect addresses, which seem to be the result of typos, resulting in invalid addresses.

These problems require a laborious solution, manual identification and correction, which we were determined to avoid. As such, our approach was to cope with the data inconsistency. We extracted all the participants headers (both standard and non-standard). This decision had a big impact on the data, as seen in table 4.2.
The column with Unique Sets' Ratio (USR) represents a ratio between the sets of participants that appear in an email message and the number of folders to which they are associated. We are trying to measure how well a group of participants can identify a folder. This ratio is given by

$$\text{USR} = \frac{\text{Groups of participants that are present in only 1 folder}}{\text{Total number of groups of participants}}. \tag{4.1}$$

Concerning the number of participants, the large amount of different participants in each user's mailbox is due to the way we extracted the data. Even so, we also analyzed the data without the non-standard headers and observed an unusual high number of participants in every message. The reason for this seems to be in the nature of this data set: some messages were sent to an extremely large number of participants (company-wide messages) and many more are group conversations, i.e. messages are shared across several participants - all of which are common in corporate email. Adding the non-standard participants increases in several folds the total number of participants.

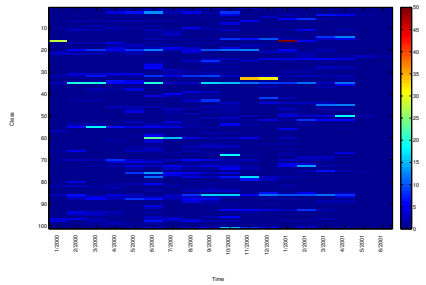

Fig. 4.8. Monthly distribution of the messages over the folders for user beck-s.

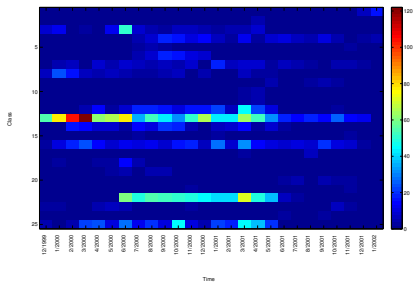

Fig. 4.9. Monthly distribution of the messages of the folders for user farmer-d.

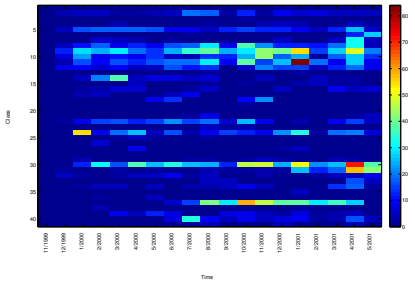

Fig. 4.10. Monthly distribution of the messages of the folders for user kaminski-v.

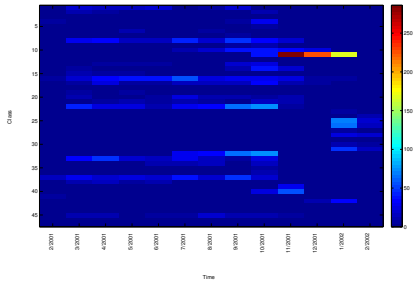

Fig. 4.11. Monthly distribution of the messages of the folders for user kitchen-l.

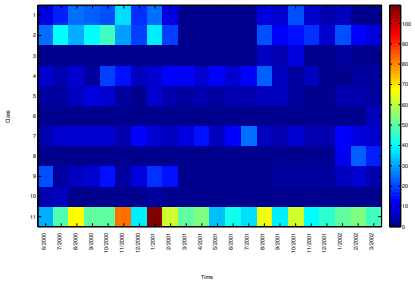

Fig. 4.12. Monthly distribution of the messages of the folders for user lokay-m.

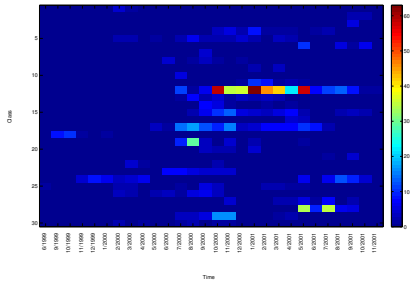

Fig. 4.13. Monthly distribution of the messages of the folders for user sanders-r.

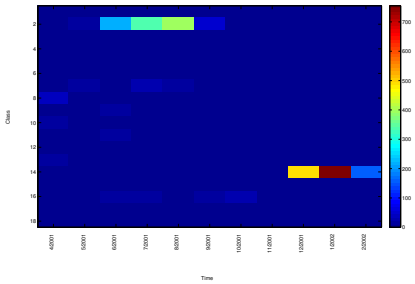

Fig. 4.14. Monthly distribution of the messages of the folders for user williams-w3.

User	Unique sets' ratio	Total num. participants
beck-s	0.8818	6048
farmer-d	0.9202	7329
kaminski-v	0.9145	6645
kitchen-l	0.8820	10146
lokay-m	0.9511	6186
sanders-r	0.9791	2802
williams-w3	0.9797	3303

Table 4.2. Participants statistics for the Enron data set.

4.1.2 INSTICC Data Set

INSTICC (Institute for Systems and Technologies of Information, Control and Communication) [1] is an international scientific, non-profit association whose aim and scope is, among others, to organize and co-sponsor conferences and to publish scientific books and journals. A large volume of their communication is done through email. We had the opportunity to analyze a small subset of their email communication.
The INSTICC data set is a private email message corpus, collected from the communication of several different users during the organization of a conference. The folders of the messages are highly contextualized and are from differente conferences. The fact that users shared a common folder structure, allowed us to perform a merge of the messages into a single set. The rationale behind this decision was due to the small number of messages in most folders. By merging, we expected to increase the number of available examples per folder and to reduce the severity of class imbalance.

Data Set Statistics

We present a similar analysis as the one previously shown for the Enron data set. In figure 4.15, the organization of the mailbox is shown. As in the Enron data set, the distribution of messages is quite imbalanced and a few folders contain the vast majority of the messages.
The time distribution analysis is shown in figure 4.16. Again, it seems that in a certain period of time, there was the arrival of many messages, belonging to several folders. This phenomenon looks likely to cause difficulties in the identification of folders using the time, since that in a certain month, the number of possible folders is very large. The distribution of most messages from the folders among the months is also very balanced, which means that these folders were recurrently used and not just created once and forgotten.

Unlike the Enron data set, the participants fields in the INSTICC data set do not exhibit inconsistency problems in the data. The statistics regarding the participants are presented in table 4.3 and show a clear distinction with the Enron data set. Here, interaction happens between a smaller number of users, with the communication happening in a more personal way, unlike the

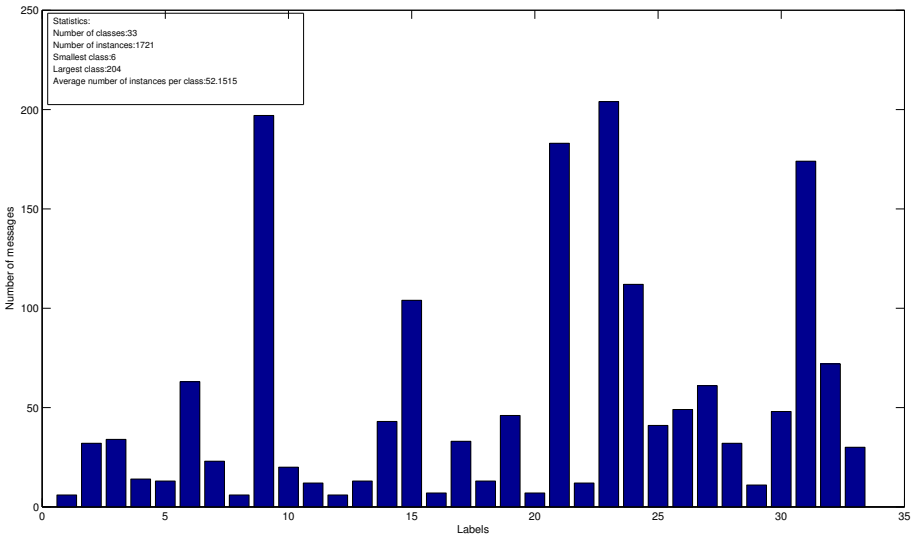

Fig. 4.15. Distribution of the messages over the folders for INSTICC.

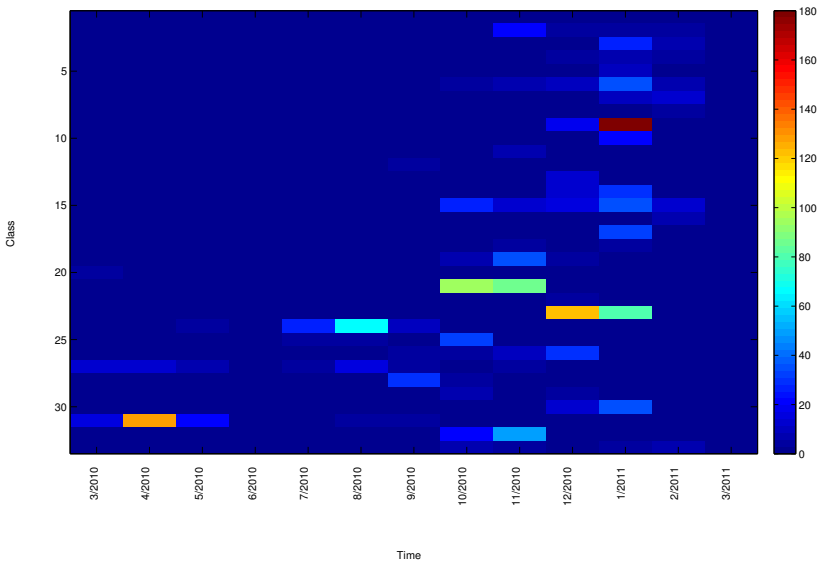

Fig. 4.16. Monthly distribution of the messages of the folders for INSTICC.

common organization-wide messages exchanged by the Enron users.

We also observed that there is an average of 2.2 participants per message. The unique set's ratio is 0.75978, which is lower than any of the Enron users, meaning that the participants do not uniquely identify the folders as accurately as in the Enron data set. This is unexpected given the

Messages w/	2 participants	3 participants	4 participants	5+ participants
Count	1487	164	44	26

Table 4.3. Participant statistics for the INSTICC data set (number of messages with {2, 3, 4, 5+} participants).

low number of participants per message, but might be explained by the addition of more than just addresses in the Enron participants. The additional participant information could increase the uniqueness of the participant sets on the Enron set.

4.2 Tests Description

Our tagging task consists on using classification for tag recommendation. We start by training the classifiers on a training set and then for every email message on the testing set we use the output of the classifiers to tag every email.

In our case, the target classes are the folders where the messages are contained. We process the data by user, that is, messages are divided according to the users. For each user, a test is executed.

Since one of the goals of our work is the study the behavior of learning techniques on the contents of email messages, many tests are done solely on the `body` field of the email messages. The testing procedure is done using tenfold cross-validation. Results are averaged on the folds.

We begin with feature weighting using term weights as feature selection and one classifier (naive Bayes) with all of the feature weighting methods we have implemented (TF, Boolean, Max-Norm TF, TF-Log $\times$ IDF, No-IDF $\times$ Cosine Normalization, No-Normalization). After running these tests, the best representations are picked for further testing.

In feature selection, naive Bayes is again used for classification. This classifier is quite sensitive to the presence of redundant features, thus being adequate to assess the performance of feature weighting and feature selection methods. Using the selected weightings, we apply the feature selection methods (TW, DF, IG, L0-Norm, Fisher and Variance) using a varying number of features (based on the total percentage of features). As previously, the best feature selection methods are chosen for further processing.

For classification, using the best performing feature weighting and feature selection methods, we test several classifiers. We start with the classifiers applied on the textual fields (`subject` and `body`). We apply Naive Bayes, Support Vector Machines, Maximum Entropy, Decision Trees and Winnow. Separately, we also use the Topic Model Classifier on both textual fields. For the participants, we apply the Peoplefier classifier and the standard vector representation using

SVMs. These tests have the goal of assessing how well the chosen techniques work, independently.

The problem of class imbalance is also studied. We use the previously described data-level techniques of Random Sampling and SMOTE. In order to compare the performance of the different algorithms and assess the best operating point, we perform several tests with a varying number of messages per folder. We define the number of messages in the largest folder as the sampling mark. We then balance the folders using one of the methods and a number of messages based on the sampling mark (e.g. 10% of the sampling mark, 50%, etc). Our tests use the Naive Bayes and the Maximum Entropy classifiers, applied on the body of the email messages.

The execution of the system for tagging is done after the best methods have been chosen. We combine the several classifiers into the meta-classifier that uses Majority Voting for determining the best labels. In the end, the most voted is returned.
Since our goal is tagging, returning multiple labels is an option. We also use the top-N rank for verifying the performance of the system.

The execution of the system is assessed through the graphics generated by the graphical analysis tools.

4.3 Results and Analysis

We begin by presenting the results and a brief analysis of the learning techniques. We inspect the performance of the feature weighting methods and determine which feature selection algorithm works best in our data. Classification review is done on different fields of the email messages (subject, body, participants) in separate. We then proceed into the analysis of classifier combination for tagging and finish with a few remarks regarding the tests.

4.3.1 Feature Weighting Tests

Table 4.4 presents the results, in the form of average accuracy and standard deviation, of feature weighting. For the Enron and INSTICC data set, we show the accuracy results obtained when applying the various feature weighting methods and using the naive Bayes classifier. Feature selection using the term weight is applied, but not shown explicitly. The values are averaged over the feature selection steps.
There is a significant variation of accuracy between the different feature weighting methods. In particular, TF-IDF ltc (logarithmic term frequency and cosine normalization) seems to be the worse performing method, while **TF-IDF** has the best results. The remaining methods do not differ significantly from the results obtained by TF-IDF.

User	TF	Boolean	TF-IDF	TF-Log	TF-IDF ltc	TF-Max-Norm
beck-s	0.50 ± 0.06	0.45 ± 0.5	**0.57 ± 0.07**	**0.57 ± 0.05**	0.22 ± 0.05	0.32 ± 0.03
farmer-d	0.77 ± 0.05	0.76 ± 0.04	**0.77 ± 0.03**	0.77 ± 0.04	0.53 ± 0.05	0.66 ± 0.04
kaminski-v	0.67 ± 0.05	0.62 ± 0.05	**0.69 ± 0.03**	0.64 ± 0.05	0.32 ± 0.06	0.48 ± 0.06
kitchen-l	0.49 ± 0.03	0.45 ± 0.04	**0.55 ± 0.03**	0.47 ± 0.04	0.23 ± 0.03	0.33 ± 0.04
lokay-m	**0.82 ± 0.03**	0.80 ± 0.04	**0.82 ± 0.03**	0.81 ± 0.04	0.55 ± 0.04	0.73 ± 0.04
sanders-r	0.70 ± 0.07	0.64 ± 0.06	**0.77 ± 0.06**	0.66 ± 0.07	0.42 ± 0.10	0.51 ± 0.08
williams-w3	**0.90 ± 0.02**	**0.90 ± 0.02**	0.87 ± 0.03	**0.90 ± 0.02**	0.87 ± 0.03	0.90 ± 0.03
INSTICC	**0.78 ± 0.03**	0.77 ± 0.03	0.73 ± 0.03	**0.78 ± 0.03**	0.57 ± 0.06	0.70 ± 0.07

Table 4.4. Accuracy results of feature weighting methods (averaged over the feature selection steps). The best performing methods on each user's sub set are highlighted in bold.

4.3.2 Feature Selection Tests

The accuracy results of applying feature selection are shown in table 4.5 for the Enron data set and figure 4.17 for the INSTICC data set. TF-IDF is chosen for feature weighting. In the table, the results are once again averaged by the feature selection steps, while the figure presents the accuracy of classification in the various steps.

User	DF	Fish-MS	Fish-SS	Fish-SSS	IG	$\ell_0 N1$	$\ell_0 N2$	TW	Var.
beck-s	0.57±0.07	0.56±0.07	0.58±0.07	0.58±0.06	0.58±0.07	0.58±0.07	0.58±0.07	0.57±0.07	0.58±0.07
farmer-d	0.77±0.03	0.76±0.03	0.79±0.03	0.79±0.03	0.79±0.03	0.77±0.03	0.77±0.03	0.77±0.03	0.77±0.03
kaminski-v	0.69±0.04	0.68±0.03	0.69±0.03	0.69±0.03	0.70±0.03	0.69±0.04	0.70±0.04	0.69±0.03	0.69±0.03
kitchen-l	0.55±0.03	0.54±0.02	0.55±0.03	0.55±0.03	0.55±0.03	0.55±0.03	0.55±0.03	0.55±0.02	0.55±0.02
lokay-m	0.82±0.02	0.80±0.03	0.82±0.02	0.83±0.03	0.83±0.02	0.82±0.02	0.82±0.02	0.82±0.03	0.81±0.02
sanders-r	0.77±0.06	0.73±0.06	0.80±0.04	0.80±0.04	0.81±0.04	0.77±0.06	0.77±0.06	0.77±0.06	0.77±0.06
williams-w3	0.87±0.03	0.87±0.02	0.90±0.02	0.90±0.02	0.89±0.02	0.87±0.03	0.87±0.03	0.87±0.03	0.87±0.03
average	0.72±0.04	0.70±0.04	0.73±0.03	0.73±0.03	074.±0.03	0.72±0.04	0.72±0.04	0.72±0.04	0.72±0.04

Table 4.5. Results of feature selection methods on the Enron data set.

It can be observed that, in general, feature selection methods have little impact on the classifier's accuracy. As seen in the table 4.5, in average, all methods are in par in terms of accuracy. However, using feature selection can improve the run time of the system due to the removal of a significant number of features, thus still being beneficial.

The best performing method in most data sets is **Information Gain**. This is in line with previous studies [84] that described the positive impact of this method.

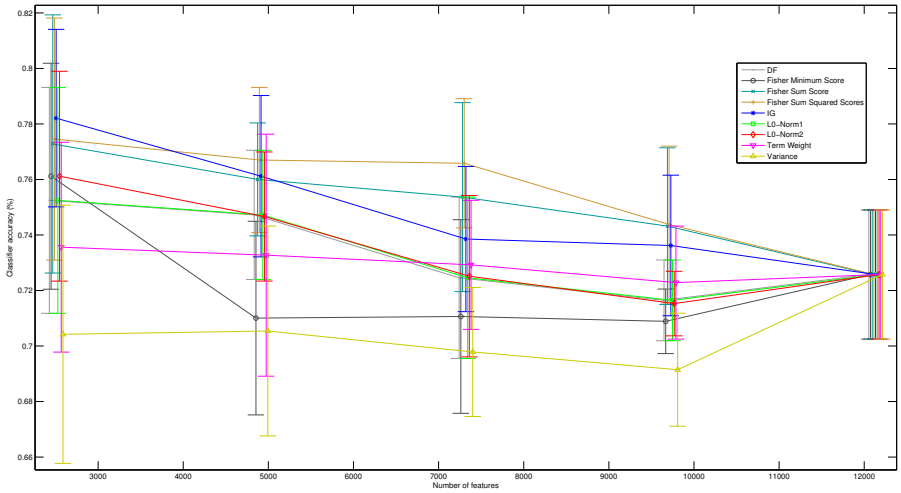

Fig. 4.17. Accuracy of feature selection on the INSTICC data set.

4.3.3 Classification Tests

The classification results of the `body` and `subject` fields are shown in table 4.6, for the Enron data set. For the INSTICC data set, they are presented in figure 4.18 (body) and 4.19 (subject). Feature weighting is applied using `TF-IDF`, while feature selection is done with `IG`. All values are averaged across the feature selection steps.
Results vary significantly between classifiers, in some cases over 40% as seen in `beck-s`. **Maximum Entropy** is the best performing algorithm, consistently achieving the best results. Decision Trees perform poorly in most of the users, although it manages to be in par in user `williams-w`.

In general, the body field produces better accuracy results than the subject field. With Maximum Entropy, the difference is significant, reaching over 30% in the user `kitchen-l`.

In addition to the textual fields, we also analyze the behavior of the classifiers that are based on the participants - i.e. using participants as a vector-space representation (`Participants SVMs`, `Peoplefier+SVMs`) and using the graph representation (`Peoplefier`) - and the topic model representation classifier (on both the subject and the body of messages).

The accuracy results for these tests are shown in table 4.7. The `Peoplefier+SVMs` is equivalent to the `PeoplefierV2` approach described in section 3.4.2. No feature selection is applied on these tests. The results are averaged over the 10 folds. The poor accuracy results for the topic model classifiers immediately stand out. Even in user `williams-w3`, which is consistently well classified, the topic model classifier, when using the subject, presents an average accuracy of

User	Balanced Winnow	Decision Tree	SVMs	Max. Entropy	Naive Bayes	Winnow
beck-s (body)	0.58±0.06	0.17±0.04	0.59±0.05	0.76±0.17	0.58±0.07	0.40±0.07
beck-s (subject)	0.52±0.08	0.18±0.03	0.55±0.06	0.56±0.07	0.53±0.09	0.45±0.07
farmer-d (body)	0.77±0.04	0.56±0.03	0.76±0.04	0.81±0.05	0.79±0.03	0.69±0.03
farmer-d (subject)	0.73±0.03	0.51±0.04	0.75±0.02	0.76±0.04	0.71±0.03	0.60±0.05
kaminski-v (body)	0.66±0.03	0.27±0.06	0.67±0.04	0.72±0.05	0.70±0.03	0.48±0.05
kaminski-v (subject)	0.55±0.05	0.19±0.03	0.57±0.04	0.58±0.04	0.55±0.04	0.42±0.05
kitchen-l (body)	0.53±0.04	0.25±0.04	0.54±0.04	0.85±0.15	0.55±0.03	0.34±0.04
kitchen-l (subject)	0.48±0.03	0.25±0.04	0.50±0.03	0.51±0.04	0.49±0.04	0.37±0.04
lokay-m (body)	0.79±0.05	0.61±0.03	0.78±0.05	0.86±0.06	0.83±0.02	0.73±0.05
lokay-m (subject)	0.71±0.06	0.53±0.03	0.72±0.07	0.75±0.05	0.73±0.06	0.65±0.05
sanders-r (body)	0.78±0.08	0.52±0.07	0.75±0.09	0.80±0.08	0.81±0.04	0.62±0.08
sanders-r (subject)	0.68±0.06	0.44±0.08	0.71±0.06	0.71±0.05	0.71±0.04	0.60±0.07
williams-w3 (body)	0.92±0.02	0.91±0.02	0.93±0.02	0.94±0.02	0.89±0.02	0.88±0.03
williams-w3 (subject)	0.89±0.03	0.89±0.02	0.90±0.03	0.92±0.02	0.83±0.03	0.86±0.03

Table 4.6. Results of classification methods (average) on the body and subject field of the Enron data set.

37%.

User	Participants (SVMs)	Peoplefier	Peoplefier+SVMs	TM (body)	TM (subject)
beck-s	0.58±0.06	0.33±0.06	0.26±0.02	0.25±0.03	0.09±0.04
farmer-d	0.79±0.03	0.60±0.03	0.37±0.02	0.34±0.04	0.29±0.17
kaminski-v	0.62±0.03	0.24±0.03	0.41±0.02	0.14±0.04	0.10±0.04
kitchen-l	0.59±0.03	0.28±0.04	0.26±0.01	0.19±0.04	0.15±0.07
lokay-m	0.86±0.02	0.79±0.04	0.48±0.02	0.15±0.22	0.11±0.05
sanders-r	0.85±0.04	0.54±0.06	0.54±0.05	0.41±0.08	0.35±0.07
williams-w3	0.97±0.01	0.86±0.03	0.63±0.04	0.78±0.20	0.37±0.03
INSTICC	0.61±0.07	0.40±0.06	0.44±0.07	0.31±0.10	0.26±0.06

Table 4.7. Results of classification using alternative email message fields.

The vectorized participants (using SVMs) obtain the best results among these classifiers. It is in par with the Balanced Winnow and SVMs when applied on the body field. Peoplefier and PeoplefierV2 obtain disappointing results. Altough they attain better results than the topic model classifiers, their accuracies are lower than that of the vectorized participants.

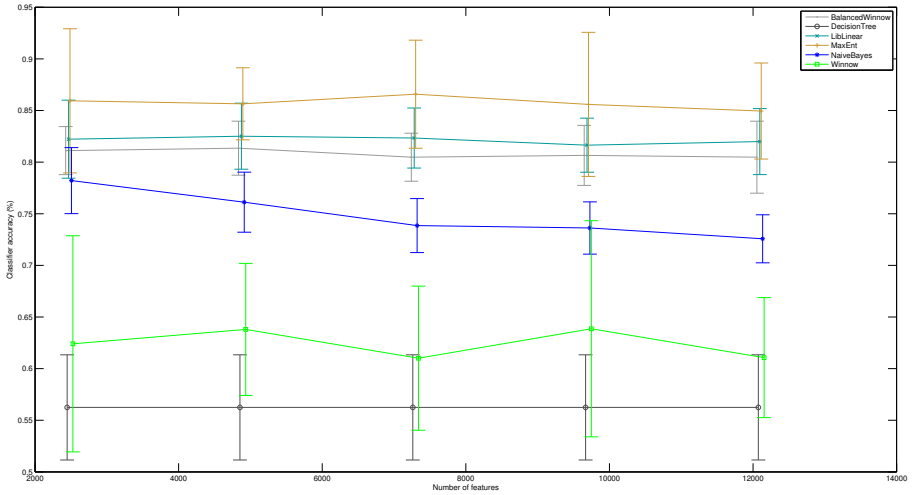

Fig. 4.18. Accuracy of classification algorithms on the body field of the INSTICC data set.

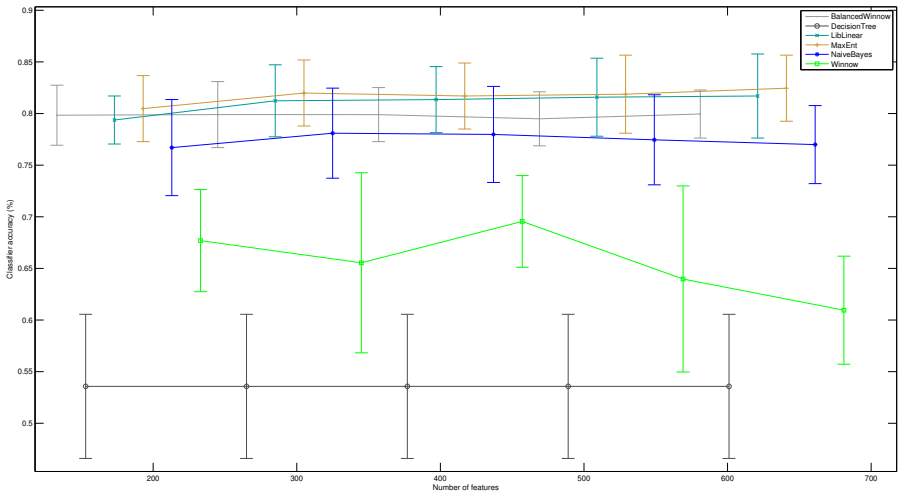

Fig. 4.19. Accuracy of classification algorithms on the subject field of the INSTICC data set.

4.3.4 Class Imbalance Tests

We begin this subsection by presenting the results of classification without the use of any balancing technique. These results serve as a baseline to which we compare the results of the balancing tests. We then present and discuss the results of the tests using Random Sampling and SMOTE.

Table 4.8 shows the baseline results of classification on the data sets. We can observe that the

data set of `beck-s` has the lowest accuracy result while also being the one with the most number of folders (over twice than the next one, `kitchen-l`). In general, apart from `williams-w3`, no other data set obtains over 90% of accuracy.

User	naive-Bayes	Maximum Entropy
beck-s	0.60 ± 0.07	0.72 ± 0.16
farmer-d	0.79 ± 0.03	0.81 ± 0.04
kaminski-v	0.70 ± 0.03	0.72 ± 0.07
kitchen-l	0.54 ± 0.03	0.86 ± 0.16
lokay-m	0.84 ± 0.02	0.85 ± 0.06
sanders-r	0.79 ± 0.06	0.80 ± 0.07
williams-w3	0.88 ± 0.03	0.94 ± 0.01
INSTICC	0.78 ± 0.03	0.88 ± 0.06

Table 4.8. Classification results for a 10-fold cross validation (average accuracy ± standard deviation) without balancing techniques. Feature selection is done by Information Gain, keeping 20% of the features.

Figure 4.20 shows the confusion matrix of the classification results of `beck-s`. The large number of folders, allied to the large variation of the folder sizes causes a large number of misclassifications. The ROC curve in Figure 4.21 backs this up. The curve is averaged over the 10-folds and over all of the classes in the data set. Its TPR varies between 0.55 and 0.65, which is relatively poor.

The case of `williams-w3` data set (and its peer set `lokay-m`) is special, since it contains a very low number of folders and two of them contain over 80% of the messages. By inspecting the confusion matrix, in Figure 4.22, we can observe that one of these folders monopolizes classification, that is, several of the smaller folders are incorrectly classified in the larger folder. But because these misclassified folders are so small, their impact in the accuracy results is unnoticeable, resulting in a high accuracy. The ROC curve of `williams` is shown in Figure 4.23 and is similar to the curve of `beck-s`, with the TPR indicator varying between 0.5 and 0.7, which is a poor result.

Classification results (average accuracy ± standard deviation) using Random Sampling are shown in Table 4.9. It can be seen that the application of such a simple balancing method significantly improves the accuracy on all data sets. The use of oversampling (i.e. when using 100% of the instances of the largest folder), which happens on the folders with fewer documents, seems to be more effective than undersampling (which happens when only 10% of the instances are kept). However, these results can be misleading. By using random oversampling, we are repeatedly selecting the same messages from small folders. A phenomenon that can occur in the testing phase is that the same messages are being presented to the classifier (which was trained on them) - this is basically a case of overfitting. As such, accuracy is inflated.

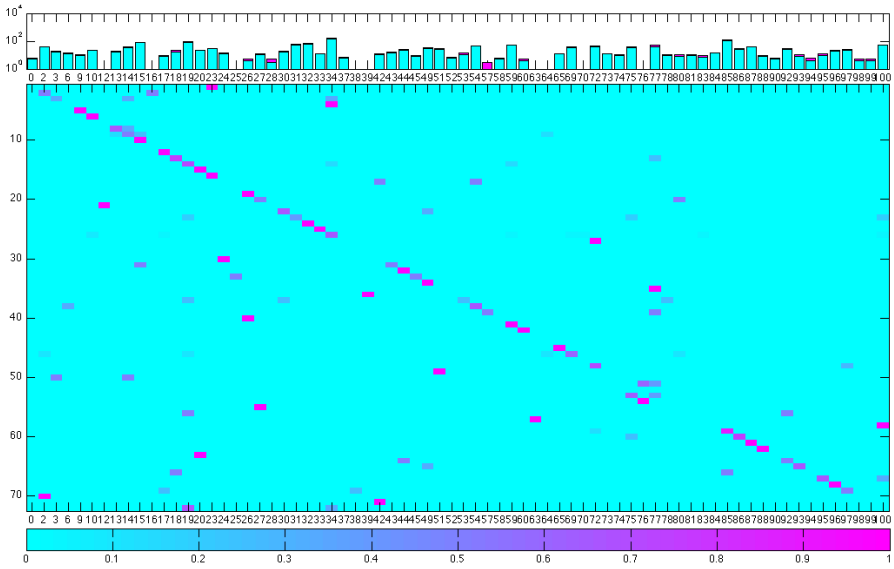

Fig. 4.20. Confusion matrix for beck-s, from the classification tests without any balancing technique. Top sub-figure represents the number of messages used in training (in blue) and testing (in pink). South sub-figure is the matrix with the ordered results.

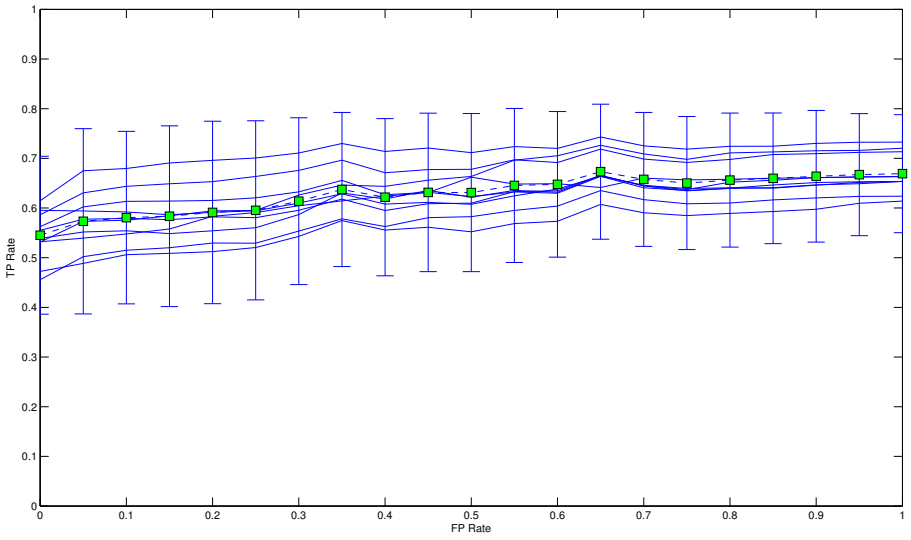

Fig. 4.21. ROC curve for beck-s, from the classification tests without any balancing technique.

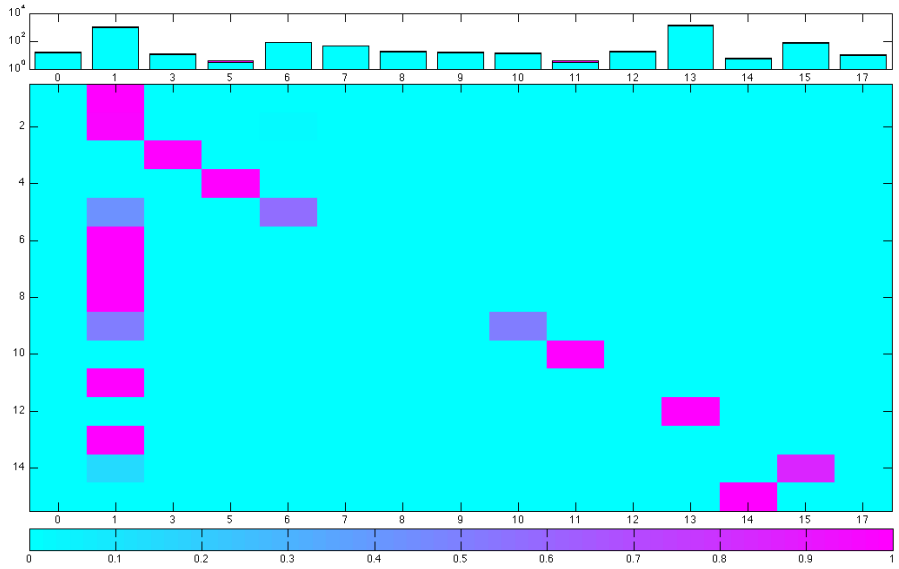

Fig. 4.22. Confusion matrix for williams-w3, from the classification tests without any balancing technique. This set is a special case due to its extreme imbalance, with two folders containing over 80% of the messages.

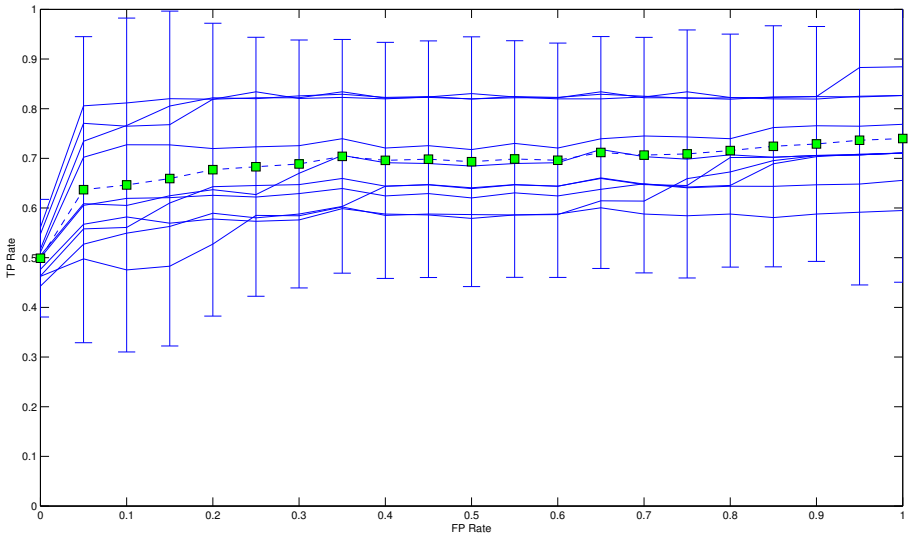

Fig. 4.23. ROC curve for williams-w3, from the classification tests without any balancing technique.

User	10%	60%	100%
beck-s	0.90 ± 0.06	0.93 ± 0.02	0.93 ± 0.02
farmer-d	0.96 ± 0.02	0.97 ± 0.00	0.98 ± 0.01
kaminski-v	0.90 ± 0.04	0.95 ± 0.01	0.98 ± 0.01
kitchen-l	0.95 ± 0.05	0.96 ± 0.02	0.97 ± 0.01
lokay-m	0.93 ± 0.04	0.99 ± 0.01	0.99 ± 0.01
sanders-r	0.97 ± 0.02	0.99 ± 0.01	0.99 ± 0.01
williams-w3	0.99 ± 0.01	1.00 ± 0.00	1.00 ± 0.00
INSTICC	0.91 ± 0.05	0.98 ± 0.01	0.99 ± 0.01

Table 4.9. Classification results for a 10-fold cross validation (average accuracy ± standard deviation) using Maximum Entropy and Random Sampling (feature selection using Information Gain, keeping 20% of the features). Each column indicates the number of messages that are used in classification (in regards to the sampling mark) and the corresponding accuracy.

User	10%	60%	100%
beck-s	0.82 ± 0.12	0.90 ± 0.08	0.91 ± 0.05
farmer-d	0.90 ± 0.03	0.93 ± 0.03	0.94 ± 0.02
kaminski-v	0.82 ± 0.04	0.90 ± 0.02	0.92 ± 0.02
kitchen-l	0.95 ± 0.10	0.96 ± 0.05	0.96 ± 0.05
lokay-m	0.87 ± 0.06	0.93 ± 0.03	0.95 ± 0.02
sanders-r	0.93 ± 0.03	0.96 ± 0.01	0.97 ± 0.02
williams-w3	0.96 ± 0.02	0.98 ± 0.01	0.99 ± 0.01
INSTICC	0.86 ± 0.07	0.91 ± 0.03	0.93 ± 0.02

Table 4.10. Classification results for a 10-fold cross validation (average accuracy ± standard deviation) using Maximum Entropy and SMOTE (feature selection using Information Gain, keeping 20% of the features). Each column indicates the number of messages that are used in classification (in regards to the sampling mark) and the corresponding accuracy.

We have assessed the results of Random Sampling by using SMOTE, due to its characteristic of avoiding overfitting in oversampling. The results of these tests are shown in Table 4.10. It can be observed that using SMOTE improves the classification results, when compared to the baseline values. Results are inferior to the ones obtained from Random Sampling but we posit that generalization is far better.
For comparison with the baseline results, in Figure 4.24 we show the confusion matrix of `beck-s` when SMOTE was applied. The usage of balancing significantly improves the classification results in an imbalanced and distributed data set, such as the one from `beck-s`.

4.3.5 Tagging using Classifier Combination Tests

In table 4.11, the results of the Majority Voting combination technique are presented. The values are averaged over the 10 folds. We present three different tests and for each, the best result

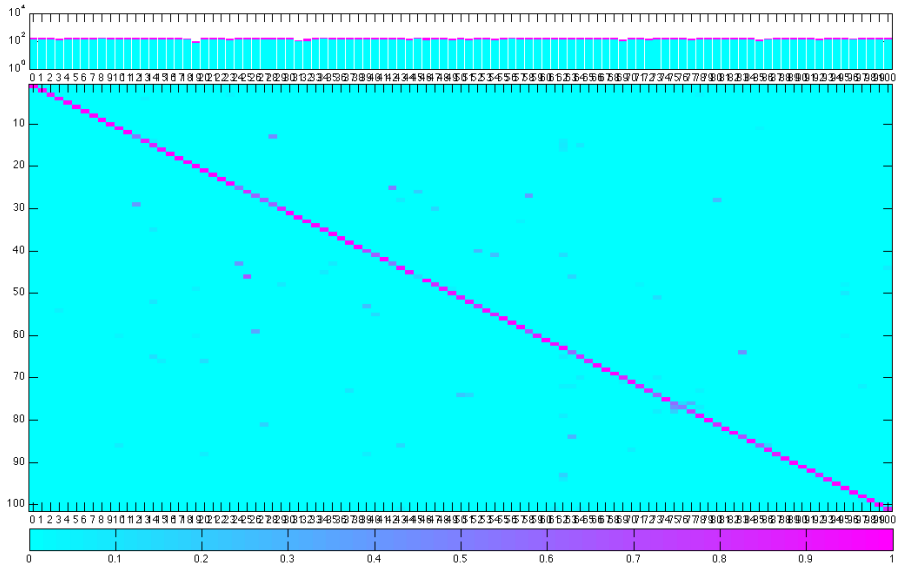

Fig. 4.24. Confusion matrix for beck-s, from the classification tests with SMOTE. Folders were oversampled to 100% of the sampling mark (i.e. undersampling is not applied).

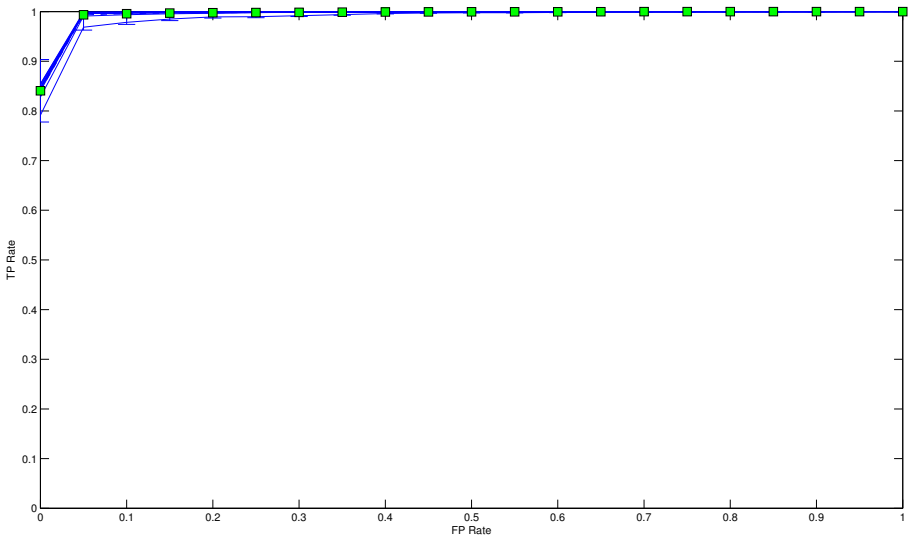

Fig. 4.25. ROC curve for beck-s, from the classification tests with SMOTE. Folders were oversampled to 100% of the sampling mark (i.e. undersampling is not applied).

(top-1) and the three best labels (top-3) are used to evaluate the accuracy.

The first type of combination, `Subject+Body` uses Maximum Entropy classifiers on the subject and body fields, respectively. The second type of combination, `Subject+Body+Participants`, uses Maximum Entropy on the subject, body and participant fields. The last type of combination, `Subject+Body+Peoplefier+BodyTopics+SubjectTopics` applies Maximum Entropy on the subject and body fields, Peoplefier on the participant fields and the Topic Model Classifier with the subject and body.

Combination	**INSTICC**	**beck-s**	**farmer-d**	**kaminski-v**	**kitchen-l**	**lokay-m**	**sanders-r**	**williams-w3**
Subject+Body (top-1)	0.9239	0.8549	0.9188	0.8608	0.7285	0.9731	0.9167	0.9545
Subject+Body (top-3)	0.9849	0.9640	0.9801	0.9978	0.9044	0.9960	**1.0000**	**1.0000**
Subject+Body+ Participants (top-1)	0.9326	0.9127	0.9469	0.9375	0.8381	0.9727	0.9823	0.9913
Subject+Body+ Participants (top-3)	**0.9861**	**0.9782**	**0.9880**	**0.9984**	**0.9462**	**0.9968**	**1.0000**	**1.0000**
Subject+Body+ Peoplefier+BodyTopics +SubjectTopics (top-1)	0.7780	0.5226	0.6217	0.4583	0.3731	0.9096	0.6069	0.8906
Subject+Body+ Peoplefier+BodyTopics +SubjectTopics (top-3)	0.9698	0.9482	0.9801	0.9665	0.9016	0.9851	0.9992	**1.0000**

Table 4.11. Accuracy results of classifier combination on the INSTICC and Enron data sets, using top-1 and top-3 evaluation.

The top-1 results of classifier combination present a surprise when compared with the results of the independent classifiers. Accuracy is improved in a very significant way, across all users, in all but one combination (`Subject+Body+Peoplefier+BodyTopics+SubjectTopics`). As expected, evaluating with top-N improves the results.

The best performance is obtained using the `Subject+Body+Participants` combination. With top-1, a small increase in the accuracy is observed when compared to `Subject+Body`'s top-1. The top-3 results are very similar between `Subject+Body+Participants` and `Subject+Body`, with the former, again, obtaining better results.

The combination of `Subject+Body+Peoplefier+BodyTopics+SubjectTopics` performs worse than any of the other two combinations, with a difference in accuracy of up to 50% in user

`kitchen-l`. top-1's accuracy values can be below Maximum Entropy's independent execution on the body field. With the top-3, accuracy of this combination improves significantly, becoming similar in terms of results with the other combinations.

We also analyze how imbalance affects the results of classifier combination. It is observed that classification improves significantly, even in a data set with many folders, such as the case of `beck-s` (figure 4.26). The problem of class imbalance is still present, but it is greatly reduced. The effect of `williams-w3`'s dominant classes is also canceled by combining classifiers (figure 4.27).

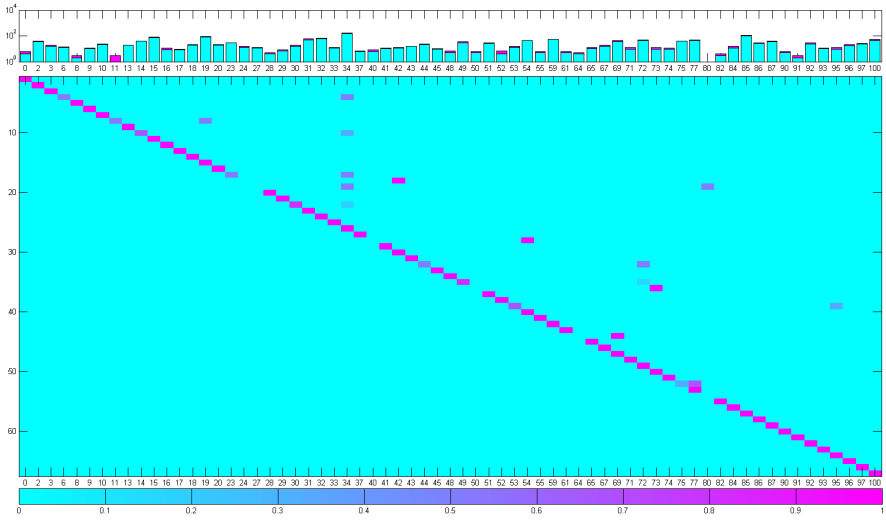

Fig. 4.26. Confusion matrix for beck-s, from the classification tests with classifier combination (subject + body).

4.3.6 Discussion of Experimental Results

The tests and results obtained in this chapter provide an interesting insight into the field of email classification and tagging.

The usage of feature weighting should be applied with caution. Even though there isn't a significant difference in the performance of the classifier when using most of the methods, TF-IDF ltc's results are suprisingly low. One reason for this can be the nature of email - the text is shorter than commonly found in other fields of text categorization (e.g. news articles or scientific abstracts). Normalization might affect the results in a harmful way.

Feature selection in average, shows little variation between the different methods. But when analyzing the feature selection results at each step, one can observe that Information Gain has the best overall performance. The usage of feature selection seems to be beneficial, given that reducing the number of features improves the run time of the system (e.g. in the INSTICC set, the number of kept features was 4 times lower than the original number of features).

The results of the classifiers differ significantly. In the textual fields, subject and body, large variations exist between the classifiers. Maximum Entropy performs best overall. Apart from Decision Trees, the remaining classifiers show similar performance. The topic model classifier presents disappointing results, the lowest of all, but not very different from Decision Trees. Classification using topic models should be retought, as this approach does not perform as expected.

In the participant fields, the previous analysis seemed to indicate that using this information for classification could prove difficult (given the results of USR). Between the two different participant representations, the best performing method is the usage of participants as features on a vector-space representation, outperforming the naive approach of Peoplefier and even the vectorized Peoplefier. It seems likely that this graph representation does not suit the task of prediction.

Common among the results is the impact of the data sets. No classification method performs consistently across the different users. The impact of the data is evident, with `beck-s` being the most difficult case, due to its great imbalance (101 labels). The results on `williams-w3` are misleading, due to the mailbox organization of this user (explained in detail in [4]) - two folders contain over 80% of the messages.
This is in conformity with previous studies. These results show that the imbalance problem affects the classification accuracy. The use of the Random Sampling technique to balance the data improves classification results, but these are inflated due to the overfitting caused by oversampling. On the other hand, the SMOTE oversampling technique avoids overfitting, and its results also show improvements as compared to the baseline results. These results, also better than the baseline error, are more reliable than those of Random Sampling.

Classifier combination using the simple Majority Voting method also presents a significant improvement on the single classifiers' performance, when using "strong" classifiers. Combination with "weak" classifiers negatively affects the performance. This is expected, due to the nature of the Majority Voting algorithm.
When using the textual fields in conjunction with the participants in a vectorized representation, accuracy is high across all of the users. The usage of top-3 evaluation presents a small increase on the performance, as compared to the top-1 evaluation.

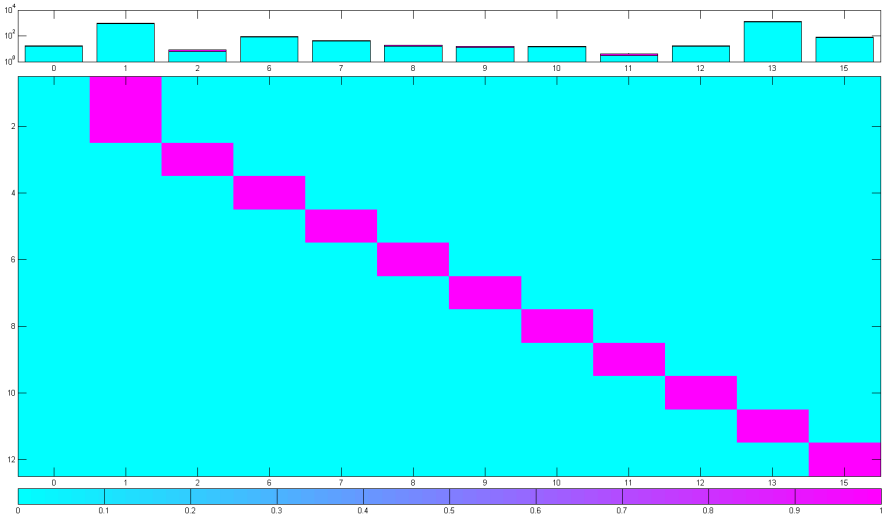

Fig. 4.27. Confusion matrix for williams-w3, from the classification tests with classifier combination (subject + body).

5

Conclusions

Email overload is a prevalent problem of the modern world. Even before Whittaker et al coined the expression, signs of overload were already being noticed. Several approaches have been proposed, such as automatic foldering, priority inbox, and many more. But still, no definite solution seems to exist. In this work, we have focused on automatic foldering and email message tagging.

Our work uses learning techniques in order to automatically assign predefined tags to email messages. We use supervised learning to generate a model for a user's mailbox, making our approach a personalized one. For this work, the tags were the folders that the users had.

A significant portion of our work is the study of different algorithms and techniques for the different steps of an email classification system. Due to the richness of the textual fields of email messages, much effort was spent on the techniques related to it. Even though some of these techniques are not text-specific, they were tested in the context of the text fields from email messages:

- We reviewed several feature weighting methods and came to the conclusion that TF-IDF produces superior results when compared to other methods;
- Feature selection algorithms based on filtering were studied, with IG achieving the best results, even though the difference to other algorithms is minimal;
- Classification has very distinct results, enhancing the importance of selecting a good classifier;
- The class imbalance problem was studied and it is understood that this phenomenon affects classification in a very significant way.

We also analyzed the classification task on other fields of an email message, namely the participants fields (from, to, cc, bcc). The participants were represented with a vector-space model but also through a graph, inspired from social analysis. Results of classification with the participants were generally low.

Two new classifiers were proposed: one based on a graph of the interaction of participants and, another, based on topic models. Their performance is rather dissapoint, with worse results than

the general classifier.

Due to a lack of a publicly tagged email data set, we took the simplistic approach of using foldered data sets. We used the publicly available Enron Corpus and a private set, from INSTICC. In our process of study, we transformed the folders into tags. An extensive analysis of the data is presented, with approaches that, to the best of our knowledge, have not been explored before.

To merge all the different sources of information, we apply a classifier combination method, Majority Voting, that combines the classifiers that were trained on different sets of features. Our tests demonstrate the usefulness of this technique, with consistent results of over 90% of accuracy across all users. The use of a top-N approach to evaluation makes our solution effective for tagging. Our tests also show that classifier combination can mitigate the problems of imbalanced data sets.

The field of email classification and tagging continues to attract the attention of the research community since is still an open problem. Our work is just one more contribution.

References

1. Institute for systems and technologies of information, control and communication. `http://www.insticc.org/`.
2. Douglas Aberdeen, O. Pacovsky, and Andrew Slater. The Learning Behind Gmail Priority Inbox. In *Neural Information Processing Systems*, 2010.
3. Charu C. Aggarwal, editor. *Social Network Data Analytics*. Springer, 2011.
4. Ron Bekkerman and Andrew Mccallum. Automatic Categorization of Email into Folders : Benchmark Experiments on Enron and SRI Corpora. Technical report, University of Massachusetts, 2004.
5. P. Bermejo, J. Gámez, and J. Puerta. Improving the performance of Naive Bayes multinomial in e-mail foldering by introducing distribution-based balance of datasets. *Expert Systems with Applications*, 38(3):2072–2080, March 2011.
6. Michael W. Berry and Jacob Kogan, editors. *Text Mining - Applications and Theory*. 2010.
7. Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 1st editio edition, 2006.
8. David M Blei. Introduction to Probabilistic Topic Models. *Communications of the ACM*, pages 1–16, 2011.
9. blekko. slash the web! `http://blekko.com/ws/+/about`, 2011.
10. B. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *Proc. of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152. ACM Press, 1992.
11. L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
12. Jake D. Brutlag and Christopher Meek. Challenges of the Email Domain for Text Classification. In *International Conference on Machine Learning*, 2000.
13. Christopher J C Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.
14. Chih-chung Chang and Chih-jen Lin. LIBSVM : A Library for Support Vector Machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):1–39, 2011.
15. Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. SMOTE : Synthetic Minority Over-sampling Technique. *Artificial Intelligence*, 16:321–357, 2002.
16. Nitesh V Chawla, Nathalie Japkowicz, and Prentice Drive. Editorial : Special Issue on Learning from Imbalanced Data. *ACM SIGKDD Explorations*, 6(1):30–39, 2004.
17. Andrea Civan, William Jones, Predrag Klasnja, and Harry Bruce. Better to organize personal information by folders or by tags?: The devil is in the details. *Proceedings of the American Society for Information Science and Technology*, 45(1):1–13, June 2009.

18. Aaron Clauset, M E J Newman, and Cristopher Moore. Finding community structure in very large networks. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 70(6):6, December 2004.
19. William W. Cohen. Enron email dataset. `http://www.cs.cmu.edu/~enron/`.
20. William W. Cohen. Learning Rules that Classify E-Mail. In *AAAI Spring Symposium on ML and IR*, 1996.
21. T. Cover and J. Thomas. *Elements of Information Theory*. John Wiley & Sons, 1991.
22. Gabor Cselle. *Organizing Email*. PhD thesis, ETH Zurich, 2006.
23. Sanmay Das. Filters , Wrappers and a Boosting-Based Hybrid for Feature Selection. In *International Conference on Machine Learning*, 1994.
24. delicious. About. `http://www.delicious.com/about`, June 2011.
25. Peter J Denning. Electronic Junk. *Communications of the ACM*, 25(3), 1982.
26. Mark Dredze, Tessa Lau, and Nicholas Kushmerick. Automatically classifying emails into activities. *Proceedings of the 11th international conference on Intelligent user interfaces - IUI '06*, page 70, 2006.
27. Mark Dredze, Hanna M. Wallach, Danny Puller, and Fernando Pereira. Generating summary keywords for emails using topics. In *International conference on Intelligent user interfaces*, page 199, New York, New York, USA, 2008. ACM Press.
28. R. Duda, P. Hart, and D. Stork. *Pattern Classification*. John Wiley & Sons, 2nd edition, 2001.
29. RIchard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. 2000.
30. B. Efron. The jackknife, the bootstrap and other resampling plans. *Society for Industrial and Applied Mathematics (SIAM)*, 1982.
31. F. Escolano, P. Suau, and B. Bonev. *Information Theory in Computer Vision and Pattern Recognition*. Springer, 2009.
32. Richárd Farkas, Berend Gábor, István Hegedűs, András Kárpáti, and Balázs Krich. Automatic free-text-tagging of online news archives. In *ECAI 2010*, 2010.
33. Tom Fawcett. ROC Graphs : Notes and Practical Considerations for Researchers. Technical report, HP Laboratories, Palo Alto, CA, 2004.
34. Artur J. Ferreira and A. T. Figueiredo. Feature Transformation and Reduction for Text Classification. In *International Workshop on Pattern Recognition in Information Systems*, pages 72–81, 2010.
35. Danyel Fisher, a. J. Brush, Eric Gleave, and Marc a. Smith. Revisiting Whittaker & Sidner's "email overload" ten years later. *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work - CSCW '06*, page 309, 2006.
36. flickr. About flickr. `http://www.flickr.com/about`, June 2011.
37. Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *Thirteenth International Conference on Machine Learning*, pages 148–156, Bari, Italy, 1996.
38. Gmail. Google's approach to email - top 10 reasons to use gmail. `http://mail.google.com/mail/help/intl/en/about.html`, 2011.
39. I. Guyon, S. Gunn, M. Nikravesh, and L. Zadeh (Editors). *Feature Extraction, Foundations and Applications*. Springer, 2006.
40. T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2nd edition, 2001.
41. Nathalie Japkowicz and Shaju Stephen. The Class Imbalance Problem: A Systematic Study. *Intelligent Data Analysis*, 6(5), 2002.
42. T. Joachims. *Learning to Classify Text Using Support Vector Machines*. Kluwer Academic Publishers, 2001.

43. Thorsten Joachims. Text Categorization with Support Vector Machines - Learning with Many Relevant Features. In *European Conference on Machine Learning*, pages 137–142, 1998.
44. Shih-wen Ke, Chris Bowerman, and Michael Oakes. PERC : A Personal Email Classifier. In *European Conference on Information Retrieval*, pages 460 – 463, 2006.
45. Victoria Keiser and Thomas G Dietterich. Evaluating Online Text Classification Algorithms for Email Prediction in TaskTracer. In *Conference on Email and Anti-Spam*, page 4, 2009.
46. Bryan Klimt and Yiming Yang. Introducing the Enron Corpus. In *Conference on Email and Anti-Spam*, 2004.
47. Bryan Klimt and Yiming Yang. The Enron Corpus : A New Dataset for Email Classification Research. In *European Conference on Machine Learning*, 2004.
48. Yehuda Koren, Edo Liberty, Yoelle Maarek, and Roman Sandler. Automatically Tagging Email by Leveraging Other Users ' Folders. In *Conference on Knowledge Discovery and Data Mining*, San-Diego, USA, 2011.
49. Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4):285–318, April 1988.
50. Huan Liu and Hiroshi Motoda, editors. *Computational Methods of Feature Selection*, volume 67. Chapman & Hall/CRC, January 2008.
51. Huan Liu and Lei Yu. Toward Integrating Feature Selection Algorithms for Classification and Clustering. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):491–502, 2005.
52. C. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.
53. Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*, volume 26. June 1999.
54. MathWorks. Matlab - the language of technical computing. `http://www.mathworks.com/products/matlab/`, June 2011.
55. Andrew Kachites McCallum. Mallet: A machine learning for language toolkit. http://mallet.cs.umass.edu, 2002.
56. Andrei Mikheev. Periods, Capitalized Words, etc. *Computational Linguistics*, 28(3), 2002.
57. Tom M. Mitchell. *Machine Learning*, volume 26. May 1997.
58. Kamal Nigam, John Lafferty, and Andrew Mccallum. Using Maximum Entropy for Text Classification. In *IJCAI Workshop on Machine Learning for Information Filtering*, 1999.
59. National Institute of Standards and Technology. Topic detection and tracking evaluation. `http://www.itl.nist.gov/iad/mig/tests/tdt/`, 2008.
60. Oracle. Learn about java technology. `http://www.java.com/en/about/`, June 2011.
61. M. F. Porter. *An algorithm for suffix stripping*, pages 313–316. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.
62. J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, October 1986.
63. Kerry Rodden and Michael Leggett. Best of Both Worlds : Improving Gmail Labels with the Affordances of Folders. In *ACM Conference on Human Factors in Computing Systems*, page 10, 2010.
64. Maayan Roth, Tzvika Barenholz, Assaf Ben-David, David Deutscher, Guy Flysher, Avinatan Hassidim, Ilan Horn, Ari Leichtberg, Naty Leiser, Yossi Matias, and Ron Merom. Suggesting (More) Friends Using the Implicit Social Graph. In *International Conference on Machine Learning*, 2011.
65. G. Salton, A. Wong, and C. S. Yang. A Vector Space Model for Automatic Indexing. *Communications of the ACM*, 18(11):613–620, November 1975.

66. Gerard Salton and Christopher Buckley. Term-Weighting Approaches in Automatic Text Retrieval. *Information Processing & Management*, 24(5):513–523, 1988.
67. R. Schapire. The boosting approach to machine learning: An overview. In *Nonlinear Estimation and Classification*, Berkeley, 2002. Springer.
68. Richard B Segal and Jeffrey O Kephart. MailCat : An Intelligent Assistant for Organizing E-Mail. In *Proceedings of the Third International Conference on Autonomous Agents*, 1999.
69. Xiance Si, Zhiyuan Liu, Peng Li, Qixia Jiang, and Maosong Sun. Content-based and Graph-based Tag Suggestion. In *ECML PKDD Discovery Challenge*, pages 243–260, 2009.
70. Randall Stross. Struggling to evade the e-mail tsunami. `http://www.nytimes.com/2008/04/20/technology/20digi.html`, April 2008.
71. Mark Suster. One man's signal is another man's noise. `http://www.bothsidesofthetable.com/2011/03/02/one-mans-signal-is-another-mans-noise/`, March 2011.
72. Tony Tam. Classificação Automática de Mensagens de Correio Electrónico Através de Técnicas de Text Mining e Ontologias. Technical report, Instituto Superior de Engenharia de Lisboa, 2008.
73. John C Tang, San Jose, Eric Wilcox, Julian A Cerruti, Hernan Badenes, Stefan Nusser, and Jerald Schoudt. Tag-it, Snag-it, or Bag-it: Combining Tags, Threads, and Folders in E-mail. In *ACM Conference on Human Factors in Computing Systems*, pages 2179–2194, Florence, 2008.
74. Lei Tang, Suju Rajan, and Vijay K Narayanan. Large Scale Multi-Label Classification via MetaLabeler. In *International conference on World wide web*, pages 211–220, 2009.
75. TaskTracer. Tasktracer. `http://tasktracer.osuosl.org/`, 2011.
76. Sergey Tulyakov, Stefan Jaeger, Venu Govindaraju, and David Doermann. Review of Classifier Combination Methods. In Hiromichi Fujisawa Simone Marinai, editor, *Studies in Computational Intelligence: Machine Learning in Document Analysis and Recognition*, volume 386, pages 361–386. Springer, 2008.
77. Twitter. About. `http://twitter.com/about`, June 2011.
78. C. J. van RIJSBERGEN. *Information Retrieval*, volume 13. Second edi edition, January 2005.
79. V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, 1999.
80. Suge Wang, Deyu Li, Xiaolei Song, Yingjie Wei, and Hongxia Li. A feature selection method based on improved fisher's discriminant ratio for text sentiment classification. *Expert Systems with Applications*, pages 1–7, February 2011.
81. Steve Whittaker, Victoria Bellotti, and Paul Moody. Revisiting and Reinventing Email. *Human-Computer Interaction*, 20(1):1–9, 2005.
82. Steve Whittaker and Candace Sidner. Email overload - exploring personal information management of email. In *ACM Conference on Human Factors in Computing Systems*, pages 276–283, 1996.
83. Fred Wilson. Email bankruptcy. `http://www.avc.com/a_vc/2010/05/email-bankruptcy.html`, May 2010.
84. Yiming Yang and Jan O. Pedersen. A Comparative Study on Feature Selection in Text Categorization. In *International Conference on Machine Learning*, volume 9714, pages 412–420, 1997.
85. Yiming Yang, Shinjae Yoo, Frank Lin, and Il-chul Moon. Personalized Email Prioritization Based on Content and Social Network Analysis. *IEEE Intelligent Systems*, 25(4):12–18, 2010.
86. Shinjae Yoo. *Machine Learning Methods for Personalized Email Prioritization*. PhD thesis, Carnegie Mellon University, 2010.