Artificial Intelligence Review (2020) 53:5019–5081
https://doi.org/10.1007/s10462-020-09814-9

Applicability of machine learning in spam and phishing email filtering: review and approaches

Tushaar Gangavarapu[1,2] · **C. D. Jaidhar**[1] · **Bhabesh Chanduka**[1]

Published online: 22 February 2020

Abstract

With the influx of technological advancements and the increased simplicity in communication, especially through emails, the upsurge in the volume of unsolicited bulk emails (UBEs) has become a severe threat to global security and economy. Spam emails not only waste users' time, but also consume a lot of network bandwidth, and may also include malware as executable files. Alternatively, phishing emails falsely claim users' personal information to facilitate identity theft and are comparatively more dangerous. Thus, there is an intrinsic need for the development of more robust and dependable UBE filters that facilitate automatic detection of such emails. There are several countermeasures to spam and phishing, including blacklisting and content-based filtering. However, in addition to content-based features, behavior-based features are well-suited in the detection of UBEs. Machine learning models are being extensively used by leading internet service providers like Yahoo, Gmail, and Outlook, to filter and classify UBEs successfully. There are far too many options to consider, owing to the need to facilitate UBE detection and the recent advances in this domain. In this paper, we aim at elucidating on the way of extracting email content and behavior-based features, what features are appropriate in the detection of UBEs, and the selection of the most discriminating feature set. Furthermore, to accurately handle the menace of UBEs, we facilitate an exhaustive comparative study using several state-of-the-art machine learning algorithms. Our proposed models resulted in an overall accuracy of 99% in the classification of UBEs. The text is accompanied by snippets of Python code, to enable the reader to implement the approaches elucidated in this paper.

Keywords Feature engineering · Machine learning · Phishing · Python · Spam

✉ Tushaar Gangavarapu
tusgan@amazon.com

[1] Department of Information Technology, National Institute of Technology Karnataka, Surathkal, Mangaluru 575025, India

[2] Automated Quality Assistance (AQuA) Machine Learning Research, Content Experience and Quality Algorithms, Amazon.com, Inc., Chennai, India

1 Introduction

Digital products and services increasingly mediate human activities. With the advent of email communication, unsolicited emails, in recent years, have become a serious threat to global security and economy (Bergholz et al. 2010). As a result of the ease of communication via emails, a vast number of issues involving the exploitation of technology to elicit personal and sensitive information have emerged. Identity theft, being one of the most profitable crimes, is often employed by felons to lure unsuspecting online users into revealing confidential information such as social security numbers, account numbers, and passwords. Unsolicited emails disguised as coming from legitimate and reputable sources often attract innocent users to fraudulent sites and persuade them to disclose their sensitive information. As per the report by Kaspersky Lab, in the first quarter of 2019, the menace of such unwanted emails was responsible for 55.97% of traffic (0.07% more than that in the fourth quarter of 2018). Unsolicited Bulk Emails (UBEs) can be broadly categorized into two distinct yet related categories: spam and phishing.

Spam emails are essentially UBEs that are sent without users' consent, primarily for marketing purposes such as selling unlicensed medicines, illegal products, and pornography (Toolan and Carthy 2010). The growth of spam traffic is a worrisome issue as such emails consume a lot of network bandwidth, waste memory and time, and cause financial loss. Phishing emails, on the other hand, are a much more serious threat that involves stealing individuals' confidential information such as bank details, social security numbers, and passwords. Most of the phishing attacks are focused towards financial institutions (e.g., banks); however, attacks against government institutions, although not as targeted, cannot be overlooked (Bergholz et al. 2010). To understand the impact of phishing, consider *pharming*, a variant of phishing, where the attackers misdirect users to fraudulent sites through domain name server hijacking (Abu-Nimeh et al. 2007). The effect of spam and phishing on valid users is multi-fold:

- Generally, UBEs promote products and services with little real value, pornography, get-rich-quick schemes, unlicensed medicines, dicey legal services, and potentially illegal offers and products.
- UBEs often hijack real users' identities to send spam to other users (e.g., business email compromise scams such as email spoofing and domain spoofing [$\approx$ amounted to almost \$1.3 billion in 2018 (20,373 victims), which was twice as much as that in 2017 (15,690 victims) (Bec scams trends and themes 2019)].
- Phishing, in particular, involves identity theft as financial identity theft, criminal identity theft, identity cloning, or business/commercial identity threat.
- Mailing efficiency and recipient's productivity are drastically affected by UBEs.

A study by the McKinsey Global Institute revealed that an average person spends 28% of the workweek ($\approx$ 650 h a year) reading and responding to emails (Gang 2017). Additionally, research on SaneBox's internal data revealed that only 38% of the emails on an average are relevant and important (Gang 2017), equivalent to $\approx$ 11% of the workweek. Furthermore, a study by the Danwood Group found that it takes an average of 64 seconds to recover from an email interruption and return to work at the rate before the interruption (Gang 2017)—adversely affecting the recipients' productivity, especially in the case of irrelevant UBEs. Based on the Kaspersky Lab report, in 2015, the UBE email volume fell by 50% for the first time since 2003 ($\approx$ three to six million). Such decline was attributed to the reduction (in billions) of major botnets responsible for spam and phishing. Conversely, by the end of

2015, the UBE volume escalated. Furthermore, Kaspersky spam report revealed an increase in the presence of pernicious email attachments (e.g., malicious macros, malware, ransomware, and JavaScript) in the spam email messages. By the end of March 2016, the UBE volume (≈ 22,890,956) had quadrupled in comparison with that witnessed in 2015. In 2017, the Internet Security Threat Report (ISTR) (Symantec 2018) estimated that the volume of spam emails had skyrocketed to an average of 55% (≈ 2% more than that in 2015 (52.7%) and 2016 (53.4%)). Clearly, spam and phishing rates are rapidly proliferating. The overall phishing rate in 2017, according to the ISTR (Symantec 2018), is nearly one in every 2, 995, while the number of Uniform Resource Locators (URLs) related to phishing rose by 182.6%, which accounted for 5.8% (one in every 224) of all malicious URLs.

Over the years, extensive research in this domain revealed several plausible countermeasures to detect UBEs. Approaches such as secure email authentication result in high administrative overload and hence, are not commonly used. Machine learning and knowledge engineering are two commonly used approaches in filtering UBEs. In knowledge engineering, UBEs are classified using a set of predefined rules. However, knowledge engineering approaches require constant rule updation to account for the dynamic nature of the UBE attacks—often suffer from scalability issues. In machine learning approaches, the algorithm itself learns the classification rules based on a training set—determining the email type through the analysis of the email content and structure has emerged, owing to the success of AI-assisted approaches in UBE classification. This area of research is actively being developed to account for the dynamic nature of UBE attacks. Past works in the existing literature explore several informative features, and many machine learning algorithms have been developed and utilized to classify the incoming mail into junk and non-junk categories (Toolan and Carthy 2010; Chandrasekaran et al. 2006; Toolan and Carthy 2009; Mohammad et al. 2015; Fette et al. 2007; Shams and Mercer 2013). Many leading internet service providers including Yahoo mail and Gmail, employ a combination of machine learning algorithms such as neural networks, to handle the threat posed by UBE emails effectively. Since machine learning models have the capacity to adapt to varying conditions, they not only filter the junk emails using predefined rules but also generate new rules to adapt to the dynamic nature of the UBE attack. Despite the success, adaptability, and predictability of machine learning models, preprocessing, including feature extraction and selection plays a critical role in the efficacy of the underlying UBE classification system (Turner et al. 1999; Michalski et al. 2013). Thus, there is a need to determine the most discriminative and informative feature subset that facilitates the classification of UBEs with a higher degree of confidence.

Due to the vast heterogeneity in the existing literature, there is no consensus on which features form the most informative and discriminative feature set. Moreover, to the best of our knowledge, only a few works have evaluated all the possible set of features and provided insights on the *importance of a feature* concerning the classification of UBEs.[1] In this paper, we aim at providing an accessible tutorial to security analysts and scientists seeking to avail benefits from the existing email data. First, we elucidate on the way of extracting vital and informative features (after extensive experimentation, we resorted to the features devised in the seminal work by Toolan and Carthy (2010), to achieve high performance in real-time)

[1] We experimented with advanced content-based features and topics extracted using Doc2Vec and hierarchical Dirichlet process. However, Doc2Vec style textual features and Dirichlet topics did not enhance in the predictability of the underlying machine learning models, owing to the similar content writing style of ham and UBE emails. The discriminative features in the email body-content, including the presence of phrases like 'verify your account,' have been considered in this study.

from the email corpus. Then, we present six prolific and widely used feature selection (extraction) methods including Variance-based filtering (LowVar), Correlation-based filtering (HighCorr), Feature Importance based filtering (FI), Minimum Redundancy Maximum Relevance (mRMR), and principal component analysis (PCA),[2] to determine an optimal feature subspace that facilitates effective learnability and generalizability of the underlying machine learning models, thus impacting the predictability of UBEs. Finally, we evaluate the obtained optimal feature subspace using eight state-of-the-art machine learning algorithms including Naïve Bayes (NB), Support Vector Machines (SVM), Bagged Decision Trees (BDT), Random Forest (RF), Extra Trees (ET), AdaBoost (AB), Stochastic Gradient Boosting (SGB), and Voting Ensemble (VE). The key contributions of this paper are mainly four-fold:

- We discussed the extraction of critical and potential email features with discriminative capabilities concerning UBEs, through the analysis of both email body-content and structure.
- We leveraged several prolific feature selection (extraction) approaches to engender an optimal informative feature subspace that enables effective and accurate UBE detection and classification.
- We present an extensive comparative study to elucidate on the applicability, learnability, and generalizability of several state-of-the-art machine learning models in facilitating UBE filtering and classification.
- To enhance the understanding of the readers, we exposed them to several feature selection and machine learning algorithms through snippets of Python code, enabling them to avail benefits from the existing email data.

The rest of the paper is organized as follows: Section 2 presents an overview of the existing works, and reviews their advantages and limitations, while Sect. 3 presents the background discussion. Section 4 elucidates on the steps employed in the process of feature extraction from emails, feature selection from the extracted email data, and understanding the *importance of a feature* with respect to UBEs. The machine learning algorithms employed in the UBE classification are presented in Sect. 5. In Sect. 6, we evaluate the obtained feature subspaces using several machine learning algorithms. Finally, Sect. 7 summarizes this paper with future enhancements.

2 Related work

Utilizing AI-assisted approaches for UBE detection and classification has become a prominent area of global research interest. This section aims at reviewing some of such existing techniques which were utilized in the development and evaluation of a potential set of features in the classification of spam and phishing emails, and to provide an overview of the existing modeling strategies.

Lueg (2005) presented a brief survey exploring the way of applying information retrieval and information filtering mechanisms to postulate spam filtering in a theoretically grounded and logical way. Although the author aimed at introducing an operationally efficient spam detector, the presented survey did not detail the simulation tools, machine learning approaches, or the datasets utilized. Wang et al. (2005) reviewed several approaches of

[2] Note that PCA facilitates feature extraction (through a linear transformation) rather than feature selection.

detecting spam emails, categorized unsolicited spam emails into hierarchical folders, and facilitated automatic regulation of the tasks concerning the response to an email. However, the author did not cover any machine learning approaches. Chandrasekaran et al. (2006) published a seminal work in the UBE detection and classification, and their work introduced and employed structural email features such as the content richness and the number of functional words (e.g., bank, credit, and credit) to discriminate phishing emails from legitimate ones. They used an SVM classifier to detect phishing emails and prevent them from reaching the user's inbox, thus reducing any possible human exposure. The work by Zhong et al. (2006) chronicled an innovative spam filtering approach that ensembled several filters. Abu-Nimeh et al. (2007) compared the accuracies of classifying 2, 889 emails using supervised machine learning (SML) models including SVM and RF using 43 potential features. The authors showed that RF classifier outperformed several other classifiers (low error rate). Despite the novelty and inventiveness in these works (Chandrasekaran et al. 2006; Zhong et al. 2006; Abu-Nimeh et al. 2007), they did not benchmark their approach against the recent works.

Cormack (2008) explored the relationship between email spam detectors and spam detectors in storage media and communication, with emphasis on the efficiency of the proposed methods. Furthermore, the characterization of email spams (e.g., users' information requirements) was scrutinized by the author. However, the work lacked detailing of certain vital components of spam filters. Sanz et al. (2008) detailed the issues concerning UBE research, the effects of such issues on the users, and the ways of reducing such effects. Their research work elucidated on several machine learning algorithms utilized in UBE detection. However, their work lacked a comparative analysis of various content filters. Ma et al. (2009) used a set of orthographic features to achieve an automatic clustering of phishing emails, which resulted in greater efficiency and better performance via Information Gain (IG) with $C4.5$ Decision Tree (DT). They used the modified global K-means approach to generate the objective function values (over a range of tolerance values), for selected feature subsets, which assisted in recognition of clusters. Toolan and Carthy (2009) used a recall-boosting ensemble approach which was based on $C5.0$ DT, and instance-based learning ensemble techniques to reclassify emails that were classified as non-phishing by $C5.0$ DT. They obtained a good precision through the use of $C5.0$ DT and 100% recall from the ensemble. Gansterer and Pölz (2009) proposed a system of filtering the incoming emails into ham, spam, and phishing, based on Feature Selection by Category (FSC), which provided better (97%) classification accuracy (ternary classification) than that resulted from the use of two binary classifiers.

Basnet and Sung (2010) proposed a method of detecting phishing emails through the use of confidence-weighted linear classifiers. The authors only utilized the email contents as features and neglected the use of any heuristic-based phishing specific features. A prominent work in the field of phishing email filtering was presented by Bergholz et al. (2010), where the authors described several novel features including statistical models for email topic descriptions, email text and external link analysis, and the analysis of embedded logos concerning hidden salting. Dhanaraj and Karthikeyani (2013) studied and developed approaches premeditated to mitigate email image spam. Despite the creativeness in designing image-based methods, their work did not elucidate on the machine learning models or the utilized corpus. Zhang et al. (2014) developed an automatic detection approach specific to Chinese e-business websites by using the URL and website-content specific features. The authors employed four machine learning classifiers including RF, Sequential Minimum Optimization (SMO), logistic regression, and Naïve Bays (NB), and evaluated their results using Chi-squared statistics (χ^2). Laorden et al. (2014) explained the importance of anomaly discovery in UBE filtering in reducing the requirement of classifying UBEs. Their work reviews an anomaly-based UBE sieving approach which utilized a data minimization approach that reduced preprocessing while maintaining the information about email message appropriateness concerning the email nature. Table 1 reviews other related and significant past works in the detection of spam and phishing emails.

More recently, many works aimed at studying the applicability of different machine learning approaches including K-Nearest Neighbors (KNN), SVM, NB, neural networks, and others, to spam and phishing email filtering, owing to the ability of such approaches to learn, adapt, and generalize. In 2016, a broad overview of some of the state-of-the-art content-based UBE filtering approaches was presented by Bhowmick and Hazarika (2016). Their work surveyed several vital concepts in UBE filtering, the effectiveness of the current efforts, and recent trends in UBE classification, while focusing on popular machine learning approaches for the detection of the nature of an email. Moreover, they discussed the changing nature of UBE attacks and examined several machine learning algorithms to combat the menace of such emails. Sah and Parmar (2017) proposed a model to effectively detect the malicious spam in emails through effective feature selection, followed by classification using three machine learning approaches including NB, SVM, and Multi Layer Perceptron (MLP). With the promising success of deep neural architectures in various applications (Gangavarapu et al. 2019c; Jayasimha et al. 2020), some of the recent works have employed deep learning models to classify UBEs. Apruzzese et al. (2018) evaluated the applicability, effectiveness, and current maturity of deep and machine learning models in the detection of malware, intrusion, and spam. The authors concluded that utilizing different machine learning classifiers to detect specific tasks can increase the UBE detection performance; however, they drew no significant conclusions concerning deep neural models. Hassanpour et al. (2018) modeled the email content as Word2Vec style features and classified them using several deep learning classification approaches—the authors achieved an overall accuracy of 96%. Vorobeychik and Kantarcioglu (2018) used adversarial machine learning to generate email samples and trained the classifier to distinguish those generated samples, making the learning model robust to adversarial manipulation and decision-time attacks. The authors concluded with a note on several issues concerning adversarial modeling that warrant further research. More prominent and impactful research works in the domain of UBE detection and filtering are tabulated in Table 2.

Some of the works presented in Table 2 employed feature-free approaches to facilitate spam and phishing detection. However, such approaches suffer from high computational complexity and cost of training. Some research works considered email header, subject line, and body as the most prominent features in classifying UBEs. However, it is worth noting that, suspicious email header, subject line, and body could be misleading, and behavior-based email features could be essential to facilitate accurate classification of UBEs. Most of the researchers focused on the classification performance in terms of classification accuracy. This work differs from the efforts of previous works by revisiting various state-of-the-art machine learning approaches for UBE classification. We employ feature selection to kindle an optimal feature subspace to lower the computational complexity and enhance the classification performance. Additionally, we present several key performance indicators other than classification accuracy to assess the performance of the underlying models accurately. Furthermore, we present an accessible tutorial to security specialists through snippets of Python code that is intended on exposing them to the presented feature selection and machine learning algorithms.

3 Background

Certain email features (e.g., keywords such as *debit*, *verify*, and *account*) are more prominent in UBEs than in ham emails, and by measuring the rate of occurrence of such features, we can ascertain the probabilities for those email characteristics which in turn aids

Table 1 Summary of some key past works that employed machine learning to facilitate UBE classification

Work	Approach(es)	Classifier(s)	Feature selection	Highlight(s)	Remark(s)
Pan and Ding (2006)	Phishing detector that examines the inconsistency in a website's identity, its HTTP transactions, and DOM[a] properties	SML	χ^2	Phishing-independent anti-phishing scheme with a low miss rate	Use of heterogeneous features; high computation time and cost
Toolan and Carthy (2010)	Set of 40 potential features from a corpus of over 10, 000 emails were generated to detect UBEs	*C*5.0 DT	IG	A detailed evaluation of the possible features	Only employs IG to evaluate the importance of the features
Khonji et al. (2011)	Enhancing the classification accuracy using an effective feature subset based on all the previously proposed features	RF	CFS[b], WFS[c], IG	Evaluates various feature selection techniques	Relies on only a limited number of classifiers
Zhuang et al. (2012)	Detection model with several phases: feature extraction, training, ensemble classification, and cluster training	SML	Maximum relevance	Better performance in comparison to commonly used tools and methods	Complex computations involving redundant features
Hamid and Abawajy (2014)	Ensemble multi-tier detector that uses clustering to weigh features and profile the best (high weighted) features	SMO, AB	IG	More efficient than the modified global *K*-means approach	Irrelevancy, redundancy, and scalability issues; high computation time
Hassan (2015)	Embedded feature selection algorithm to analyze the features and mitigate redundant and irrelevant ones	SML, SMO, DT, NB	CFS[b], FSC, WFS[c], FFS[d]	Higher accuracy, and low FPR and FNR with DT classifier	Greedy approach and might not always work

[a] Document object model

[b] Correlation-based feature selection

[c] Wrapper-based feature selection

[d] Filter-based feature selection

Table 2 Summary of some key existing works in the field of UBE detection and filtering

Work	Approach(es)	Compared algorithm(s)	Remark(s)	Dataset(s)	Evaluation metric(s)
Zhao and Zhang (2005)	Rough set classifier to incorporate fuzziness and uncertainty	NB and rough set	Low performance	Spambase	Accuracy, precision, and recall
Norte Sosa (2010)	Forward search feature selection with MLP as the classifier and five-fold double classification	–	No comparison of performance	Collected emails (2, 200)	Accuracy
Mousavi and Ayremlou (2011)	Classification using NB	–	No comparison of performance	Collected emails	Precision and recall
Awad and ELseuofi (2011)	Classification using NB, KNN, MLP, SVM, rough sets, and artificial immune system	NB, KNN, MLP, SVM, rough set, and artificial immune system	State-of-the-art UBE classification approaches were neglected	SpamAssassin	Accuracy, precision, and recall
Choudhary and Dhaka (2013)	Automatic classification using the genetic algorithm	–	No comparison of performance	Words in a data dictionary	–
Shrivastava and Bindu (2013)	Classification using the genetic algorithm with a heuristic fitness function	–	No comparison of performance	Collected emails (2, 248)	Accuracy
Bhagyashri et al. (2013)	Classification using NB	–	No comparison of performance	SpamAssassin	Accuracy, precision, and recall
Akinyelu and Adewumi (2014)	RF	Compared with Fette et al. (2007)	Inadequate evaluation metrics to estimate the efficacy of the proposed approach	Collected emails (2, 000)	False positives and negatives
Idris and Abdulhamid (2014)	Classification using artificial immune system	–	Lack of standard evaluation metrics for performance evaluation	Datasets from UCI repositories	False positive rate
Sharma et al. (2014)	Classification using MLP	Low performance	NB and MLP	TREC 07	Accuracy, precision, and recall

Table 2 (continued)

Work	Approach(es)	Compared algorithm(s)	Remark(s)	Dataset(s)	Evaluation metric(s)
Dhanaraj and Palaniswami (2014)	Classification using firefly and NB	NB, firefly, particle swarm optimization, and neural networks	Low performance	CSDMC 2010	Accuracy, sensitivity, and specificity
Kumar and Arumugam (2015)	Particle swarm optimization for feature selection and probabilistic neural network for classification	NB, probabilistic neural network, and BLAST	Low performance	Collected emails	Sensitivity and specificity
Renuka et al. (2015)	Classification using genetic algorithm with NB and ant colony optimization with NB	Genetic algorithm with NB and ant colony optimization with NB	No performance improvement	Spambase	Accuracy, precision, recall, and F-measure
Karthika and Visalakshi (2015)	Classification using the hybrid of ant colony optimization and SVM	NB, KNN, and SVM	Very low performance	Spambase	Accuracy, precision, and recall
Awad and Foqaha (2016)	Classification using the hybrid of particle swarm optimization and radial basis function neural networks	Particle swarm optimization, MLP, neural networks, and radial basis function neural networks	High model build time	Spambase	Accuracy
Sharma and Suryawanshi (2016)	KNN classification with Spearman's rank-order correlation	KNN classification with Spearman's rank-order correlation and KNN classification with Euclidean distance	Low performance	Spambase	Accuracy, precision, recall, and F-measure
Alkaht and Al-Khatib (2016)	Classification using multi-stage neural networks	MLP and neural networks	High training time	Collected emails	Accuracy
Palanisamy et al. (2016)	Classification using negative selection and particle swarm optimization	NB, SVM, particle swarm optimization, and negative selection algorithm	High training time	Ling	Accuracy

Table 2 (continued)

Work	Approach(es)	Compared algorithm(s)	Remark(s)	Dataset(s)	Evaluation metric(s)
Zavvar et al. (2016)	Classification using SVM, neural networks, and particle swarm optimization	KNN, SVM, particle swarm optimization, and self organizing map	No comparison of performance	Spambase	AUROC
Tyagi (2016)	Classification using deep neural networks	Dense MLP, deep belief networks, and stacked denoising autoencoder	High training time	Enron, PU1, PU2, PU3, and PUA	Accuracy, precision, recall, and F-measure
Rajamohana et al. (2017)	Classification using adaptive binary flower pollination algorithm	Binary particle swarm optimization, shuffled frog leaping algorithm, and adaptive binary flower pollination algorithm for feature selection, and NB and KNN for classification	Lack of standard evaluation metrics for performance evaluation	Dataset in Ott et al. (2011)	Global best positions

in the determination of the email type. The existing literature presents a wide variety of techniques to determine and utilize such discriminative features, and in this section, we describe the different categories of UBE filtering approaches widely used to overcome the menace of such emails. We also elucidate on the UBE filters widely used by popular internet service providers to curtail the dangers posed by email-borne malware, phishing, and malware in UBEs.

3.1 Categorization of the existing UBE filtering techniques

Over the years, academicians and researchers have proposed various UBE detection and filtering approaches which have been utilized successfully to classify email data into groups. These approaches can be broadly categorized into: content-based and behavior-based filters, sample base or case base filters, rule-based or heuristic filters, previous likeness based filters, and adaptive filters.

3.1.1 Content-based and behavior-based filters

Content-based and behavior-based UBE filtering approaches aim at analyzing the email content and structure to create automatic classification rules using machine and deep learning approaches such as KNN, NB, MLP, and neural networks. Content-based and behavior-based filters analyze the tokens (words), their distribution, their occurrences and co-occurrences, in addition to the analysis of scripts and URLs, in the context of emails, and then utilize the learned knowledge to generate rules to facilitate automatic filtering of incoming UBE emails (Christina et al. 2010).

3.1.2 Sample base or case base filters

Sample base or case base filtering techniques are popular in spam and phishing email filtering. Through an email collection model, all the emails, including ham, spam, and phishing, are extracted from every user's email. Then, preprocessing of the raw email data into a machine-processable form is facilitate through feature selection (extraction) and grouping the email data. Finally, the preprocessed data is mapped into distinct UBE categories, and a machine learning algorithm is employed to train the existing email data. The trained models are then tested on the incoming emails to categorize them into ham, spam, or phishing (Christina et al. 2010).

3.1.3 Rule-based or heuristic filters

Rule-based or heuristic UBE filtering approaches [e.g., SpamAssassin (Mendez et al. 2006)] utilize the existing heuristics or rules to assess several patters (specifically, regular expressions) against an incoming email message—the score of an incoming email is reliant on the number of patterns in the email message (when the patterns in the email message do not correspond to the preset regular expressions, the score is reduced). The UBE emails are then filtered using a specific predetermined threshold. While certain heuristics do not change over time, other heuristics require constant updating to cope with the changing and dynamic nature of the UBE emails (Christina et al. 2010).

3.1.4 Previous likeness based filters

Previous likeness based UBE filtering approaches utilize instance-based or memory-based machine learning approaches to classify the incoming email messages based on their likeness and resemblance to the stored training sample emails. A multi-dimensional vector is created using the attributes of the sample emails, which is then used to plot new instances. A new instance is mapped to a target class using the most common class among the K-nearest neighbors of the point (Sakkis et al. 2001). Finally, the KNN classifier is employed to classify the incoming email messages.

3.1.5 Adaptive filters

Adaptive UBE filtering approaches facilitate the detection and classification of UBEs by categorizing emails to distinct groups. In this approach, the email corpus is segregated into several groups, and each group poses an emblematic text. The similarity between an incoming email and a particular group determines the email message score with respect to that particular group. The scores computed across all the groups are utilized in deciding the most probable group concerning the incoming email message (Pelletier et al. 2004).

3.2 UBE filters: how yahoo mail and gmail filter UBEs

Leading internet service providers including Yahoo mail and Gmail have employed several machine learning approaches such as neural networks, to handle the threat posed by UBEs effectively. Recent research revealed that the machine learning model employed by Google facilitates the detection of UBEs with 99.9% classification accuracy—one in a thousand email messages succeeds in evading the UBE filter in Gmail. To account for the considerable UBE volume (≈ 50–70% of the emails), the UBE detection models developed by Google incorporate Google safe browsing tools to identify websites with malicious URLs. The performance of UBE filtering is enhanced further through additional, comprehensive scrutiny of phishing emails. Such more in-depth examination causes additional delay; however, only 0.05% of the emails are subject to such delay. Further details on the email UBE filters employed by popular internet service providers are presented in the following subsections.

3.2.1 Yahoo mail UBE filtering

Yahoo mail is one of the first free webmail service providers with more than 320 million users. Yahoo mail utilizes several algorithms and a combination of methods rooted in basic techniques, including spam and email content users' complaints and URL filtering. The email provider employs email filtering by domains rather than by IP addresses. Furthermore, Yahoo mail provides ways of preventing a valid internet user for being mistaken for a cybercriminal (e.g., ability to troubleshoot SMTP errors using SMTP logs). The complaint feedback loop service helps users maintain trust in the services and UBE filtering approaches employed by Yahoo mail. Moreover, the email service provider also facilitates Yahoo whitelisting (return path certification and internal whitelisting)—whitelisting rolls back to the user to specify the list of senders to receive email messages from (placed in a list of trusted users), unlike in blacklisting. The service user can employ a combination of Yahoo's spam-fighting techniques along with whitelisting to reduce the volume of legitimate emails being erroneously classified as unsolicited emails. Whitelisting alone can result in a strict implication on unapproved senders, in which case, Yahoo mail utilizes an automatic whitelisting procedure, where the anonymous sender's address is checked against a database for any history of spamming or phishing—if the unapproved user has no record of cyber attacking, the email message is sent to the recipient, and the user's email is added to the whitelist.

3.2.2 Gmail UBE filtering

Google mail employs hundreds of rules to determine the nature of an incoming email—each rule depicts a specific feature or aspect of a UBE with some statistical value which is reliant on the likelihood that a particular feature corresponds to UBEs. The weighted importance of the features is utilized to determine the final score for an incoming email message. The score is measured against a sensitivity threshold determined using each user's UBE filter, and consequently, an incoming email is classified as ham or unsolicited. Unlike Yahoo mail, Gmail filters email messages by IP addresses rather than by domains. To facilitate accurate classification of UBEs, Gmail utilizes state-of-the-art machine learning algorithms including neural networks and logistic regression. Additionally, to shield Gmail users from any possible image UBEs, Google utilizes optical character recognition. Furthermore, the UBE filtering by Gmail is greatly enhanced by linking several features through the use of machine learning algorithms utilized in combining and ranking large sets of Google search results. Factors like links in the email message headers and domain reputation depict the evolving and dynamic nature of the UBEs over time—due to these factors, legitimate emails could be classified as UBEs. With the emergence of state-of-the-art algorithms, tools, users' feedback, and new UBE discovery, the filtering settings are updated continuously.

4 Methods: feature extraction and selection

In this section, we focus on describing the way of processing the raw email data[3] based on forty discriminative features devised by Toolan and Carthy (2010), to facilitate the detection of spam and phishing emails. Moreover, we elucidate on determining the *importance of a feature* concerning the features of UBEs. The following subsections give tactful insights on the entire procedure employed as a part of feature engineering, which deals with the process of transforming raw email data into informative and discriminative features that better represents the underlying email corpus. Such representations aid the classification models to learn, adapt, and generalize, which is essential in the accurate classification of unseen email instances. The entire workflow of the procedure employed to draw informative inferences from the raw email data is depicted in Fig. 1. The text is accompanied by snippets of Python code to familiarize the readers with the methods utilized in this study. The code is aimed at readers with Python familiarity, more resources concerning the same can be found at https://www.python.org/about/gettingstarted/.

[3] The email data utilized in this research can be found at https://goo.gl/gkuJ2g.

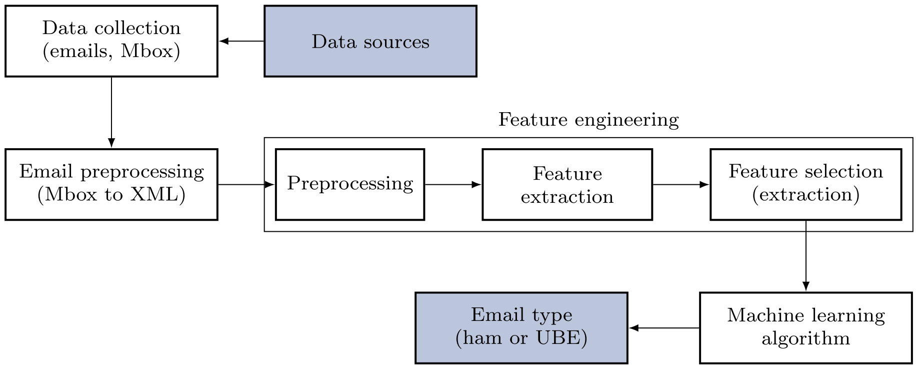
Data collection (emails, Mbox)
Data sources
Feature engineering
Email preprocessing (Mbox to XML)
Preprocessing
Feature extraction
Feature selection (extraction)
Email type (ham or UBE)
Machine learning algorithm

Fig. 1 An overview of the procedure employed to draw inferences from the collected data

Table 3 Summary of the email corpora utilized in this study

Dataset	Rate of ham	Rate of UBE (%)	Year of creation	References
SpamAssassin	83.6%	16.4	2002	Apache SpamAssassin (Mendez et al. 2006)
Phishing corpus	–	100	2015–2016	Nazario (2018)

4.1 Materials: raw email corpus

Most of the existing publicly available datasets including spam archive (Almeida and Yamakami 2010; Biggio et al. 2011), phishing corpus (Abu-Nimeh et al. 2007), and Princeton spam image benchmark (Wang et al. 2007) are lopsided towards UBE detection—the volume of UBEs utilized in evaluating the filter is much greater than that of ham emails, resulting in the machine learner recording a higher accuracy by concentrating solely on detecting UBEs, which might not scale well with the real-world data. Hence, a more suitable dataset is the one with near equal volumes of ham and non-ham emails, thus facilitating the underlying machine learner to learn and discriminate between ham emails and UBEs. The raw email data used in this paper consists of around 3844 emails in total, which is comprised of 2, 551 ham emails ($\approx$ 66.4%), 793 phishing emails (303 from 2015 and 490 from 2016, contributing to $\approx$ 20.6%), and 500 spam emails ($\approx$ 13%). These emails were collected from a variety of sources[4]—the spam and ham emails were collected from the SpamAssassin project (2002) (Mendez et al. 2006), while Nazario (2018) provided the phishing emails (see Table 3). We mine these emails to extract the information needed to facilitate the accurate classification of those emails into ham, spam, and phishing emails. To clarify the methods and techniques presented in this study and present all the intermediate results, we use the *test email* presented in Block 1. Note that the *test email* is constructed in a way that includes most characteristics of a UBE—such a choice can help mitigate the sampling problem while presenting intermediate results.

[4] Note that the individual corpus possesses highly distinctive qualities that are indicated through the experiments conducted on that specific corpus.

Block 1 An example *test email* that contains most of the spam and phishing email features.

```
From tushaar@nitk.edu.in  Fri Sep 22 11:04:35 2017
Return-Path : <tushaar@nitk.edu.in>
Delivered-To: test@localhost.examples.com
Received    : from localhost [127.0.0.1]
              by localhost with POP3 (fetchmail-5.8.8)
              for test@localhost (single-drop);
              Fri, 22 Sep 2017 11:07:38 +0200 (EDT)
Received    : from emztd2202.com ([68.85.145.178])
              by webnotes.net (7.8.4/7.8.4) with SMTP id KAA08354
              for <test@examples.com>;
              Fri, 22 Sep 2017 10:14:09 +0200
Message-Id  : <200206230815.KAA08354@webnotes.net>
From        : "Tushaar Gangavarapu" <tushaar@nitk.edu.in>
Reply-To    : 15it117.tushaar@nitk.edu.in
To          : test@examples.com
Date        : Fri, 22 Sep 2017 10:12:41 -0800
Subject     : Re: Example of .eml format
X-Mailer    : Microsoft Outlook Express 5.01.2818.6800 DM
MIME-Version: 1.0
Content-Type: text/html; charset="us-ascii"
X-MIME-Auto
converted   : from quoted-printable to 8bit by webnote.net
              id KAA08354
Content-
Transfer-
Encoding    : 8bit

<a href="http://researchPhishing.net/info352"> <img border="0" alt="
    SBI" src="sbi.png"> </a>
<html>
  <body> <p> This email is from State Bank of India (SBI) </p>
  </body>
  <form>
    Enter your card number: <input type="text"> </input> <br/>
    Enter your pin: <input type="text"> </input>
  </form>
</html>
We as a bank access social services and help risk management.
These links help you learn more on risks associated
View: https://10.10.54.4:80/nation/education <br/>
Visit: http://researchIAS.net/it352 <br/>
Read: http://192.32.19.1:8000/blog <br/>
Risk: http://nitk@georgia.com/los_angeles <br/>
<a href="internal_link_01.php"> Click here to view terms </a>
<a href="internal_link_02.asp"> Click here to view policies </a>
<a href="https://hack.com"> Platinum cards on limited offer </a>
<html>
  <head>
    <script> window.status = "SBI passwords" ; </script>
    <script type="text/javascript">
      function popup() {
        window.alert("Enter account number!") ;
        window.open("http://www.hackPasswds.com/hack/email") ;
      }
      function verifyFunc() {
        window.open("http://www.hackPasswds.com/hack/login") ;
      };
    </script>
    <script src="myscripts.js"> Hey there </script>
  </head>
  <body>
    <p> Finally, login and verify your account <p>
    <a href="http://www.hackPasswds.com/hack/login"> Help with login
        </a>
    <button onclick="verifyFunc()"> Verify your account </button>
  </body>
</html>
```

From the *test email* in Block 1 it can be observed that an email contains additional 'metadata,' including reply-to address, from address, to address, and others (lines 1 to 26), that can be explored to aid in the classification of the email into ham, spam, or phishing. The following subsection presents a detailed discussion on the features of a given email [derived from Toolan and Carthy (2010)] that are prominent in the prediction of the nature of an email.

4.2 Preprocessing and feature extraction: obtaining informative feature space

In this section, we discuss the features employed in this study to transform raw email data into a machine-processable form. These features are internal to the emails and are not derived from external sources such as search engine information, spam assassin score, or domain registry information. Such external features were neglected, owing to the fact that such information might not be present always, and hence cannot be a part of a truly automated UBE filtering system. Moreover, research has shown that features internal to emails form a comparatively more informative feature set as most of the external data, including search engine results or domain name service information changes regularly.

As stated earlier, we carried out several experiments on the obtained email corpus to determine a suitable feature space that best represents the underlying corpus. These experiments included the utilization of advanced content-based features and topics extracted using paragraph vector network (vector size of 200) and hierarchical Dirichlet process (150 topics); however, the addition of such sophisticated features did not enhance the classification performance, and instead increased the computational complexity of training. Additionally, we employed the genetic algorithm (population size of 50, crossover rate of 0.6, and mutation rate of 0.1 for 25 iterations) to facilitate feature selection among the advanced content-based features and topics—this resulted in the proliferation of the training time with no significant improvement in the performance. The final feature space used in this study employed forty informative features with the capabilities of spam and phishing email discrimination, and they can be roughly divided into five distinct categories:

- Body-based features: that features that are extracted from the email message content.
- Subject line based features: the features that are extracted from the subject line of the email.
- Sender address based features: the features that are extracted from the information about the email address of the sender.
- URL-based features: the features that are extracted from the anchor tags of HTML emails.
- Script-based features: the features that are extracted from the information concerning the presence or absence of scripts in the email and the impact of such scripts.

The feature space composed of forty features is tabulated in Table 4. These features include nine body-based, eight subject line based, four sender address based, 13 URL-based, and six script-based features.

Note the presence of features like body_numFunctionWords, body_suspension, body_verifyYourAccount, subject_verify, subject_debit, and subject_bank—these features require exact word-to-word match, and their values could be easily miscalculated through deliberate spelling errors, unattended typographical errors (e.g., 'bank' and 'bnak'), or the

Table 4 The forty features utilized in the transformation of raw email data for the determination of the nature of an email

Feature category	Feature	Feature type	Summary
Body	html	Binary	Presence or absence of HTML tags in the body
	forms	Binary	Presence or absence of forms in the body
	numWords	Continuous	Total number of words in the body
	numCharacters	Continuous	Total number of characters in the body
	numDistinctWords	Continuous	Total number of distinct words in the body
	richness	Continuous	Ratio of numWords to numCharacters in the body
	numFunctionWords	Continuous	Total occurrence of keywords such as account, access, bank, click, credit, identity, information, inconvenience, limited, log, minutes, password, risk, recently, social, security, service, and suspended in the body
	suspension	Binary	Presence or absence of the word ‘suspension’ in the body
	verifyYourAccount	Binary	Presence or absence of the phrase ‘verify your account’
Subject line	reply	Binary	Checks if the email is a reply to a previous mail
	forward	Binary	Checks if the email is forwarded from another account
	numWords	Continuous	Total number of words in the subject line
	numCharacters	Continuous	Total number of characters in the subject line
	richness	Continuous	Ratio of numWords to numCharacters in the subject line
	verify	Binary	Presence or absence of the word ‘verify’ in the subject line
	debit	Binary	Presence or absence of the word ‘debit’ in the subject line
	bank	Binary	Presence or absence of the word ‘bank’ in the subject line
Sender address	numWords	Continuous	Total number of words in the sender address field
	numCharacters	Continuous	Total number of characters in the sender address field
	diffSenderReplyTo	Binary	Checks if the sender’s domain and reply-to domain are different
	nonModalSenderDomain	Binary	Checks if the sender’s domain and email’s modal are the same

Table 4 (continued)

Feature category	Feature	Feature type	Summary
URL	ipAddress	Binary	Checks for the use of IP address rather than a qualified domain
	numIpAddresses	Continuous	Number of links with IP addresses and not domain names
	atSymbol	Binary	Presence of links that contain an ‘@’ symbol.
	numLinks	Continuous	Total number of links in the email body
	numInternalLinks	Continuous	Total number of links in the body with internal targets
	numExternalLinks	Continuous	Total number of links in the body with external targets
	numImageLinks	Continuous	Total number of links in the body with an image
	numDomains	Continuous	Total number of domains from all the URLs in the body
	maxNumPeriods	Continuous	Highest number of periods from all the links
	linkText	Binary	Checks if the link text contains words like click, here, login, or update
	nonModalHereLinks	Binary	Checks for ‘here’ links mapping to a non-modal domain
	ports	Binary	Checks for URLs accessing the ports other than 80
	numPorts	Continuous	Number of links in the email with the port information
Script	scripts	Binary	Presence or absence of scripts in the body
	javaScript	Binary	Presence or absence of JavaScript in the body
	statusChange	Binary	Checks if any script overwrites the status bar of the email client
	popups	Binary	Presence or absence of any popup code in the body
	numOnClickEvents	Continuous	Total number of onClick events in the body
	nonModalJsLoads	Binary	Checks for any non-modal external JavaScript forms

usage of verb forms (e.g., ‘bank’ and ‘banking’). To cope with these shortcomings and obtain a standard canonical form from the raw email textual entries, we used the implementations in the Python `NLTK` library. The canonical form was obtained through tokenization, stemming, and lemmatization. In tokenization, we aimed at transforming the given text in the raw email entry into smaller words (tokens). Then, we facilitated suffix stripping using stemming, followed by lemmatization to convert the suffix stripped words to their base forms. Moreover, to handle spelling and typographical errors, we employed Jaro similarity scoring (Gangavarapu et al. 2019a, b) (through the implementations in the Python `textdistance` library) between the intended word spelling and the actual spelling. The Jaro similarity score is normalized (range of [0, 1]), and is given by,

$$\mathrm{Jaro}(t_i, t_j) = \begin{cases} 0, & \mathrm{m} = 0 \\ \frac{1}{3}\left(\frac{m}{|t_i|} + \frac{m}{|t_j|} + \frac{2m-T}{2m}\right), & \text{otherwise} \end{cases} \tag{1}$$

where t_i (of length $|t_i|$) and t_j (of length $|t_j|$) are the tokens under comparison with m matching characters and T transpositions. The threshold that determines if two tokens under comparison are the same was set to 0.9. The code in Block 2 details the entire preprocessing process utilized to obtain a canonical form. Thus, we mitigated the shortcomings arising due to spelling errors, typographical errors, and irregular verb forms.

Block 2 Code block to facilitate preprocessing of raw email textual entries to obtain a canonical form.

```
# Tokenization of a given email textual entry
tokens = mailTextEntry.split(' ')

# Obtaining the base form of a token by stemming and lemmatization
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()
stemmedToken = stemmer.stem(token)
lemmatizedToken = lemmatizer.lemmatize(stemmedToken)

# Finding the Jaro score between two tokens
jaro = textdistance.Jaro()
similarityScore = jaro(actualToken, obtainedToken)
```

4.2.1 Using python for feature extraction

Feature extraction aims at transforming raw email data into informative features that best represent the data without any loss of information. In our email corpus, we have 3, 844 emails (see Sect. 4.1). As explained in Sect. 4.2, we need to extract forty features (refer Table 4) from the collected raw email data. Before extracting the features, it is vital to parse the email to obtain the email body, subject line, sender address, reply-to address, modal URL, and all the links. We utilized the implementations in several Python libraries including `re`, `urlparse`, `BeautifulSoup`, `email`, `HTMLParser`, and `IPy`. Before proceeding any further, ensure that the encoding is set to UTF-8. The code in Block 3 elucidates on the way of extracting several parts (e.g., email body) from a raw email.

Block 3 Code block to extract the body, subject line, sender and reply-to address, modal URL, and all the links from a raw email.

```
# Extracting the email information from the raw data
mail = email.message_from_string(rawEmailAsString)

# Extracting the body of the email
bodyContent = mail.get_payload()

# Extracting the subject line of the email
decodeSubj = email.header.decode_header(mail[‘Subject’])[0]
subjLine = unicode(decodeSubj[0])

# Extracting the sender address from the email
decodeSend = email.header.decode_header(msg[‘From’])[0]
sendAddress = unicode(decodeSend[0])

# Extracting the reply-to address from the email
decodeReplyTo = email.header.decode_header(msg[‘Reply-To’])[0]
replyToAddress = unicode(decodeReplyTo[0])

# Extracting the modal URL from the email
URLs = re.findall(r"http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\(\)
    ,]|(?:%[0-9a-fA-F][0-9a-fA-F]))+", str(mail))
modalURL = max(set(URLs), key = URLs.count)

# Extracting all the links, both internal and external
soup = BeautifulSoup(msg, "lxml")
allAnchorLinks, anchorURLs = [], []
for link in soup.findAll(‘a’, attrs={‘href’: re.compile("^http[s
    ]?://")}):
    anchorUrls.append(link.get(‘href’))
for link in soup.findAll(‘a’):
    allAnchorLinks.append(link.get(‘href’))
nonAnchorURLs = difference(URLs, anchorURLs)*
allLinks = allAnchorLinks + nonAnchorURLs
```

* Diference (a, b) returns elements in a not in b (a–b).

The implementations in the Python email library provide extensive support to handle and parse email data and multipurpose internet mail extensions. First, we extracted the raw email data from the string format into the email format, which was then utilized to extract various parts of the email. To ensure the consistency in the encoding of UTF-8, we first decoded the required field and then encoded it in Unicode. The modal domain is the most frequently used domain in the email (Fette et al. 2007). Finally, to find all the links in the email, we needed to extract all the URLs linked in the form of href, as well as those present just as such in the email, i.e., both anchor links and non-anchor links comprising both internal and external email links. We used the implementations in the Python lxml library, which is a simple and powerful API to parse both XML and HTML. Now that we have extracted various parts of the email, we need to obtain the features from each part, as shown in Table 4.

Block 4 Code block to extract body-based features.

```
# Checking if the email body has HTML tags and forms
bodyHasHtml = bool(BeautifulSoup(bodyContent, "html.parser").find())
bodyHasForms = bool(BeautifulSoup(bodyContent, "html.parser").find("
    form"))
```

Since most of the body-based features such as body_numWords, body_richness, body_numCharacters, and others are easier to extract, we have only shown the process of extracting and checking for HTML tags and forms in the email (see Block 4). All the subject line based features are easily implementable through elementary Python programming modules.

Block 5 Code block to extract the domain for sender address based and URL-based features.

```
# Extracting the domain from the given email
domain = re.search("@[{\textbackslash}w.]+", emailAddress)
emailDomain = str(domain.group())[1:]

# Extracting the domain from the given URL
parsedURI = urlparse(URL)
domain = '{uri.netloc}'.format(uri=parsedURI)
URLDomain = domain[4:] if domain.startswith("www.") else domain
```

Utilizing the utility methods listed in Block 5, we can straightforwardly obtain sender address based features. Note that the sender address in the email is not merely the address, but is usually of the form: "Tushaar Gangavarapu" < *tushaar@nitk.edu.in* > (Toolan and Carthy 2010). URL based features are among the most important in the determination of the nature of the email, and most of the URL-based features are related to IP addresses. We use the implementations in the Python `IPy` package to facilitate the extraction of URL-based features (see Block 6).

Block 6 Code block to extract URL-based features.

```
# Checking if IP addresses are used instead of a quantified domain
for linkDomain in linksDomainList:
    if ":" in str(linkDomain):
        linkDomain = linkDomain[:linkDomain.index(":")]
    try:
        IP(linkDomain)
        urlIPAddress = True
        break
    except: continue

# Finding the count of the image links in the email
soup = BeautifulSoup(bodyContent)
numImgLinks = len(soup.findAll('img'))
```

Note that, the function `IP(.)` uses the dotted IP format without the port number; thus, if the port number is present in the IP address, it must be excluded before any further processing. Moreover, while obtaining the count of the domains in the email, we must include the domains of both the sender and the reply-to addresses. All the other URL-based features such as url_ports, url_numPorts, and others can be implemented effortlessly using the above-established methods. Finally, we show how to mine for script-based features from the email body in Block 7.

Block 7 Code block to extract script-based features.

```
# Checking for the presence of scripts in the given email
 hasScripts = bool(BeautifulSoup(bodyContent, "html.parser").find("script"))

# Checking for the scripts containing JavaScript
soup = BeautifulSoup(bodyContent)
for script in soup.findAll(‘script’):
    if script.get(‘type’) == "text/javascript": scriptJS = True

# Checking if a script overrides the status bar of the email client
for script in soup.findAll(‘script’):
    if "window.status" in str(script.contents): statChange = True

# Checking if an email contains a popup window code
for script in soup.findAll(‘script’):
    if "window.open" in str(script.contents): popups = True

# Finding the number of onClick events in the given email
numOnClickEvents = len(soup.findAll(‘button’,{"onclick":True}))
```

Using the above utility methods, we can easily verify if JavaScript comes from outside the modal domain. Table 5 shows the scores of all the forty features concerning the *test email* presented in Block 1. Now that we have obtained the feature space (forty informative features) from the given email, the subsequent step would be to measure the *importance of each feature*, to understand the contribution of each feature towards the determination of the nature of a given email.

4.3 Feature selection: engendering optimal feature space

In this study, we employ three combinations of the available ham ($\mathcal{H}$), spam ($\mathcal{S}$), and phishing (2015: $\mathcal{P}_{2015}$, 2016: $\mathcal{P}_{2016}$) email data, to obtain three datasets, as shown in Table 6. The first dataset comprises ham and spam components, and is aimed at investigating the efficacy of the proposed approaches in spam detection, while the second dataset comprises ham and phishing components, and investigates on the efficacy of the proposed techniques in phishing detection. Such individual analysis is useful in understanding and analyzing the relative importance of features in spam and phishing email detection, respectively. The third dataset comprises all the three components and reflects the fact that real-world email data is composed of ham, spam, and phishing email data. All the experiments performed in this study employ these three datasets.

Not all the features in the obtained feature space contribute towards the accurate classification of the email type, which makes it mandatory to eliminate features of negative

Table 5 The scores of all the forty features concerning the *test email*

Feature	Score	Feature	Score
body_html	True	sender_nonModalSenderDomain	True
body_forms	True	url_ipAddress	True
body_numWords	162	url_numIpAddresses	1
body_numCharacters	1298	url_atSymbol	True
body_numDistinctWords	115	url_numLinks	11
body_richness	0.1248	url_numIntLinks	2
body_numFunctionWords	12	url_numExtLinks	9
body_suspension	False	url_numImgLinks	1
body_verifyYourAccount	True	url_numDomains	8
subject_reply	True	url_maxNumPeriods	3
subject_forward	False	url_linkText	True
subject_numWords	5	url_nonModalHereLinks	True
subject_numCharacters	22	url_ports	True
subject_richness	0.2273	url_numPorts	2
subject_verify	False	script_scripts	True
subject_debit	False	script_javaScript	True
subject_bank	False	script_statusChange	True
sender_numWords	3	script_popups	True
sender_numCharacters	41	script_numOnClickEvents	1
sender_diffSenderReplyTo	False	script_nonModalJsLoads	True

Table 6 Statistics of the datasets utilized in this study

Dataset	Components	Size	#Classes
1	$\mathcal{H}, \mathcal{S}$	3051	2
2	$\mathcal{H}, \mathcal{P}_{2015}, \mathcal{P}_{2016}$	3344	2
3	$\mathcal{H}, \mathcal{S}, \mathcal{P}_{2015}, \mathcal{P}_{2016}$	3844	3

or no importance.[5] We aim at introducing a few of the many feature selection (extraction) techniques, including mRMR (Peng et al. 2005) and PCA (Pearson 1901).

One of the prominent considerations of feature selection (extraction) techniques is the determination of the number of features (dimensions, denoted by k) to extract. There exists no single method to determine k; it is application dependent—a smaller number of dimensions suffice while obtaining insights about the data, while the same is not valid while developing predictive models (Kosinski et al. 2016).

[5] While features with no importance do not hinder the classification performance, they add to the training complexity.

4.3.1 Obtaining the optimal threshold for threshold-based approaches

Several feature selection approaches, including missing values filter and low variance filter, require a threshold to be preset—the threshold is primarily dependent on the input data. That being said, the preset threshold determines if a given feature is important enough to affect the classification or not. Lower values of the threshold include most of the features from the given feature space, thus under-fitting the data, while higher values of the threshold exclude most of the features, causing the loss of critical information. Hence, finding an optimal threshold that facilitates optimal feature selection is vital. The procedure described in Algorithm 1 elucidates on the process of obtaining the optimal threshold. The procedure described in Algorithm 1 utilizes certain utility functions that:

- `scoreFn`(featureColumn): returns the score that is specific to a feature selection technique (e.g., variance in case of low variance filter) for a given feature column.
- `compareFn`(score, threshold): returns a Boolean value that is subject to a technique-specific comparison of the score and the threshold (e.g., score $<$ threshold, returns true for variance filter and feature importance filter, and false for missing values filter).

This procedure (Algorithm 1) is dependent on the underlying machine learning algorithm that is used to compute the performance (accuracy); this study employs an extensive study involving eight state-of-the-art machine learning algorithms (see Sect. 6). Thus, to accommodate all the utilized machine learning algorithms, we chose the smallest, most frequently occurring threshold. Note that the thresholds were computed using the training datasets, and then were utilized on the testing datasets.

4.3.2 Handling the missing attribute values

Usually, handling missing values is accomplished through either deletion techniques such as pair-wise deletion and list-wise deletion, or imputation techniques such as hot-deck imputation, cold-deck imputation, and Monte Carlo simulation based multiple data imputation. In most of the cases, if a data column (feature) has only 5% to 10% of the required data, then it is less likely to be useful in the classification of most samples (Silipo et al. 2014). The missing values ratio captures a relative value indicating the number of missing rows, and this value compared with the preset threshold to infer if data is to be subject to deletion or imputation. The missing values ratio is computed as:

$$\text{Missing values ratio} = \frac{\text{Number of missing rows}}{\text{Total number of rows}} \tag{2}$$

Algorithm 1 Obtaining the value of the optimal threshold

```
procedure OptimalThreshold (dataset, step, scoreFn, compareFn, algorithm)
Variables:
    threshold ← 0.0
    score: Real
    featureColumn: List
    numFeatures ← len(dataset.columns) − 1
    optimalThreshold: Real
    accuracyMax ← 0.0
begin:
    while threshold $\neq$ 1.0 do
        datasetCopy ← dataset
        while numFeatures $\neq$ 0 do
            score ← scoreFn(featureColumn)
            if compareFn(score, threshold) = true then
                datasetCopy.delete(featureColumn)
                accuracy ← algorithm(dataset)
                if accuracyMax < accuracy then
                    accuracyMax ← accuracy
                    optimalThreshold ← threshold
            numFeatures ← numFeatures − 1
        threshold ← threshold + step
    return optimalThreshold
end
```

Algorithm 2 Dealing with missing values in the dataset

```
procedure HandlingMissingValues (dataset, $\theta$, imputationFn)
Constants:
    threshold ← $\theta$
Variables:
    missingValuesRatio: Real
    missingRows: List
    numMissingRows: Real
    featureColumn: List
    totalNumRows ← len(dataset.rows)
    numFeatures ← len(dataset.columns) − 1
begin:
    while numFeatures $\neq$ 0 do
        missingRows ← missing(featureColumn)
        missingValuesRatio ← $\frac{\text{len(missingRows)}}{\text{totalNumRows}}$
        if missingValuesRatio $\leq$ threshold then
            imputationFn(accuracy)
        else
            dataset.delete(featureColumn)
        numFeatures ← numFeatures − 1
end
```

The procedure followed in handling missing attribute values is explained in Algorithm 2. In this procedure, we utilize the utility function `missing`(featureColumn), which returns a list of missing rows in the given feature column. The preset threshold value used in Algorithm 2 can be computed using the procedure in Algorithm 1, with a step value of 0.1 (Silipo et al. 2014). Since the datasets utilized in this study have been programmatically mined, we have considered all possible cases, to avoid any missing values.

4.3.3 Feature selection using low variance filter (LowVar)

Algorithm 3 Removing the features with low variance

```
procedure LowVarianceFilter (dataset, $\theta$)
Constants:
    threshold $\leftarrow \theta$
Variables:
    variance: Real
    featureColumn: List
    numFeatures $\leftarrow$ len(dataset.columns) $-$ 1
begin:
    while numFeatures $\neq 0$ do
        variance $\leftarrow$ Var(featureColumn)
        if variance $<$ threshold then
            dataset.delete(featureColumn)
        numFeatures $\leftarrow$ numFeatures $-$ 1
end
```

One of the many ways of measuring the contribution of a feature (data column) towards the classification performance, is by measuring the variance (sample variance[6]) of the values of that feature. Variance measures the amount of dispersion provided by the values in the given data, and evidently, zero variance is the limiting case, where the values of a feature are constant; such a case offers no inference. Variance (`Var(.)`) is computed as:

$$\texttt{Var}(X) = \frac{1}{N-1}\sum_{x_i \in X}(x_i - \bar{x})^2 \tag{3}$$

where $\bar{x}$ is the arithmetic mean of X. The computed variance is compared with the preset threshold [the threshold obtained using the procedure in Algorithm 1, with a step value of 0.01 (Silipo et al. 2014)] to infer about the contribution of a feature in the classification performance—this study employs a preset threshold of 0.01 for the LowVar approach.

The procedure to remove the features with low variance is described in Algorithm 3. Note that the feature values are normalized prior to low variance filtering, to avoid any unnecessary bias arising due to the data irregularities. It is interesting to note that, by using the correlation between a feature and the target variable as the scoring scheme instead of variance, we obtain a low correlation filter.

[6] This paper uses the terms ‘variance’ and ‘sample variance’ interchangeably. However, all the computations performed in this study employ sample variance, as we only have a sample (3, 844 emails) of all the possible data.

4.3.4 Removing redundancy by measuring correlation (HighCorr)

Algorithm 4 Removing redundancy in the dataset

```
procedure HighCorrelationFilter (dataset, $\theta$)
Constants:
    threshold ← $\theta$
Variables:
    innerIdx: Integer
    outerIdx: Integer
    currentColumn: List
    setColumn : List
    numColumns ← len(dataset.columns)
    correlation: Real
    correlatedColumns: List of Lists
begin:
    for outerIdx ← 0 to numCols do
        setColumn ← dataset.column[outerIdx]
        for innerIdx ← 0 to outerIdx do
            currentColumn ← dataset.column[innerIdx]
            correlation ← Corr(setColumn, currentColumn)
            if correlation ≥ threshold then
                correlatedColumns.add(currentColumn)
    dataset.delete(correlatedColumns)
end
```

Sometimes, the features in a dataset are correlated, i.e., they depend on one another, and thus carry nearly the same information (data redundancy). All redundant features can be replaced with one of the redundant features, without any loss of information. Such replacement can reduce the computational time and enhance prediction accuracy. In this paper, we utilize the Pearson correlation coefficient, denoted by $\texttt{Corr}(X_1, X_2)$ (Pearson 1920; Nagelkerke 1991) [other correlation measures include Kendall Tau correlation and Spearman rank correlation (Bolboaca and Jäntschi 2006)] and given by:

$$\texttt{Corr}(X_1, X_2) = \frac{\texttt{E}[(X_1 - \bar{x_1})(X_2 - \bar{x_2})]}{\sqrt{\texttt{Var}(X_1)} \cdot \sqrt{\texttt{Var}(X_2)}} \tag{4}$$

where $\bar{x_1}$ and $\bar{x_2}$ denote the arithmetic means of X_1 and X_2 respectively, and $\texttt{E[x]}$ denotes the expected value of $\texttt{x}$.

Algorithm 4 details the procedure to eliminate redundancy using a correlation-based filter. Correlation computed using Eq. 4 is compared with a preset threshold [the threshold obtained using the procedure in Algorithm 1, with a step value of 0.1 (Silipo et al. 2014)] to infer if a feature is to be included or excluded in the classification—this study employs a preset threshold of 0.5 for the HighCorr approach.

4.3.5 Measuring feature importance using the random forest classifier (FI)

RFs often referred to as DT ensembles, can be utilized for feature selection (Díaz-Uriarte and De Andres 2006). We can obtain the importance of a feature by using a broad set of carefully constructed trees against the prediction attribute and analyzing the usage statistics of each feature. The process of obtaining the feature importance involves the creation of shallow trees and checking if an attribute appears as a splitting attribute in most of the constructed trees, in which case, that particular feature is regarded as informative. Upon the generation of the ensemble tree, each feature in the feature space can be scored against the number of times that specific feature has been selected as the splitting attribute and at which level of the tree it has been selected. This method is usually robust to noise and is usually faster than boosting and bagging (Breiman 2001).

Usually, feature importance is computed as the Gini impurity or the mean decrease in the impurity (Breiman 2017, 2002; Louppe et al. 2013), which measures the total decrease in the node impurity—a measure of the decrease in the classification performance decreases upon dropping a particular feature. The value of FI for a feature (X_m) can be computed as:

$$\mathtt{Imp}(X_m) = \frac{1}{N_T}\sum_{t=1}^{T}\sum_{n\in\phi_t}(v_n = m)\big[p(n)\cdot\Delta i(n)\big] \tag{5}$$

where N_T is the number of trees, ϕ_t denotes the tth tree structure, n is a node in the tree ϕ_t, v_n denotes the variable at a node n, $p(n)$ is the measure N_n/N of the samples reaching a node n, and $\Delta i(n)$ denotes the impurity reduction (e.g., Shannon entropy, Gini index, and variance of the target variable) at node a n. The impurity reduction at a node n is given by (R: right, L: left):[7]

$$\Delta i(n) = i(n) - \frac{N_{n_L}}{N_n}i(n_L) - \frac{N_{n_R}}{N_n}i(n_R) \tag{6}$$

Upon the computation of the importance of all the features in the feature space using Eq. 5, the FI scores are compared with a preset threshold [the threshold obtained using the procedure in Algorithm 1, with a step value of 0.01 (Silipo et al. 2014)] to infer if a feature is to be included or excluded in the classification—this study employs a preset threshold of 0.06 for the FI approach.

4.3.6 Feature selection using minimum redundancy maximum relevance (mRMR)

The mRMR approach (Peng et al. 2005; Chanduka et al. 2018) is an information-based incremental feature selection technique (filter approach) that aims at integrating the relevance [defined as the distributional similarity between the feature vector and the target vector (Auffarth et al. 2010)] and redundancy (∝ 1/robustness) information into a single scoring function. Relevance can be measured through mutual information (MI) between the given two random variables. MI quantitatively measures the amount of information (bits or Shannons) that two random variables share, and is given by (holds for discrete variables, for continuous variables we integrate over all values of X_1 and X_2):

[7] This study assumes a binary partition (split), which need not be true always.

$$\mathtt{MI}(X_1;X_2) = \sum_{x_2 \in X_2} \sum_{x_1 \in X_1} Pr(x_1, x_2) \cdot \left(\frac{Pr(x_1, x_2)}{Pr(x_1)\, Pr(x_2))} \right) \tag{7}$$

$$\mathtt{MI}(X_1;X_2) = H(X_1) + H(X_2) - H(X_1, X_2) \tag{8}$$

where $Pr(x_1, x_2)$ denotes the joint probability, which measures the likelihood of both x_1 and x_2 occurring together, and is estimated by a histogram or a kernel-based Probability Density Function (PDF) estimator of one or two variables; $Pr(x_i)$ denotes the marginal probability of X_i. MI can be expressed in terms of entropy (see Eq. 8), where the entropy measures the uncertainty of a random variable (Vergara and Estévez 2014) and can be computed as:

$$H(X) = -\sum_{x_i} p(x_i) \cdot \log_2(p(x_i)) \tag{9}$$

Ultimately, we aim at maximizing $\mathtt{MI}(X^{'};Y)$, where $X \in \mathbb{R}^d$ and $X^{'} \in \mathbb{R}^k = \{x^{(1)}, x^{(2)}, \cdots, x^{(k)}\}$, $k < n$. It is hard to estimate the joint probability of high-dimensional variables using a histogram or a kernel-based PDF, as the number of samples needed to estimate the PDF exponentially increases with the increase in the number of dimensions (Rossi et al. 2006). To cope with this issue, we modified the objective function so as to estimate with the available samples.

It is essential to understand that the features contributing to a high MI index need not necessarily be non-redundant, and hence it is crucial to consider redundancy along with MI, to obtain an optimal representative set of k features. The objective function Φ (mRMR[8]) is employed to balance the trade-off between redundancy and relevance; is computed using:

$$\Phi = R - R^{-} = \frac{1}{|X'|} \sum_{x^{(i)}} \mathtt{MI}(x^{(i)};Y) - \frac{1}{|X'|^2} \sum_{x^{(i)},x^{(j)}} \mathtt{MI}(x^{(i)};x^{(j)}) \tag{10}$$

where R measures the average relevance of the feature vectors with the target vector, while R^{-} captures the average pair-wise redundancy among the selected features, and thus, by maximizing the objective function, we can obtain an optimal feature subspace. The incremental approach is facilitated by adding one feature at a time to the set $X^{'}$, starting from the feature that maximizes the objective function. For every feature addition, the cross-validation classification error is computed—the reduced feature space is the subspace with the least classification error. In this study, we utilize the mRMR feature selection approach as a wrapper approach, with $C4.5$ DT and 10-fold cross-validation. Moreover, binning was employed ton discretize the continuous data, before subjecting the data to mRMR feature selection.

Sometimes, the mRMR approach generates high error thresholds (as high as 34%). Moreover, mRMR only considers pair-wise interactions (see Eq. 10); by considering higher-order interactions, we can obtain more informative subspaces. Maximum Joint

[8] The mRMR approach facilitates two variants including MID (difference), where Φ = relevance−redundancy, and MIQ (quotient) where Φ = relevance/redundancy. This study employs the MID variant of mRMR.

Relevance (MJR) (Yang and Moody 2000) and adaptive MJR (Jiao et al. 2015) are a few of the modified mRMR algorithms that are aimed at tackling these shortcomings.

4.3.7 Feature extraction using principal component analysis (PCA)

PCA is an unsupervised approach that aims at converting a set of observations of (possibly) correlated variables into a set of values of uncorrelated variables (principal components) using orthogonal transformations (Pearson 1901; Wold et al. 1987). PCA aims at maximizing the variance of the data in a new dimensional space. PCA produces the same number of orthogonal dimensions as that of the initial data, but what makes PCA interesting is that the eigenvalues corresponding to these eigenvectors (principal components) monotonically decrease as we move away from the first principal component. The dimension with an eigenvalue of approximately zero value (zero variance) does not provide any information in the original space and can be considered to be irrelevant.[9]

PCA usually provides the *best reconstruction*, i.e., the loss of information from the transformation is minimal, and this can be attributed to the fact that PCA only performs linear transformations. PCA makes a compelling assumption of the presence of a linear relationship between observed variables, and also that all the data points are Independent and Identically Distributed (IID). Consider PCA for a single dimension subspace, where $X \in \mathbb{R}^d$ and $\{x_1, x_2, \cdots, x_n\}$ are IID distributions of X ($d \ll n$). We aim at maximizing $u^T \Sigma u$ subject to $u^T u = 1$, where Σ is the covariance matrix ($\in \mathbb{R}^{d \times d}$), and u is a principal component ($\in \mathbb{R}^{d \times k}$). Using Lagrange multipliers (Klein 2004), we obtain $\Sigma u = \lambda u$, for some λ. So, u is an eigenvector of Σ, with an eigenvalue of λ.

The preprocessing steps in PCA include zeroing out the mean of the data, and normalizing the variance of each coordinate, to ensure they are all measured on the same scale. Then, we compute Σ, followed by the computation of eigenvalues and eigenvectors. If we intend on projecting the data into a k–dimensional space ($k < n$), we should choose top–k eigenvectors of Σ, i.e., $\{u_1, u_2, \cdots, u_k\}$, which then form the basis of the new orthogonal space. Any given data point $X \in \mathbb{R}^d$ can be represented in the new basis as:

$$X' = u^T X = \left[u_1^T X u_2^T X \cdots u_k^T X\right]^T; X = \left[x^{(1)} x^{(2)} \cdots x^{(d)}\right]^T \tag{11}$$

Now, we know that all the dimensions in the projected space are orthogonal, and thus, we can ensure that the variables are uncorrelated. PCA is comparatively fast, owing to the ease of computation concerning eigenvectors (Hand 2007). Furthermore, PCA provides the ease of interpretability and visualization. In this study, we only retained those principal components of PCA that accounted for 90% of the variance.

4.3.8 Using python for feature selection (extraction)

In this section, we explain the way of obtaining an optimal feature subspace from the given feature space through LowVar, HighCorr, FI, mRMR, and PCA approaches, using Python. The low variance filter and high correlation filter can be implemented by following the procedure in Algorithm 3 and Algorithm 4, respectively. Alternatively, the implementations in the Python `pandas.corr` (for high correlation filter) and `sklearn.`

[9] Note that the difference between 'irrelevant' and 'useless' is that irrelevant features have zero entropy while the usefulness of a feature is application-specific.

`feature_selection.VarianceThreshold` (for low variance filter) can be utilized to achieve the same (see Block 8).

Block 8 Code block to facilitate feature selection using LowVar and HighCorr.

```
# Using LowVar (threshold of 0.01) to facilitate feature selection
selector = VarianceThreshold(threshold=0.01)
transformedData = selector.fit_transform(trainingData)

# Using HighCorr (threshold of 0.5) to facilitate feature selection
corrMatrix = trainingDataframe.corr().abs()
upperTriangle = corrMatrix.where(np.triu(np.ones(corrMatrix.shape),
     k=1).astype(np.bool))
dropFeatures = [column for column in upperTriangle.columns if any(
     upperTriangle[column] > 0.5)]
trainingDataframe.drop(trainingDataframe[upperTriangle], axis=1)
```

To obtain the importance of the features in the obtained feature space using the RF classifier, we utilized the implementations available in the Python `sklearn.ensemble.RandomForestClassifier` library. The code in Block 9 elucidates on the implementation details concerning the computation of the FI. Note that the code presented here utilizes 100 classification and regression trees with a maximum depth of 2.

Block 9 Code block to facilitate feature selection by computing the importance of features through RF classifier.

```
# Using the RF classifier to classify the training data
classifier = RandomForestClassifier(n_estimators=100, max_depth=2)
classifier.fit(trainingData, targetClasses)

# Obtaining the feature importances using the trained classifier
featureImportances = classifier.feature_importances_
```

To implement the mRMR approach in Python, we utilize the implementations in the `pymrmr` library. The code in Block 10 details the process of feature selection using mRMR. The code presented here takes as the input, a discretized dataframe, a method of internal selection (MID or MIQ), and the value of k (number of dimensions). To discretize a continuous attribute ($X^{(i)}$) based on two thresholds, we use $\texttt{Mean}(X^{(i)}) \pm (\psi \times \texttt{Var}(X^{(i)}))$, where ψ can be 0, 0.5, or 1 (Peng et al. 2005).

Block 10 Code block to facilitate feature selection using mRMR.

```
# Obtaining the optimal feature subspace of ten features using mRMR
pymrmr.mRMR(discretisedDataframe, ‘MIQ’, 10)
```

Finally, to perform PCA and find the directions of maximum variance using Python, we employ the implementations in the `sklearn.decomposition.PCA` library. Upon fitting the PCA model, the principal components and eigenvalues can be accessed via *components_* and `explained_variance_` attributes.

Block 11 Code block to facilitate feature extraction using PCA.

```
# By default, numDimensions = min(numSamples, numFeatures)
pca = sklearn.decomposition.PCA(n_components=None)
pca.fit(dataMatrix)
newDimensions = pca.components_
```

5 Methods: email classification

In recent years, most researchers have resorted to machine learning approaches to detect and differentiate between ham, spam, and phishing emails. Machine learning algorithms facilitate a sense of experience-based learning, resulting in the automatic generation of adaptive classification rules, in turn enhancing the performance. Such adaptive and automated approaches outperform blacklisting or rule-based filtering approaches which rely on hand-coded rules susceptible to the changing nature of spam and phishing email attacks. In this section, we review eight state-of-the-art machine learning algorithms employed in UBE classification. The Python code is presented in-line with the text, to aid readers to implement the proposed classifiers.

5.1 Classification using Naïve bayes (NB)

The NB classifier exemplifies both supervised learning and statistical learning. NB serves as a straightforward probabilistic approach that classifies the input email data by influencing the probabilities of the outcomes. The Bayesian classification merges the experimental data with the previous knowledge, and can solve both predictive and analytical problems. Furthermore, the NB algorithm is robust to noise, and computes likelihoods for postulation. Note that, the NB classifier is based on the Bayes theorem with a sound assumption of independent events. The Bayes probability theorem is an autonomous characteristic model (Wu and Deng 2008; Issac and Jap 2009), and is given as:

$$\begin{aligned} Pr(\text{class}|(x_1, x_2, \dots, x_n)) &= \frac{Pr((x_1, x_2, \dots, x_n) \text{ and class})}{Pr((x_1, x_2, \dots, x_n))} \\ &= \frac{Pr(\text{class})}{Pr((x_1, x_2, \dots, x_n))} \prod_{i=1}^{n} Pr(x_i|\text{class}) \end{aligned} \tag{12}$$

where n denotes the number of features in the feature space. Since the value $Pr((x_1, x_2, \dots, x_n))$ is a constant, the classification rule can be rewritten as:

$$Pr(\text{class}|(x_1, x_2, \dots, x_n)) \propto Pr(\text{class}) \prod_{i=1}^{n} Pr(x_i|\text{class}) \tag{13}$$

$$\hat{y} = \underset{\text{class}}{\arg\max}\, Pr(\text{class}) \prod_{i=1}^{n} Pr(x_i|\text{class}) \tag{14}$$

The notion of class restrictive autonomy was utilized to ensure the ease of computation, thus, tagging the Bayesian classification as *naïve*—nevertheless, the classifier is robust, effective, and computationally efficient. Owing to the promising performance of the NB classifier, it has been adopted to solve several real-world tasks, including spam detection, recommender systems, and sentiment analysis (social media analytics). Additionally, due to its superior performance in multi-class problems, it has been exclusively adopted to text classification tasks. It is interesting to note that Bayesian spam filters have been widely implemented by many email clients—the software that ensures the effective performance of email clients is entrenched with server-side email filters utilizing Bayesian filters. Generally, a Gaussian NB classifier is utilized to accommodate numerical features, where the likelihood of the features is assumed to be Gaussian (normally distributed):

$$Pr(x_i|\text{class}) = \frac{1}{\sqrt{2\pi \mathtt{Var}(\text{class})}} \cdot \exp\left(-\frac{(x_i - \mu_{\text{class}})^2}{2\,\mathtt{Var}(\text{class})}\right) \tag{15}$$

However, in this study, we employ the supervised discretization approach to discretize the continuous attributes as it overcomes the assumption of the normality of continuous features.

To facilitate the classification of UBEs using the NB classifier, we utilize the implementations in the Python `sklearn.naive_bayes.GaussianNB` library, as shown in Block 12.

Block 12 Code block to facilitate classification using NB classifier.

```
# Using the NB classifier to learn from the training data
classifier = GaussianNB()
classifier.fit(trainingData, targetClasses)

# Using the NB classifier to classify the testing data
predictions = classifier.predict(testingData)
```

5.2 Classification using support vector machines (SVM)

The SVM classifier is a supervised learning algorithm that solves both regression and classification problems, and is proven to superior in performance when compared to several attendant learning algorithms (Sculley and Wachman 2007). The applications of SVM include solving quadratic programming problems with inequality constraints and linear equality, by differentiating groups using hyperplanes. Despite the higher training time in comparison to several other classifiers, the SVM classifier facilitates promising results, owing to its capacity to model multi-dimensional borderlines which are neither straightforward nor sequential. Furthermore, the classifier model is not *disproportionately complex*, in the sense that the number of trainable parameters is lower than the number of observations, thus making SVM an ideal suit for real-world tasks like speech and handwriting recognition.

To understand the SVM classifier, let us consider the simple case of a binary classification problem, with features x and target classes $y \in \{-1, +1\}$, where data points are linearly separable. Let us consider two support vectors (forming a street) passing through the data points lying closest to the decision surface (hyperplane), and a vector $\bar{w}$ that is perpendicular to the median line of the street. Ultimately, we need to find support vectors that maximize the street width, thus finding the optimal decision surface. For an unknown sample $\bar{u}$, by measuring the projection of the unknown sample on to the perpendicular vector, we can determine if the sample is a positive ($y = +1$) or negative ($y = -1$), i.e., $\bar{w} \cdot \bar{u} \geq c$ or $\bar{w} \cdot \bar{u} + b \geq 0$ for a positive sample. Now, for a positive training sample (x_+), we have $\bar{w} \cdot \bar{x}_+ + b \geq 1$, and likewise, for a negative training sample ($x\_$), we have $\bar{w} \cdot \bar{x}_- + b \leq -1$. So,

$$y^{(i)}(\bar{w} \cdot \bar{x}^{(i)} + b) - 1 \geq 0 \tag{16}$$

where $y^{(i)} = 1$ for positive samples ($y = +1$) and $y^{(i)} = -1$ otherwise. Let $x_+^{(c)}$ and $x_-^{(c)}$ be the points on the support vectors, note that, $y^{(i)}(\bar{w} \cdot x^{(c)} + b) - 1 = 0$ for $x^{(c)} \in \{x_+^{(c)}, x_-^{(c)}\}$. Now, we can compute the street width as:

$$\text{width} = (x_+^{(c)} - x_-^{(c)}) \cdot \frac{\bar{w}}{||\bar{w}||_2} = \frac{2}{||\bar{w}||_2} \tag{17}$$

Now, we transform the optimization problem from maximizing the street width, to:

$$\max \frac{2}{||\bar{w}||_2} \text{ (or) } \min ||\bar{w}||_2 \text{ (or) } \min \frac{1}{2}||\bar{w}||_2^2 \tag{18}$$

Now, using Lagrange multiplier α_i (constrained to be ≥ 0), we have the Lagrangian as:

$$\mathcal{L}(\bar{w}, b, \alpha) = \frac{1}{2}\bar{w}^{\mathrm{T}}\bar{w} - \sum \alpha_i[y^{(i)}(\bar{w} \cdot \bar{x}^{(i)} + b) - 1] \tag{19}$$

Now, by differentiating with respect to $\bar{w}$ and b, we get:

$$\frac{\partial \mathcal{L}}{\partial \bar{w}} = \bar{w} - \sum \alpha_i y^{(i)} \bar{x}^{(i)} = 0 \implies \bar{w} = \sum \alpha_i y^{(i)} \bar{x}^{(i)} \tag{20}$$

$$\frac{\partial \mathcal{L}}{\partial b} = -\sum \alpha_i y^{(i)} = 0 \tag{21}$$

Using Eqs. 20 and 21 in Eq. 19, we can simplify the Lagrangian as:

$$\mathcal{L}(\bar{w}, b, \alpha) = \sum \alpha_i - \frac{1}{2}\sum_i \sum_j (\alpha_i \alpha_j)(y^{(i)} y^{(j)})(\bar{x}^{(i)} \bar{x}^{(j)}) \tag{22}$$

Now, using Eq. 20 in the decision rule of the unknown sample ($\bar{u}$) to be a positive sample, we get:

$$\sum \alpha_i y^{(i)} \bar{x}^{(i)} \cdot \bar{u} + b \geq 0 \tag{23}$$

From Eqs. 22 and 23, we observe that the decision rule depends on the dot product of the sample vectors and the unknown vector. Now, when the data points are not linearly separable, we transform (using function ϕ) the data points to a space where they are separable, i.e.,

$$\mathcal{K}(\bar{x^{(i)}}, \bar{x^{(j)}}) = \phi(\bar{x^{(i)}}) \cdot \phi(\bar{x^{(j)}}) \tag{24}$$

Note that, all that we need to know is the kernel function $\mathcal{K}$ (e.g., linear, Radial Basis Function (RBF), and sigmoid) that facilitates the transformation into the new space, rather than the transformation itself. In this study, we employ the SVM classifier with an RBF kernel and a cost factor of 32 (obtained empirically using grid search). The cost factor aims at regulating the modeling error that results when the function is fit too close to the data points.

To facilitate the classification of UBEs using the SVM classifier, we utilize the implementations in the Python `sklearn.svm.SVC` library, as shown in Block 13.

Block 13 Code block to facilitate classification using SVM classifier.

```
# Using the SVM classifier to learn from the training data
classifier = SVC(kernel=‘rbf’, C=32)
classifier.fit(trainingData, targetClasses)

# Using the SVM classifier to classify the testing data
predictions = classifier.predict(testingData)
```

5.3 Ensemble classifiers

Ensemble learning is an approach of grouping several classifiers for training on the input data, intended on improving the classification performance. Researchers have advocated the assembling of various classifiers to handle UBE attacks effectively (Guerra et al. 2010). In this study, we employ six widely used ensembling approaches to facilitate UBE classification.

5.3.1 Classification using bagged decision trees (BDT)

A DT is a supervised learning approach that decomposes complex problems into a hierarchy of simpler ones. The internal nodes of a DT pave the way to the final decision rule, each time (at each level) adding to the previous decision rule, while the leaf nodes associate an output (class label) to the input features. Sometimes, DT tends to overfit the data, owing to the stringent decision rules at various levels of the tree. To cope with this issue, bootstrap-aggregated (bagged) DT aims at combining the results of several DT classifiers. This approach enhances generalizability and is hence adopted in a variety of tasks including spam detection and power system fault detection. BDT classifier is effective in mapping more than one parameter to a target variable (Netsanet et al. 2018) and hence is extremely useful in UBE classification.

To understand the process of bagging, let us consider the training set T to be $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)})\}$, where $x^{(i)} \in X$ and $y^{(i)} \in \Omega = \{l_1, l_2, \ldots, l_k\}$. A classifier $\mathcal{C}$ aims at mapping from T to a decision rule $(\hat{f})$, which then maps X to Ω, i.e., $\mathcal{C}(T) = \hat{f}$ and $\hat{f}(x) \in \Omega$. Now, a bootstrap sample $T_b = \{x_b^{(i)}, y_b^{(i)}\}_{i=1}^{n}$ is obtained through independent draws from T, with replacement. The obtained T_b produces the decision rule $\hat{f}_b = \mathcal{C}(T_b)$, and the final bootstrap-aggregated estimate $\hat{F}_b$ is computed as the majority vote of all the B bootstrap predictors:

$$\hat{F}_b = \underset{y\in\Omega}{\arg\max} \sum_{i=1}^{B} I_{\{y=\hat{f}_b(x)\}} \tag{25}$$

where $I_{\{M\}}$ is the indicator of M. Intuitively, bagging serves as a variance reduction process that mimics the procedure of averaging over various training sets. In this study, we employ BDT classifier with 100 $C4.5$ DT estimators. Moreover, we employ the Gini impurity in the measurement of the quality of the split.

To facilitate the classification of UBEs using the BDT classifier, we utilize the implementations in the Python `sklearn.ensemble.BaggingClassifier` library (we used the Python `sklearn.tree.DecisionTreeClassifier` library to implement the DT classifier), as shown in Block 14.

Block 14 Code block to facilitate classification using BDT classifier.

```
# Using the BDT classifier to learn from the training data
treeModel = DecisionTreeClassifier()
classifier = BaggingClassifier(base_estimator=treeModel,
    n_estimators=100)
classifier.fit(trainingData, targetClasses)

# Using the BDT classifier to classify the testing data
predictions = classifier.predict(testingData)
```

5.3.2 Classification using random forest (RF)

While BDT classifier is effective in classification, the trees produced by a BDT classifier can be very similar, and thus, slowing down the learning process. The RF classifier overcomes this shortcoming by employing two sources of randomness including bagging and random input vectors. RF uses DT classifiers to facilitate prediction of the target variable. RF classifier has been shown to have better performance (low error rate) than several learners such as SVM and DT, in several classification tasks including speech analysis and UBE detection. Furthermore, RF performs well even in the cases of disproportionate data characterized by missing variables, by providing an approximation to the missing data and preserving the precision in cases where a significant amount of data is lost.

To understand the process of classification using RF, let us consider the training set T to be $\{(x^{(1)},y^{(1)}),(x^{(2)},y^{(2)}),\dots,(x^{(n)},y^{(n)})\}$, where $x^{(i)} \in X$ $(X \in \mathbb{R}^p)$ and $y^{(i)} \in \Omega = \{l_1,l_2,\dots,l_k\}$. Now, a bootstrap sample $T_b = \{x_b^{(i)},y_b^{(i)}\}_{i=1}^{n}$ is obtained through independent draws from T, with replacement. The obtained T_b is used to generate an RF tree Tr_b. At every node of Tr_b, we choose m out of p features (optimal value is $\sqrt{p}$), select the splitting attribute among the m selected features using IG or Gini impurity. Then, we split the current node based on the chosen splitting attribute. This procedure is recursively repeated until the minimum node size $n_{\min}$ (maximum tree depth) is obtained. Ultimately, the classification is facilitated as:

$$\hat{y}(x) = \text{majority vote}\big\{y_b(x)\big\}_{b=1}^{B} \tag{26}$$

In this study, we employ the RF classifier with 100 $C4.5$ DT classifiers, and the nodes of the tree are expanded until all the leaf nodes contain less than two samples or until all the leaf nodes are pure. Moreover, we employ the Gini impurity in the measurement of the quality of the split. The RF classifier is implemented using the procedure in Algorithm 5.

Algorithm 5 Random forest algorithm for UBE classification

```
procedure RandomForest (trainingSamples, B, $n_{\min}$)
Constants:
    m ← $\sqrt{p}$
Variables:
    $b$: Integer
begin:
    for $b \leftarrow 1$ to $B$ do
        From the trainingSamples, draw a bootstrap sample $T_b$ of size $n$
        while $n_{\min} > 0$ do
            Randomly select $m$ out of $p$ features
            Select the splitting attribute of the tree $Tr_b$ among $m$ features
            Split the node into two daughter nodes
    Output the ensemble of all the generated trees $\{Tr_b\}_{b=1}^{B}$
end
```

To facilitate the classification of UBEs using the RF classifier, we utilize the implementations in the Python `sklearn.ensemble.RandomForestClassifier` library, as shown in Block 1.

Block 15 Code block to facilitate classification using RF classifier.

```
# Using the RF classifier to learn from the training data
classifier = RandomForestClassifier(n_estimators=100)
classifier.fit(trainingData, targetClasses)

# Using the RF classifier to classify the testing data
predictions = classifier.predict(testingData)
```

5.3.3 Classification using extra trees (ET)

The extremely randomized trees classifier was aimed at randomizing the tree building further, in the context of numerical input attributes, where the choice of the optimal cut-point (discretization threshold) is responsible for a large proportion of the variance induced in the tree. Experiments (Geurts et al. 2006) have shown that the ET classifier is competitive with the RF classifier in terms of accuracy, and sometimes superior (especially when the data is noisy). Moreover, since the need for the optimization of discretization thresholds is removed in ET classifiers, they are computationally fast and easy to implement. The ET classifier has yielded state-of-the-art results in various high-dimensional complex problems.

The ET classifier is similar to an RF classifier in the sense that both these algorithms are based on choosing m (out of p, optimally $\sqrt{p}$) features at each node, to determine the split. However, unlike in an RF classifier, an ET classifier learns from the entire learning sample T (no bootstrap copying) or a sample drawn from T without replacement. More importantly, instead of choosing from the best cut-point based on the local sample as in BDT or RF, an ET classifier randomly selects the cut-point to determine the split. It is interesting to note that the algorithm is primarily reliant on the value of m, and when $m = 1$, the resulting extra tree structure is built independently of the target class labels in the training set. From a statistical perspective, dropping the randomization through bagging leads to an advantage concerning the bias, while cut-point randomization often leads to an excellent reduction in the variance. From a functional perspective, the ET approach facilitates piece-wise multi-linear approximations as opposed to piece-wise constant approximations of RF classifiers. In this study, we employ the ET classifier with 100 $C4.5$ DT classifiers, and the nodes of the tree are expanded until all the leaf nodes contain less than two samples or until all the leaf nodes are pure. Moreover, we employ the Gini impurity in the measurement of the quality of the split.

To facilitate the classification of UBEs using the ET classifier, we utilize the implementations in the Python `sklearn.ensemble.ExtraTreesClassifier` library, as shown in Block 14.

Block 16 Code block to facilitate classification using ET classifier.

```
# Using the ET classifier to learn from the training data
classifier = ExtraTreesClassifier(n_estimators=100)
classifier.fit(trainingData, targetClasses)

# Using the ET classifier to classify the testing data
predictions = classifier.predict(testingData)
```

5.3.4 Classification using AdaBoost (AB)

The adaptive boosting algorithm is a meta-estimator that combines several weak decision rules into one strong decision rule, and is shown to provide good performance even with the unsatisfactory performance of the individual weak learners. By convention, a strong learner is the one with an error rate close to zero, while a weak learner is the one with an error rate just below 0.5. AB is widely adopted, owing to the astounding performance of the algorithm in a wide variety of classification tasks, including UBE classification and text categorization. Furthermore, AB is straightforward, adaptive, fast, easy to program, and less cumbersome (due to minimal parameter tuning).

To understand the AB classifier, let us consider the simple case of a two-class problem, with training samples $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(n)}, y^{(n)})\}$, where $x^{(i)} \in X$ and $y^{(i)} \in \{-1, +1\}$. In each round $t = 1, 2, \dots, T$, we compute a distribution D_t over the (n) training samples. A weak learner is utilized to compute a weak hypothesis h^t, where the weak learner is aimed at generating h^t with low weighted error $\mathcal{E}_t$ relative to D_t. At every step the distribution is normalized using a factor Z_t, to ensure that D_{t+1} is a distribution. The final hypothesis $H(t)$ computes the overall majority vote (sign) of all the weak learners through a weighted combination of weak hypotheses, where each hypothesis is weighted by α^t. The entire procedure for the AB algorithm is shown in Algorithm 5. Alternatively, for multi-class (more than two classes) problems, we have Stagewise Additive Modeling using a Multi-class Exponential loss function (SAMME) (Hastie et al. 2009), which implements the multi-class Bayes rule by modeling a forward stagewise additive model. A widely used variant of SAMME is the SAMME.R algorithm (R for Real), which converges faster than SAMME, and achieves a lower test error with fewer rounds. In this study, we employ the AB classifier with a $C4.5$ DT classifier for 100 rounds and the SAMME.R algorithm in the case of three-class classification. The procedure shown in Algorithm 5 is employed in the implementation of the AB classifier.

Algorithm 6 AdaBoost algorithm for UBE classification

```
procedure AdaBoost (trainingSamples)
Constants:
    $D_1(i) \leftarrow 1/n$, for $i = 1, 2, \ldots n$
Variables:
    $t$: Integer
    $T$: Integer
    $i$: Integer
    $Z_t$: Real
begin:
    for $t \leftarrow 1$ to $T$ do
        Train a weak learner using the distribution $D_t$
        Obtain the weak hypothesis $h^t : X \to \{-1, +1\}$
        Select $h^t$ with low weighted error, $\mathcal{E}_t \leftarrow Pr_{i \sim D_t}[h^t(x^{(i)}) \neq y^{(i)}]$
        Choose $\alpha^t = \frac{1}{2} \log_e \left( \frac{1-\mathcal{E}_t}{\mathcal{E}_t} \right)$
        for $i \leftarrow 1$ to $n$ do
            $D_{t+1}(i) \leftarrow \frac{D_t(i) \exp(-\alpha^t y^{(i)} h^t(x^{(i)}))}{Z_t}$
    Final hypothesis, $H(x) = \text{sign} \left( \sum_{t=1}^{T} \alpha^t h^t(x) \right)$
end
```

To facilitate the classification of UBEs using the AB classifier, we utilize the implementations in the Python `sklearn.ensemble.AdaBoostClassifier` library, as shown in Block 15.

Block 17 Code block to facilitate classification using AB classifier.

```
# Using the AB classifier to learn from the training data
classifier = AdaBoostClassifier(n_estimators=100)
classifier.fit(trainingData, targetClasses)

# Using the AB classifier to classify the testing data
predictions = classifier.predict(testingData)
```

5.3.5 Classification using stochastic gradient boosting (SGB)

The AB and related classifiers (step-wise algorithms) are categorized under adaptive re-weighting and combining statistical framework, where the objective is to minimize the weighted error, followed by a re-computation of the weak hypotheses. Gradient boosting machines enhance this framework further, by casting the process of boosting as a numerical optimization with an objective of loss minimization through the addition of weak learners using the steepest gradient algorithm. In the SGB approach, we add a new weak learner at a time, while the existing weak learners are left unchanged, and thus, facilitating a stage-wise additive approach. The SGB algorithm is related to both bagging and boosting, where many small trees are built sequentially from the gradient of the loss function of the previous tree (*pseudo*-residuals). At each round, a tree is built from a random bootstrap sample (drawn without replacement), resulting in an incremental improvement in the model. Thus, the SGB algorithm is computationally fast, resistant to outliers, and avoids over-fitting of the data, and is hence adopted in a variety of applications including microscopy image analysis and slate deposit estimation.

To understand the working of the SGB classifier, let us first understand a naïve formalization of gradient boosting. Let us consider the training set T to be $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(n)}, y^{(n)})\}$, where $x^{(i)} \in X$ and $y^{(i)} \in \Omega = \{l_1, l_2, \dots, l_k\}$. A classifier $\mathcal{C}$ aims at mapping from T to a decision rule $(\hat{f})$, which then maps X to Ω, i.e., $\mathcal{C}(T) = \hat{f}$ and $\hat{f}(x) = y \in \Omega$. First, let us fit a model to T, i.e., $\hat{f_0}(x) = y$. Now, let us fit another model $\hat{h_0}$ to the residuals obtained, i.e., $\hat{h_0}(x) = y - \hat{f_0}(x)$. Now, in the subsequent round, create a stage-wise additive model to correct the errors of the previous model as $\hat{f_1}(x) = \hat{f_0}(x) + \hat{h_0}(x)$. Now, let us generalize this idea for R rounds as:

$$\hat{f_R}(x) = \hat{f_0}(x) \mapsto \hat{f_1}(x) = \hat{f_0}(x) + \hat{h_0}(x) \cdots \mapsto \hat{f_R}(x) \tag{27}$$

$$= f_{\hat{R-1}}(x) + h_{\hat{R-1}}(x) \tag{28}$$

At each step r, we aim at finding $\hat{h_r}(x) = y - \hat{f_r}(x)$. In practice, $\hat{h_r}$ is almost always a tree-based classifier. Now, let us tweak the model to conform to the actual SGB classifier; since we aim at minimizing the loss function (L), let us initialize $\hat{f}$ with the mean of the target classes in T, i.e.,

$$\hat{f_0}(x) = \arg\min_{\gamma} \sum_{i=0}^{n} L(y^{(i)}, \gamma) \tag{29}$$

Now, we recursively define each subsequent $\hat{f_r}$ $(r \geq 0)$ as $\hat{f_r}(x) = f_{\hat{r-1}}(x) + h_{\hat{r-1}}(x)$, where $h_{\hat{r-1}}(x)$ is a classifier that aims at fitting the residuals (σ_{r-1}) (computed as the gradient of the loss function), i.e.,

$$\sigma_{r-1} = -\frac{\partial L(y, f_{\hat{r-1}}(x))}{\partial f_{\hat{r-1}}(x)} \tag{30}$$

The final learner obtained after R rounds $(\hat{f_R})$ is the trained SGB classifier. In this study, we employ a SGB learner with a $C4.5$ DT classifier of maximum depth two $(\hat{h}(x))$, trained for 100 rounds. Moreover, we employ deviance as the loss function, which measures the goodness of the fit.

To facilitate the classification of UBEs using the SGB classifier, we utilize the implementations in the Python `sklearn.ensemble.GradientBoostingClassifier` library, as shown in Block 18.

Block 18 Code block to facilitate classification using SGB classifier.

```
# Using the SGB classifier to learn from the training data
classifier = GradientBoostingClassifier(num_estimators=100,
    max_depth=2)
classifier.fit(trainingData, targetClasses)

# Using the SGB classifier to classify the testing data
predictions = classifier.predict(testingData)
```

5.3.6 Classification using voting ensemble (VE)

A voting ensemble classifier is a naïve approach to aggregating the predictions of a variety of diverse classifiers using a majority rule. For a set of classifiers C^rs (total R classifiers) trained on the same training data ($T = \{x^{(i)}, y^{(i)}\}_{i=1}^{n}$, $y^{(i)} \in \Omega$), we have predictions (y^rs) such that $C^r(x) = y^r$, where $y^r \in \Omega$. Now, the final classification is facilitated as:

$$\hat{y}(x) = \text{majority vote}\{y^r\}_{r=1}^{R} \tag{31}$$

Such voting is often referred to as the *hard* voting scheme. In this study, we employ a VE classifier with seven diverse classifiers including Gaussian NB, logistic regression, $ID3$ DT, RF, ET, AB, and SGB (with the parameters described in the above sections). Additionally, we tested the plurality voting scheme; however, the majority voting scheme outperformed the plurality voting scheme.

To facilitate the classification of UBEs using the VE classifier, we utilize the implementations in the Python `sklearn.ensemble.VotingClassifier` library, as shown in Block 19.

Block 19 Code block to facilitate classification using VE classifier.

```
# Creating the sub-models to be used by the voting classifier
subModels = []
subModels.append((‘DT’, DecisionTreeClassifier()))
subModels.append((‘Logistic’, LogisticRegression()))
subModels.append((‘SVM’, SVC()))

# Using the VE classifier to learn from the training data
classifier = VotingClassifier(subModels)
classifier.fit(trainingData, targetClasses)

# Using the VE classifier to classify the testing data
predictions = classifier.predict(testingData)
```

5.4 WEKA workbench for machine learning

Apart from Python programming, the Waikato Environment for Knowledge Analysis (WEKA) workbench (Hall et al. 2009) is recognized as a landmark system in machine learning and data mining, which provides a toolbox of learning algorithms, along with a framework for the development of novel algorithms without the burden of the supporting infrastructure for scheme evaluation and data manipulation.

The WEKA project aims to provide a comprehensive collection of data preprocessing and machine learning algorithms for practitioners and researchers. It facilitates easy and quick comparison of several machine learning algorithms on datasets. Furthermore, the WEKA graphical user interface enables beginners to seamlessly perform data preprocessing, regression, classification, clustering, feature selection, association rule mining, and visualization. The WEKA tool has achieved widespread acceptance in business and academia alike, and has become a widely adopted tool for the research in data mining. Table 7 tabulates the capabilities of several machine learning and feature selection approaches employed in this study, with respect to WEKA workbench.

6 Performance evaluation and discussion

To evaluate the efficacy of the utilized feature selection (extraction) and machine learning algorithms in spam and phishing email detection, we performed extensive experimentation on the datasets described in Table 6. All the experiments in this study were performed on a PC with Intel Core i7 × 2.5 GHz with 16 GB RAM in the Mac 10.14 OS. Furthermore, all the experiments were carried out through 10-fold cross-validation, and the overall performance was computed as the average across all the folds. In this section, we first discuss the evaluation metrics employed in this study and their relevance concerning UBE detection. Then, we present the results of our experimentation, followed by a discussion on the implications of the presented results.

Table 7 Capabilities of the algorithms concerning WEKA workbench

Class	Algorithm	Allowed class types	Allowed attribute types
Feature selection	LowVar	–	Continuous
	HighCorr	–	Continuous
	FI	Discrete	Continuous and discrete
	mRMR	Discrete	Discrete
	PCA	–	Continuous and discrete
Classification	NB	Discrete	Continuous and discrete
	SVM	Discrete	Continuous and discrete
	BDT	Continuous and discrete	Continuous and discrete
	RF	Continuous and discrete	Continuous and discrete
	ET	Continuous and discrete	Continuous
	AB	Discrete	Continuous and discrete
	SGB	Discrete	Continuous and Discrete
	VE	Continuous and discrete	Continuous and discrete

6.1 Performance evaluation metrics

Most of the works in the existing literature employ classification accuracy as the key performance indicator (see Table 2). However, only measuring the number of correctly classified email messages is not sufficient, owing to the costs attached with the misclassification of UBEs; other metrics derived from information retrieval and decision theory (e.g., precision and recall) can help gain better insights into the obtained results. When a spam email message is misclassified as a ham email, it causes a rather insignificant problem (user only needs to delete such an email). However, when ham emails are misclassified as spam or phishing emails, there is a possibility of losing vital information (specifically in scenarios where spam emails are deleted automatically), while phishing emails that are misclassified as ham emails result in a breach of privacy (a much more serious concern). Moreover, in scenarios with imbalanced data (such as in our case), accuracy does not consider all the relevant aspects of the classification inference. In this study, we employ seven standard evaluation metrics including accuracy, precision, recall, F1-measure (F1 score), Matthews correlation coefficient (MCC) score, area under the ROC curve (AUROC), and Area Under the Precision-Recall Curve (AUPRC), to assess the performance of our extensive evaluation accurately.

Accuracy: This metric aims at evaluating the average number of correctly classified email messages over the given email corpus. The classification accuracy can be computed using:

$$\text{Accuracy} = \frac{|\mathcal{H} \to \mathcal{H}| + |\mathcal{S} \to \mathcal{S}| + |\mathcal{P} \to \mathcal{P}|}{N_{\mathcal{H}} + N_{\mathcal{S}} + N_{\mathcal{P}}} \tag{32}$$

where $\mathcal{M}$ denotes the email type ($\mathcal{M} = \mathcal{H}$ for ham, $\mathcal{M} = \mathcal{S}$ for spam, and $\mathcal{M} = \mathcal{P}$ for phishing), and $N_{\mathcal{M}}$ denotes the number of email messages of type $\mathcal{M}$. Also, $|\mathcal{M} \to \mathcal{M}'|$ denotes the number of email messages of type $\mathcal{M}$ that are classified as $\mathcal{M}'$. It is necessary to note that in Dataset$_1$, $|\mathcal{S} \to \mathcal{H}|$ (false-negative event (miss)) occurrences are inexpensive mistakes, while $|\mathcal{H} \to \mathcal{S}|$ (false-positive event (false alarm)) is a more serious concern. However, in Dataset$_2$, both $|\mathcal{H} \to \mathcal{P}|$ and $|\mathcal{P} \to \mathcal{H}|$ incur the same cost. Hence, in Dataset$_1$, metrics that account for false positives, such as precision of UBEs, recall of ham emails, F1-measure, MCC score, AUROC, or AUPRC, serve to be more appropriate.

Precision: This metric computes the positive predictive value (reliability or worth of the UBE filter) by measuring the true positives and false positives. Precision aims at measuring the number of relevant results, i.e., what proportion of ham email identifications were actually ham in nature. For a given email type $\mathcal{M}$, it can be computed as:

$$\text{Precision}(\mathcal{M}) = \frac{|\mathcal{M} \to \mathcal{M}|}{|\mathcal{M} \to \mathcal{M}| + |\neg\mathcal{M} \to \mathcal{M}|} \tag{33}$$

The precision is computed for individual email types, and the overall precision is computed as the weighted average of the individual components as:

$$\text{Precision} = \frac{\text{Precision}(\mathcal{M}) \cdot N_{\mathcal{M}} + \text{Precision}(\neg\mathcal{M}) \cdot N_{\neg\mathcal{M}}}{N_{\mathcal{M}} + N_{\neg\mathcal{M}}} \tag{34}$$

Precision (of UBEs) is more appropriate in measuring the performance of Dataset$_1$, where false-positive events cost more than false-negative events. However, it is not very appropriate in measuring the performance of Dataset$_2$, where both false positives and negatives incur the same cost. Hence, we need metrics that incorporate both false positives and negatives, to obtain a generalized performance metric.

Recall: This metric evaluates the sensitivity (effectiveness of the UBE filter) by measuring the number of UBE messages that the filter succeeded in preventing from reaching the email inbox of the user. For a given email type $\mathcal{M}$, it can be computed as:

$$\text{Recall}(\mathcal{M}) = \frac{|\mathcal{M} \rightarrow \mathcal{M}|}{|\mathcal{M} \rightarrow \mathcal{M}| + |\mathcal{M} \rightarrow \neg\mathcal{M}|} \tag{35}$$

The recall is computed for individual email types, and is aggregated using Eq. 34. As discussed earlier, recall (of ham emails) is appropriate in measuring the performance of Dataset$_1$, while in Dataset$_2$, where false negatives are equally as important as false positives, recall is inappropriate.

F1 score: This metric seeks a balance between the precision and recall, and is interpreted as the weighted harmonic mean of the precision and recall. It differs from accuracy in the sense that, accuracy only accounts from true positives and negatives, while neglecting false positive and negatives. The F1 ($F_{(\beta=1)}$) score can be computed as:

$$F_{(\beta=1)} = (1 + \beta^2)\frac{\text{Precision} \cdot \text{Recall}}{(\beta^2 \cdot \text{Precision}) + \text{Recall}} \tag{36}$$

Since F1-measure uses both false positives and negatives by capturing precision and recall, it serves as a generalized metric for both Dataset$_1$ and Dataset$_2$. However, F1-measure does not account for the true negative occurrences (e.g., $|\mathcal{S} \rightarrow \mathcal{S}|$).

MCC score: This metrics serves as a balanced measure even in scenarios of class imbalanced data (such as in our case) by measuring the True and False Positives and Negatives (TP, TN, FP, and FN). The MCC score computes the essence of the correlation between the predicted and the observed classifications. The MCC score can be computed as:

$$\text{MCC} = \frac{\text{TP} \cdot \text{TN} - \text{FP} \cdot \text{FN}}{\sqrt{(\text{TP} + \text{FN}) \cdot (\text{TP} + \text{FP}) \cdot (\text{TN} + \text{FN}) \cdot (\text{TN} + \text{FP})}} \tag{37}$$

Since MCC score accounts for true and false positives and negatives, it serves as a more generalized metric than F1-measure, in evaluating the performance of the underlying machine learning approaches.

Area under the ROC curve (AUROC): The ROC probability curve is a graphical plot of sensitivity (Eq. 35) against fall-out (1-specificity, see Eq. 38). The AUROC metric measures the capability of a model to distinguish between classes. A greater value of AUROC indicates that the underlying UBE filter is able to distinguish between ham, spam, and phishing emails.

$$\text{Specificity}(\mathcal{M}) = \frac{|\neg\mathcal{M} \rightarrow \neg\mathcal{M}|}{|\neg\mathcal{M} \rightarrow \neg\mathcal{M}| + |\neg\mathcal{M} \rightarrow \mathcal{M}|} \tag{38}$$

Although AUROC effectively captures the hit and miss rates, it does not vary with the change in the ratio of the target classes, and hence is not very inferential in scenarios with imbalanced data.

Area under the precision-recall curve (AUPRC): The precision-recall curve is a graphical plot of precision (Equation 33) against the recall (Eq. 35). A higher value of AUPRC signifies that the underlying model minimizes the misclassifications and false alarms. When dealing with skewed datasets (such as in our case), the AUPRC reveals more informative insights concerning the performance of the underlying model, in comparison to AUROC (Saito and Rehmsmeier 2015).

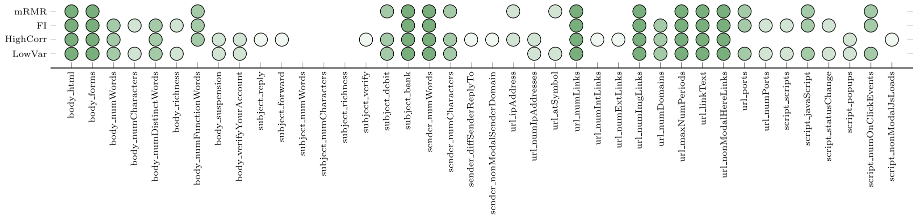

Fig. 2 A dotted heatmap mapping the occurrence frequency of the features (feature space in Table 4) in the utilized feature selection methods

6.2 Results and discussion

In this section, we report the results of our exhaustive experimentation on spam and phishing datasets in Table 6. Note that, Dataset$_3$ has the maximum number of samples and classes among the obtained datasets, and is hence utilized as the representative sample subject to feature selection (extraction). The features subspace obtained using Dataset$_3$ was then employed in Dataset$_1$ and Dataset$_2$, to facilitate accurate filtering of spam and phishing emails. Table 8 tabulates the performance of various machine learning algorithms (see Sect. 5) in the classification of spam emails of Dataset$_1$ using the email features obtained using feature selection (see Sect. 4.2) of the feature space of Dataset$_3$. Similarly, the performance of the machine learners on Dataset$_2$, using the features extracted from Dataset$_3$ is summarized in Table 9. It is important to point out that PCA facilitates feature extraction rather than feature selection, through a linear transformation of the input data. Table 10 shows the performance of the machine learning classifiers using PCA-transformed Dataset$_3$. From Tables 8, 9, and 10, it is interesting to note that the RF classifier consistently outperforms all other machine learners. Such superior performance can be attributed to the ability of RF to perform well and generalize even in the cases of disproportionate data through bagging and random input vectors. Additionally, we also remark that the features selected using FI-based feature selection (using RF) on Dataset$_3$, when classified using an RF classifier, outperforms the performance obtained using other feature selection approaches (98.4% accuracy and 99.8% AUPRC on Dataset$_1$, and 99.4% accuracy and 99.9% AUPRC on Dataset$_2$, see Tables 8 and 9)—FI (using RF) measures the usefulness of the features in the construction of the RF tree, and since the RF classifier is able to learn and generalize the underlying UBE data, it is only natural that FI (using RF) accounts for the highest performance.

From the analysis of the features selected by the utilized feature selection techniques, it can be noted that the features such as body_html, body_forms, subject_bank, sender_numWords, url_numLinks, url_numImgLinks, url_linkText, url_maxNumPeriods, and url_nonModalHereLinks, are selected by all feature selection techniques (LowVar, HighCorr, FI, and mRMR). However, certain features such as subject_numWords, subject_numCharacters, and subject_richness are never selected. Figure 2 depicts a dotted heatmap that captures the occurrence frequency of the features (feature space in Table 4) in the utilized feature selection techniques. It is worth understanding the occurrence frequency employed in Fig. 2 uses a naïve counting scheme, and a more advanced and informed decision concerning the *information of a feature* can be drawn using a weighted occurrence frequency scheme that accounts for the position of a feature in the ranked feature subspace (Gangavarapu and Patil 2019). Intuitively, the weighted

Table 8 Performance evaluation of various machine learning classifiers in the classification of spam emails (Dataset$_1$) using the email features obtained from the feature selection on Dataset$_3$

Feature selection	#Selected features (%)	Metric	Performance scores								Build time (s)
			NB	SVM	BDT	RF	ET	AB	SGB	VE	
None	40 (100%)	Accuracy	0.933	0.936	0.965	**0.982**	0.968	0.960	0.977	0.973	6.609
		Precision	0.932	0.938	0.967	**0.982**	0.960	0.959	0.977	0.973	
		Recall	0.933	0.936	0.968	**0.982**	0.960	0.960	0.977	0.973	
		F1-measure	0.932	0.930	0.967	**0.981**	0.960	0.959	0.976	0.972	
		MCC score	0.752	0.750	0.880	**0.932**	0.855	0.849	0.913	0.900	
		AUROC	0.967	0.814	0.991	**0.995**	0.926	0.982	0.989	0.924	
		AUPRC	0.975	0.889	0.992	**0.996**	0.943	0.987	0.992	0.953	
LowVar	27 (67.5%)	Accuracy	0.927	0.915	0.963	**0.980**	0.966	0.956	0.973	0.970	0.015
		Precision	0.927	0.918	0.962	**0.979**	0.965	0.955	0.973	0.970	
		Recall	0.927	0.915	0.963	**0.979**	0.966	0.956	0.973	0.970	
		F1-measure	0.927	0.904	0.962	**0.979**	0.966	0.954	0.973	0.969	
		MCC score	0.732	0.659	0.860	**0.923**	0.874	0.832	0.901	0.887	
		AUROC	0.692	0.751	0.987	**0.995**	0.930	0.984	0.990	0.915	
		AUPRC	0.971	0.854	0.990	**0.996**	0.948	0.987	0.992	0.948	
HighCorr	28 (70%)	Accuracy	0.941	0.931	0.963	**0.981**	0.962	0.955	0.966	0.964	2.033
		Precision	0.940	0.933	0.963	**0.981**	0.963	0.954	0.966	0.964	
		Recall	0.941	0.931	0.963	**0.981**	0.962	0.955	0.966	0.964	
		F1-measure	0.940	0.925	0.963	**0.981**	0.962	0.954	0.966	0.963	
		MCC score	0.780	0.730	0.864	**0.930**	0.863	0.830	0.874	0.864	
		AUROC	0.970	0.800	0.985	**0.995**	0.934	0.980	0.984	0.903	
		AUPRC	0.977	0.881	0.986	**0.996**	0.947	0.984	0.988	0.939	

Table 8 (continued)

Feature selection	#Selected features (%)	Metric	Performance scores								Build time (s)
			NB	SVM	BDT	RF	ET	AB	SGB	VE	
FI	21 (52.5%)	Accuracy	0.950	0.913	0.965	**0.984**	0.965	0.952	0.973	0.970	0.834
		Precision	0.948	0.916	0.964	**0.983**	0.965	0.951	0.973	0.970	
		Recall	0.950	0.913	0.965	**0.984**	0.965	0.952	0.973	0.970	
		F1-measure	0.948	0.901	0.964	**0.984**	0.965	0.951	0.973	0.969	
		MCC score	0.809	0.650	0.868	**0.936**	0.874	0.818	0.900	0.888	
		AUROC	0.974	0.745	0.989	**0.998**	0.939	0.981	0.988	0.917	
		AUPRC	0.980	0.851	0.990	**0.998**	0.951	0.985	0.991	0.948	
mRMR	17 (42.5%)	Accuracy	0.932	0.915	0.960	**0.972**	0.957	0.943	0.959	0.953	0.659
		Precision	0.929	0.920	0.958	**0.972**	0.957	0.942	0.958	0.954	
		Recall	0.932	0.915	0.959	**0.972**	0.956	0.943	0.959	0.953	
		F1-measure	0.929	0.904	0.958	**0.971**	0.956	0.940	0.958	0.951	
		MCC score	0.736	0.661	0.845	**0.895**	0.841	0.778	0.844	0.951	
		AUROC	0.944	0.749	0.978	**0.991**	0.925	0.966	0.971	0.871	
		AUPRC	0.964	0.854	0.984	**0.993**	0.939	0.977	0.982	0.920	
PCA	19 (47.5%)	Accuracy	0.917	0.900	0.956	**0.970**	0.938	0.941	0.948	0.947	1.499
		Precision	0.915	0.905	0.955	**0.968**	0.938	0.940	0.947	0.947	
		Recall	0.917	0.900	0.956	**0.968**	0.938	0.941	0.948	0.947	
		F1-measure	0.916	0.882	0.954	**0.968**	0.938	0.939	0.947	0.944	
		MCC score	0.691	0.586	0.832	**0.881**	0.774	0.775	0.804	0.797	
		AUROC	0.934	0.702	0.982	**0.992**	0.887	0.967	0.975	0.857	
		AUPRC	0.955	0.828	0.987	**0.994**	0.914	0.976	0.982	0.911	

Bold values indicate the highest recorded performance score across the chosen classifiers

Table 9 Performance evaluation of various machine learning classifiers in the classification of phishing emails (Dataset$_2$) using the email features obtained from the feature selection on Dataset$_3$

Feature selection	#Selected features (%)	Metric	Performance scores								Build time (s)
			NB	SVM	BDT	RF	ET	AB	SGB	VE	
None	40 (100%)	Accuracy	0.964	0.976	0.985	**0.991**	0.977	0.983	0.988	0.989	8.012
		Precision	0.964	0.976	0.985	**0.991**	0.977	0.984	0.988	0.989	
		Recall	0.964	0.976	0.985	**0.991**	0.977	0.984	0.988	0.989	
		F1-measure	0.964	0.976	0.985	**0.991**	0.977	0.983	0.988	0.989	
		MCC score	0.900	0.932	0.958	**0.976**	0.936	0.954	0.967	0.969	
		AUROC	0.987	0.958	0.995	**0.999**	0.966	0.996	0.998	0.979	
		AUPRC	0.988	0.961	0.995	**0.999**	0.965	0.997	0.998	0.982	
LowVar	27 (67.5%)	Accuracy	0.944	0.915	0.978	**0.992**	0.972	0.977	0.987	0.985	0.729
		Precision	0.944	0.915	0.977	**0.992**	0.972	0.977	0.964	0.985	
		Recall	0.944	0.915	0.978	**0.992**	0.972	0.977	0.964	0.985	
		F1-measure	0.944	0.915	0.977	**0.992**	0.972	0.977	0.964	0.985	
		MCC score	0.845	0.765	0.938	**0.978**	0.923	0.935	0.900	0.958	
		AUROC	0.983	0.883	0.996	**0.999**	0.957	0.997	0.900	0.972	
		AUPRC	0.984	0.883	0.996	**0.999**	0.957	0.996	0.988	0.975	
HighCorr	28 (70%)	Accuracy	0.967	0.976	0.983	**0.987**	0.976	0.973	0.980	0.982	2.906
		Precision	0.967	0.976	0.983	**0.987**	0.976	0.973	0.980	0.982	
		Recall	0.967	0.976	0.983	**0.987**	0.976	0.973	0.979	0.982	
		F1-measure	0.967	0.976	0.983	**0.987**	0.976	0.973	0.979	0.982	
		MCC score	0.909	0.933	0.953	**0.964**	0.933	0.925	0.941	0.950	
		AUROC	0.989	0.959	0.995	**0.998**	0.966	0.993	0.994	0.967	
		AUPRC	0.990	0.962	0.995	**0.998**	0.964	0.993	0.994	0.970	

Table 9 (continued)

Feature selection	#Selected features (%)	Metric	Performance scores								Build time (s)
			NB	SVM	BDT	RF	ET	AB	SGB	VE	
FI	21 (51.5%)	Accuracy	0.962	0.971	0.985	**0.994**	0.977	0.984	0.987	0.987	1.503
		Precision	0.962	0.971	0.985	**0.994**	0.977	0.984	0.987	0.987	
		Recall	0.962	0.971	0.985	**0.994**	0.977	0.984	0.987	0.987	
		F1-measure	0.962	0.971	0.985	**0.994**	0.977	0.984	0.987	0.987	
		MCC score	0.895	0.919	0.958	**0.980**	0.937	0.956	0.964	0.964	
		AUROC	0.986	0.955	0.995	**0.999**	0.966	0.996	0.997	0.976	
		AUPRC	0.988	0.956	0.995	**0.999**	0.966	0.997	0.998	0.979	
mRMR	17 (42.5%)	Accuracy	0.952	0.973	0.974	**0.980**	0.970	0.967	0.970	0.974	1.190
		Precision	0.952	0.973	0.974	**0.979**	0.970	0.967	0.970	0.974	
		Recall	0.952	0.973	0.974	**0.980**	0.970	0.967	0.970	0.974	
		F1-measure	0.951	0.973	0.974	**0.979**	0.970	0.967	0.970	0.973	
		MCC score	0.864	0.926	0.928	**0.942**	0.917	0.907	0.917	0.927	
		AUROC	0.978	0.956	0.990	**0.997**	0.959	0.987	0.987	0.955	
		AUPRC	0.981	0.958	0.992	**0.997**	0.956	0.989	0.990	0.958	
PCA	22 (55%)	Accuracy	0.943	0.926	0.968	**0.979**	0.956	0.963	0.970	0.970	2.276
		Precision	0.943	0.931	0.968	**0.979**	0.956	0.963	0.970	0.970	
		Recall	0.943	0.926	0.968	**0.979**	0.956	0.963	0.970	0.969	
		F1-measure	0.943	0.921	0.968	**0.979**	0.956	0.963	0.970	0.969	
		MCC score	0.843	0.791	0.911	**0.941**	0.879	0.896	0.917	0.915	
		AUROC	0.947	0.846	0.988	**0.995**	0.939	0.988	0.989	0.947	
		AUPRC	0.973	0.876	0.989	**0.996**	0.937	0.990	0.991	0.951	

Bold values indicate the highest recorded performance score across the chosen classifiers

Table 10 Performance evaluation of various machine learning classifiers in the classification of spam and phishing emails (Dataset$_3$) with and without PCA transformation

Feature selection	#Selected features (%)	Metric	Performance scores								Build time (s)
			NB	SVM	BDT	RF	ET	AB	SGB	VE	
None	40 (100%)	Accuracy	0.891	0.900	0.938	**0.962**	0.922	0.780	0.949	0.943	10.110
		Precision	0.886	0.885	0.936	**0.962**	0.922	0.779	0.948	0.943	
		Recall	0.891	0.889	0.938	**0.961**	0.922	0.779	0.949	0.943	
		F1-measure	0.887	0.875	0.936	**0.961**	0.922	0.779	0.948	0.940	
		MCC score	0.802	0.798	0.895	**0.934**	0.871	0.607	0.912	0.896	
		AUROC	0.972	0.888	0.989	**0.997**	0.936	0.915	0.992	0.933	
		AUPRC	0.942	0.819	0.973	**0.991**	0.879	0.820	0.981	0.899	
PCA	24 (60%)	Accuracy	0.852	0.840	0.903	**0.934**	0.866	0.771	0.905	0.900	4.814
		Precision	0.848	0.837	0.898	**0.933**	0.866	0.771	0.900	0.898	
		Recall	0.852	0.840	0.903	**0.934**	0.867	0.770	0.905	0.900	
		F1-measure	0.849	0.814	0.897	**0.931**	0.866	0.771	0.901	0.890	
		MCC score	0.735	0.681	0.828	**0.886**	0.765	0.582	0.829	0.811	
		AUROC	0.938	0.810	0.978	**0.990**	0.882	0.876	0.976	0.880	
		AUPRC	0.899	0.681	0.949	**0.975**	0.805	0.801	0.949	0.827	

Bold values indicate the highest recorded performance score across the chosen classifiers

occurrence frequency captures the importance of a feature f_i over $\{f_{i+1}, f_{i+2}, \ldots, f_{k-i+1}\}$ in the selected k-dimensional feature subspace.

Note the superior performance of various classifiers utilizing all the features in all the three datasets—this can be explained by the informative and discriminative capabilities of the chosen feature space with respect to the underlying email corpus. The effect of increasing dimensions on the classification time is shown in Fig. 3. From Fig. 3, it can be remarked that, with the increase in the dimensionality of the data, we observe an increase in the time taken to classify the email messages. It must be noted that the average build (training) time utilized in this paper (in Tables 8, 9, and 10, and in Fig. 3) is computed as the average of the runtime taken by all the eight utilized machine learning algorithms. It is worth mentioning that, the RF classifier is scalable with high-dimensional data, and several variants of the RF classifier that utilize the MapReduce algorithm further improve the scalability and efficiency of classification (Han et al. 2013). Since the RF classifier outperforms other machine learning approaches, the subsequent analysis is only presented with respect to RF classification. The effect of increasing dimensions on the classification performance with respect to various feature selection approaches is depicted from Figs. 4, 5, 6, 7, 8, 9 and 10. It can be remarked that the features selected using Dataset$_3$ model the data from the Dataset$_2$ better than that from Dataset$_1$. From Tables 8, 9, and 10, and from Figs. 4, 5, 6, 7, 8, 9 and 10, we observe that PCA indicates the lowest performance, in the case of all the datasets (Dataset$_1$ with 19 dimensions, Dataset$_2$ with 22 dimensions, and Dataset$_1$ with 24 dimensions). Such low performance can be attributed to the fact that PCA is an unsupervised feature extraction approach whose main objective is to maximize the variance. As explained earlier, the 'usefulness' and 'relevance' of a feature are not interchangeable, i.e., a relevant feature does not warrant usefulness and vice versa. Thus, the filters that only aim at maximizing the variance, often ignore the usefulness of the chosen features, which in turn impacts the classification performance. This fact is clearly corroborated by the lower performance of the LowVar filter in Dataset$_1$ with 27 dimensions.

7 Summary

Feature engineering and machine learning are indispensable in building any intelligent system. In this study, we surveyed various aspects of feature engineering in spam and phishing email detection. Moreover, we detailed various attempts by the researchers in mitigating the menace of UBE emails through the use of machine learning classifiers. In general, the volume of existing literature evaluated in this study corroborates the significant progress that has been and will be made in the field of spam and phishing email detection. In this research, we employed forty informative and discriminative content-based and body-based features that were selected in accordance with the underlying email corpus. First, we elucidated on the process of extraction of the discriminative feature space from the raw email corpus. Then, we leveraged five widely used prolific feature selection (extraction) approaches to engender an optimal feature subspace to improve the classification performance and eliminate the noise in the data. We presented an exhaustive comparative study through the use of several state-of-the-art machine learning classifiers to facilitate UBE filtering and classification. Furthermore, we explained the key performance indicators vital in the accurate assessment of the performance of the underlying UBE filters. We observed that the feature subspace determined by the FI-based feature selection approach (using RF),

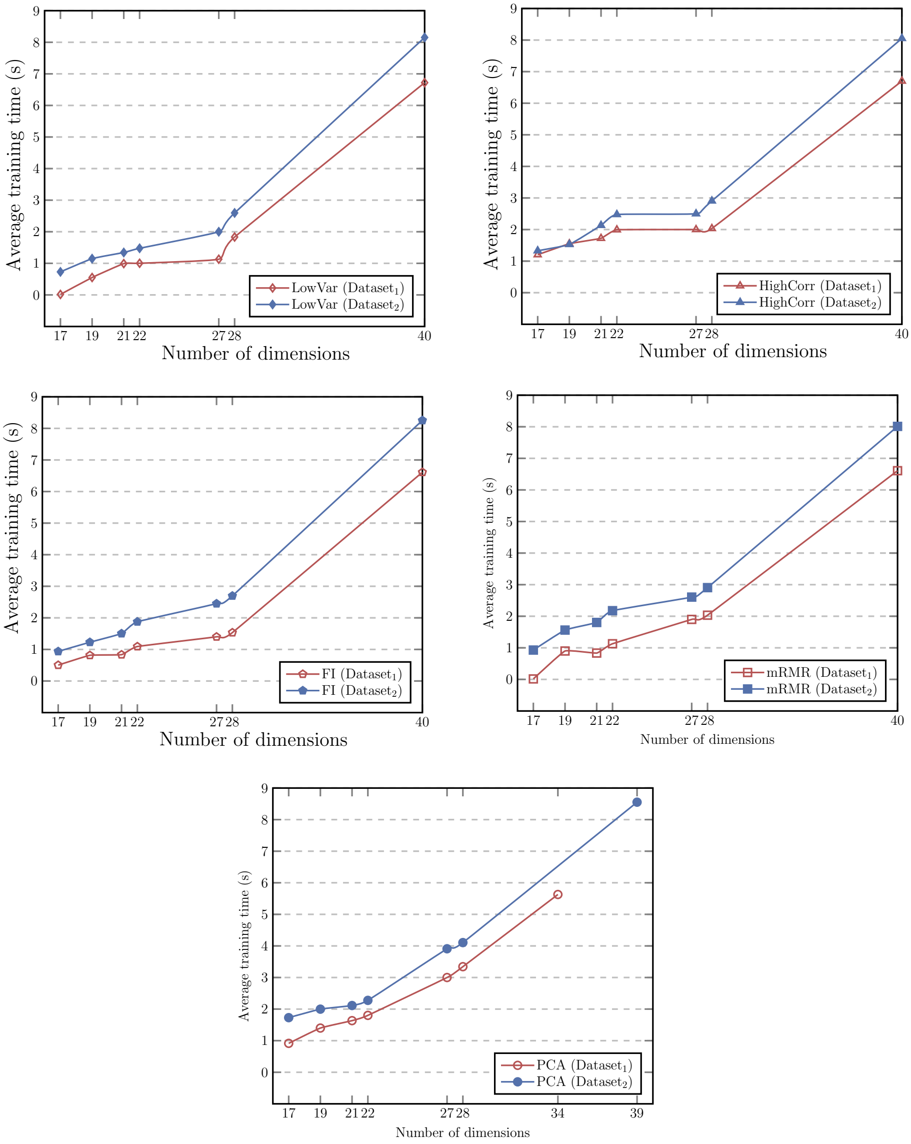

Fig. 3 The effect of increasing dimensions on the average training time

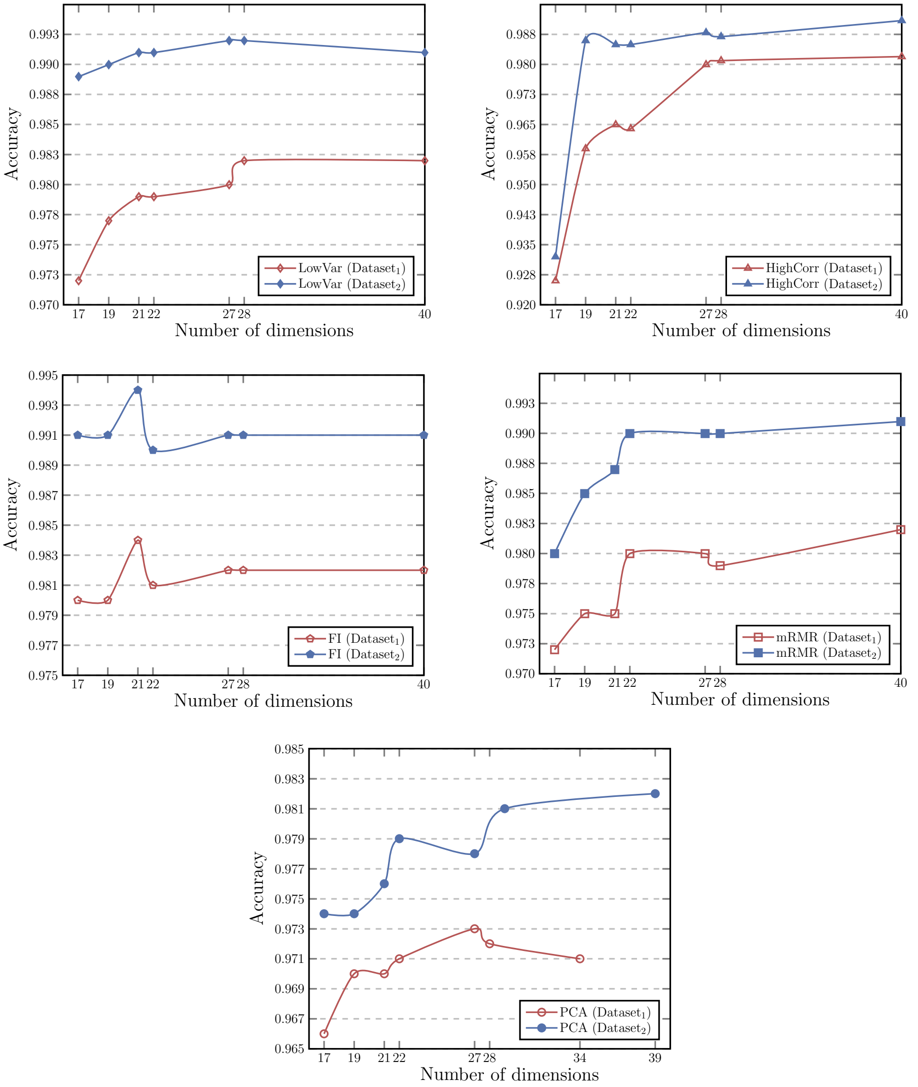

Fig. 4 The effect of increasing dimensions on the accuracy of the RF classification

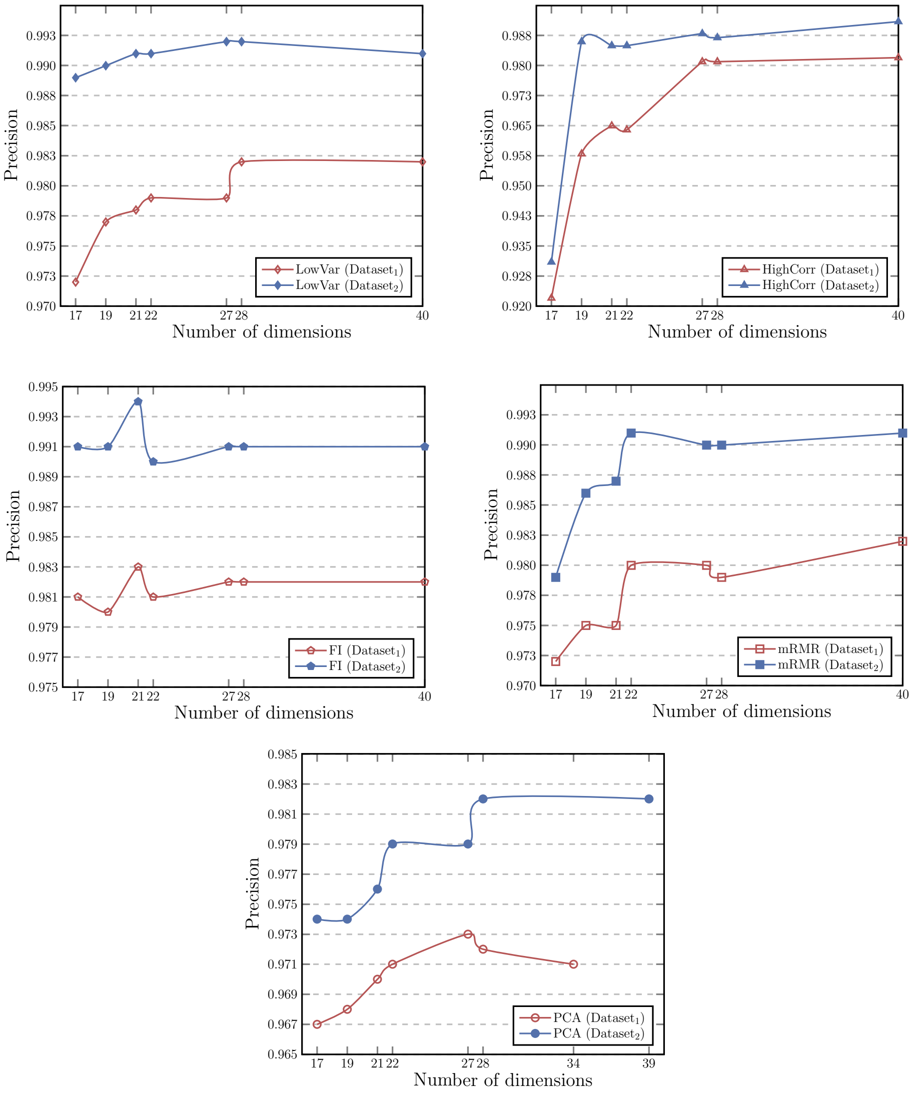

Fig. 5 The effect of increasing dimensions on the precision of the RF classification

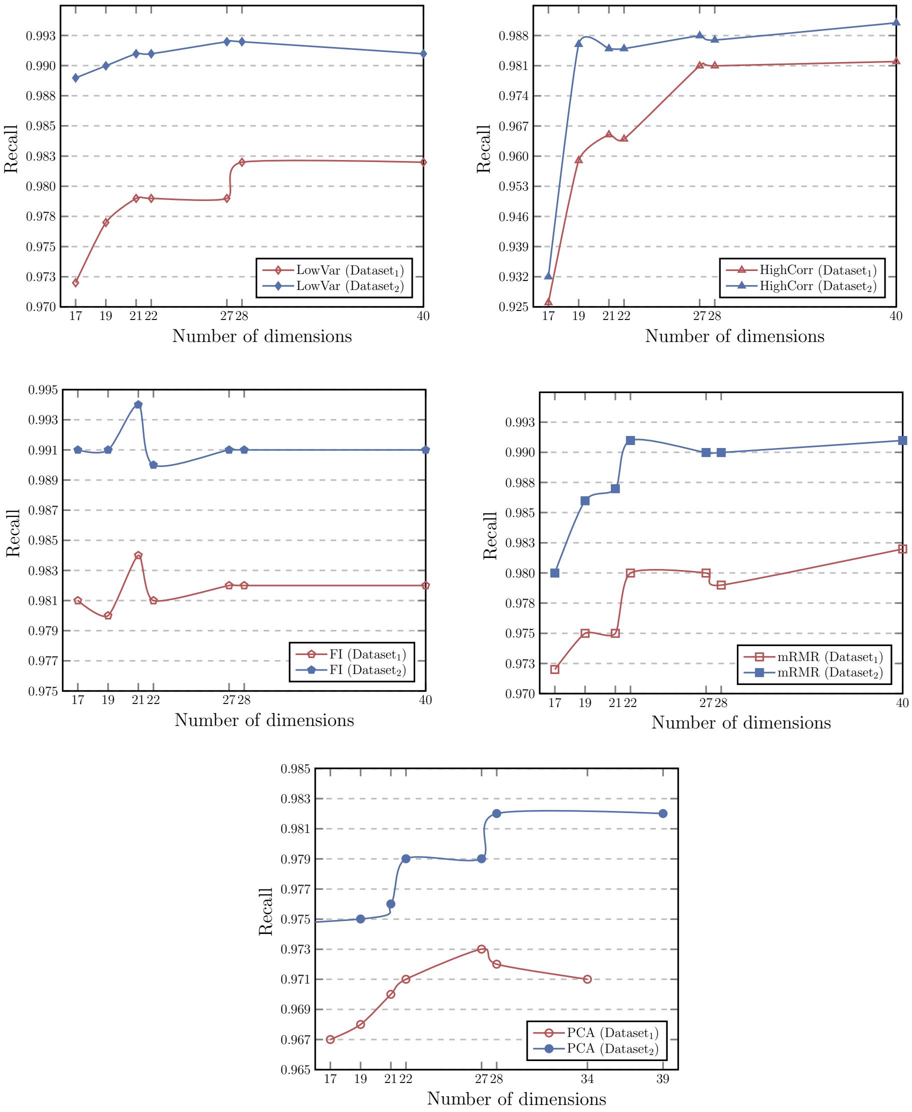

Fig. 6 The effect of increasing dimensions on the recall of the RF classification

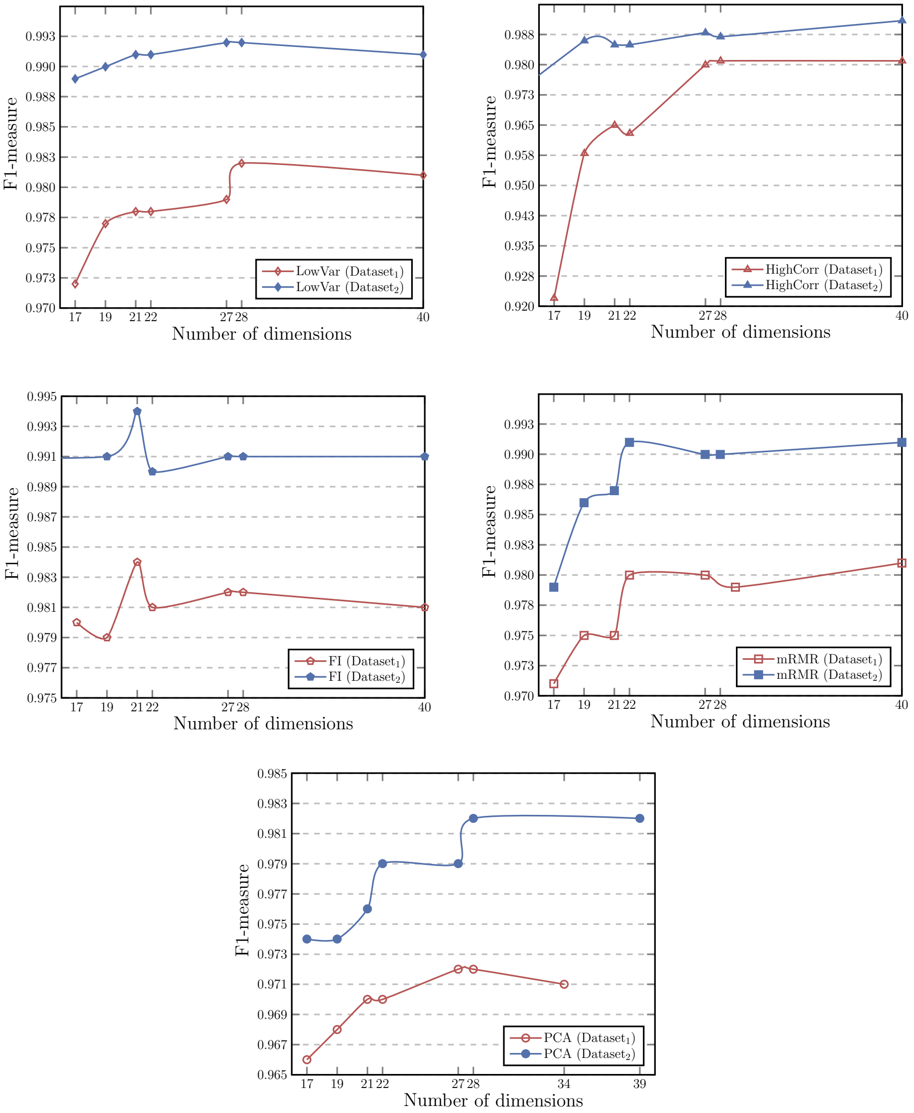

Fig. 7 The effect of increasing dimensions on the F1-measure of the RF classification

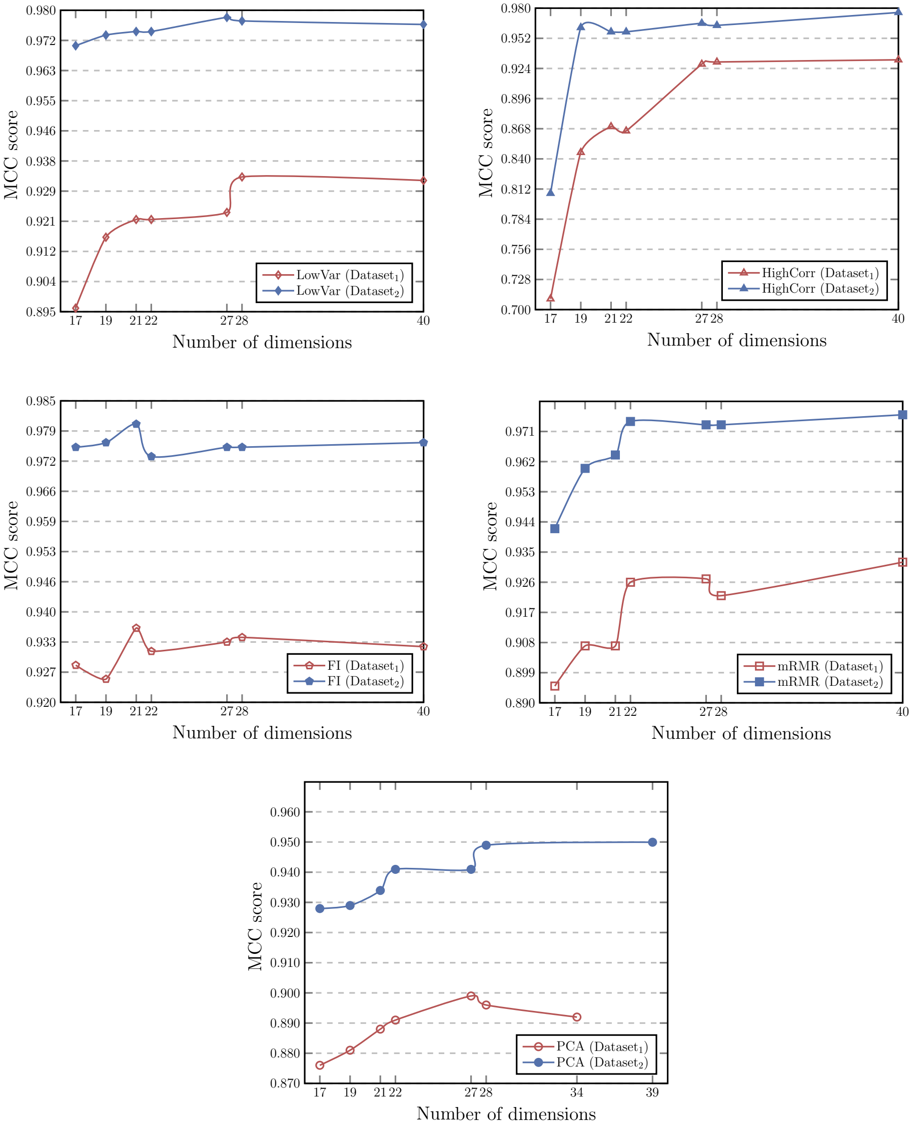

Fig. 8 The effect of increasing dimensions on the MCC score of the RF classification

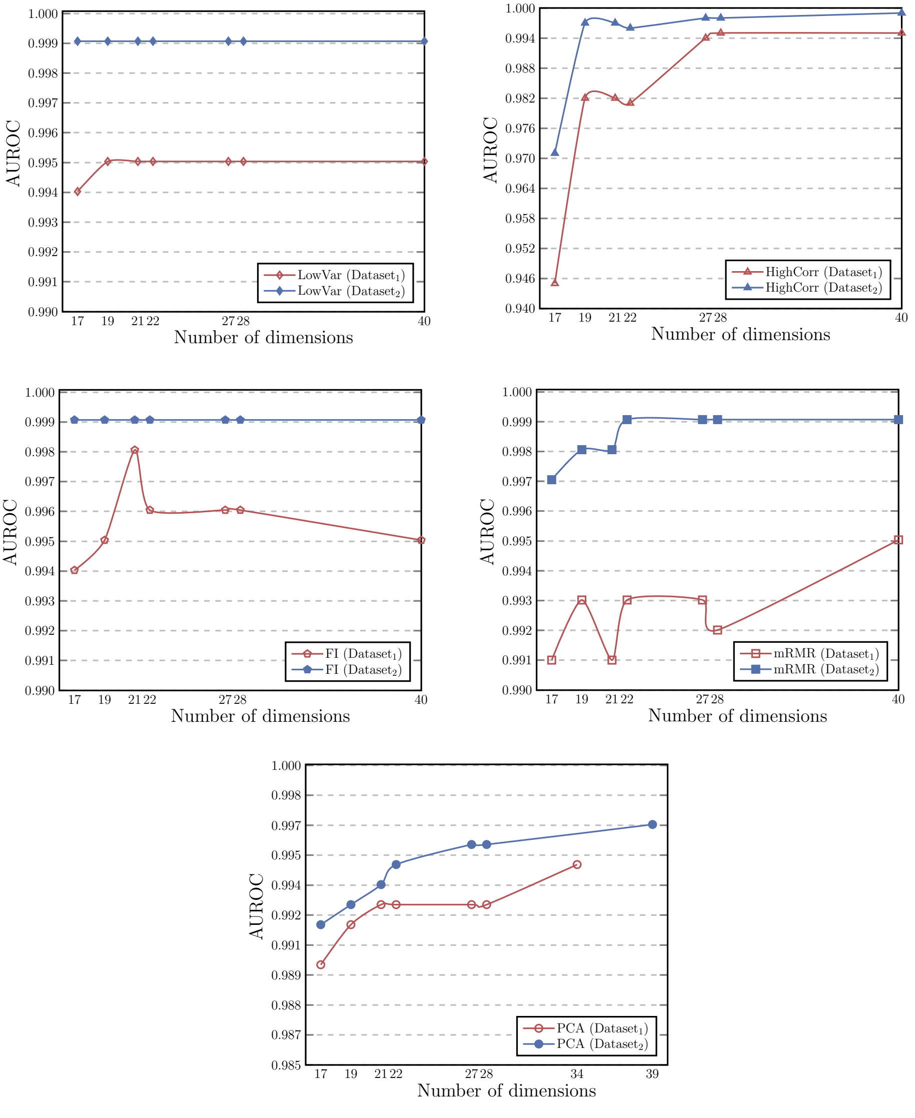

Fig. 9 The effect of increasing dimensions on the AUROC of the RF classification

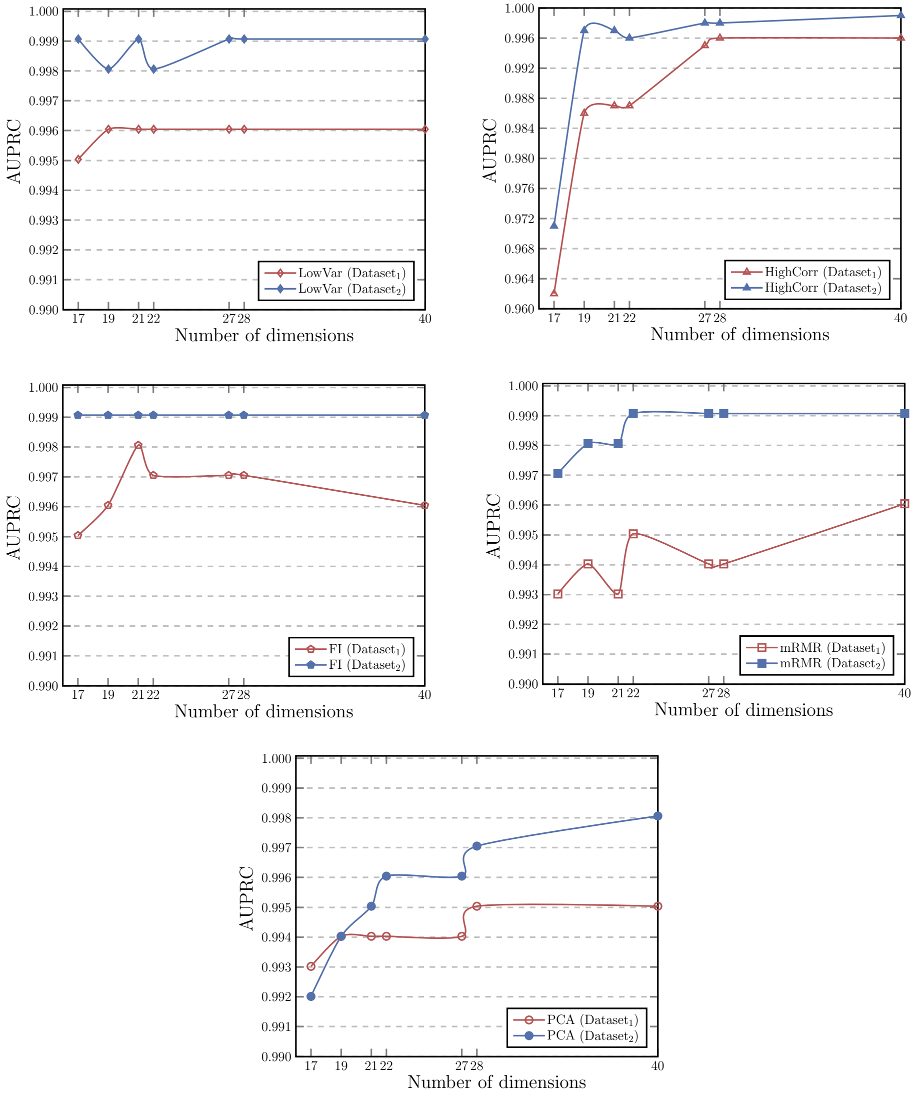

Fig. 10 The effect of increasing dimensions on the AUPRC of the RF classification

when classified using an RF classifier, resulted in an overall accuracy of 98.4% on ham−spam dataset (AUPRC of 99.8%) and 99.4% on ham−phishing dataset (AUPRC of 99.9%). Additionally, to enhance the understanding of the readers, we presented snippets of Python code, in-line with the text, enabling them to avail benefits from the existing email data.

Despite the extensive research in the field of UBE detection and filtering, certain issues need to be addressed. These issues include the lack of an effective strategy to handle security attacks on UBE filters, the inability of the current UBE filters to tackle concept drift phenomenon, and lack of effective UBE filters that utilize graphical features. In the future, we aim at improving the effectiveness of the proposed approaches by addressing the aforementioned open issues. Additionally, we also aim at exploring adversarial learning approaches to learn and adapt to the concept drifts effectively.

References

Abu-Nimeh S, Nappa D, Wang X, Nair S (2007) A comparison of machine learning techniques for phishing detection. In: Proceedings of the anti-phishing working groups 2nd annual eCrime researchers summit. ACM, pp 60–69 (2007)

Akinyelu AA, Adewumi AO (2014) Classification of phishing email using random forest machine learning technique. J Appl Math. https://doi.org/10.1155/2014/425731

Alkaht I, Al-Khatib B (2016) Filtering spam using several stages neural networks. Int Rev Comp Softw 11:2

Almeida TA, Yamakami A (2010) Content-based spam filtering. In: The 2010 international joint conference on neural networks (IJCNN). IEEE, pp 1–7 (2010)

Apruzzese G, Colajanni M, Ferretti L, Guido A, Marchetti M (2018) On the effectiveness of machine and deep learning for cyber security. In: 2018 10th International conference on cyber conflict (CyCon). IEEE, pp 371–390 (2018)

Auffarth B, López M, Cerquides J (2010) Comparison of redundancy and relevance measures for feature selection in tissue classification of ct images. In: Industrial conference on data mining. Springer, pp 248–262 (2010)

Awad W, ELseuofi S (2011) Machine learning methods for spam e-mail classification. Int J Comput Sci Inf Technol 3(1):173–184

Awad M, Foqaha M (2016) Email spam classification using hybrid approach of rbf neural network and particle swarm optimization. Int J Netw Secur Appl 8(4):17–28

Basnet RB, Sung AH (2010) Classifying phishing emails using confidence-weighted linear classifiers. In: International conference on information security and artificial intelligence (ISAI), pp 108–112 (2010)

Bec scams trends and themes (2019) Bec scams remain a billion-dollar enterprise, targeting 6k businesses monthly. https://www.symantec.com/blogs/threat-intelligence/bec-scams-trends-and-themes-2019. Accessed 07 May 2019

Bergholz A, De Beer J, Glahn S, Moens MF, Paaß G, Strobel S (2010) New filtering approaches for phishing email. J Comput Secur 18(1):7–35

Bhagyashri G, Pratap H, Patil D (2013) Auto e-mails classification using bayesian filter. Int J Adv Technol Eng Res 3(4)

Bhowmick A, Hazarika SM (2016) Machine learning for e-mail spam filtering: review, techniques and trends. arXiv preprint arXiv:1606.01042

Biggio B, Corona I, Fumera G, Giacinto G, Roli F (2011) Bagging classifiers for fighting poisoning attacks in adversarial classification tasks. In: International workshop on multiple classifier systems. Springer, pp 350–359 (2011)

Bolboaca SD, Jäntschi L (2006) Pearson versus spearman, kendall tau correlation analysis on structure–activity relationships of biologic active compounds. Leonardo J Sci 5(9):179–200

Breiman L (2002) Manual on setting up, using, and understanding random forests v3. 1. Statistics Department University of California, Berkeley, p 1

Breiman L (2001) Random forests. Mach Learn 45(1):5–32

Breiman L (2017) Classification and regression trees. Routledge, Abingdon

Chandrasekaran M, Narayanan K, Upadhyaya S (2006) Phishing email detection based on structural properties. In: NYS cyber security conference, vol 3. Albany, New York (2006)

Chanduka B, Gangavarapu T, Jaidhar CD (2018) A single program multiple data algorithm for feature selection. In: Abraham A, Cherukuri AK, Melin P, Gandhi N (eds) Intelligent systems design and applications. Springer, Cham, pp 662–672

Choudhary M, Dhaka V (2013) Automatic e-mails classification using genetic algorithm. In: Special conference issue: national conference on cloud computing and big data. Citeseer, pp 42–49 (2013)

Christina V, Karpagavalli S, Suganya G (2010) Email spam filtering using supervised machine learning techniques. Int J Comput Sci Eng 2:3126–3129

Cormack GV (2008) Email spam filtering: a systematic review. Found Trends Inf Retriev 1(4):335–455

Dhanaraj S, Karthikeyani V (2013) A study on e-mail image spam filtering techniques. In: 2013 International conference on pattern recognition, informatics and mobile engineering. IEEE, pp 49–55 (2013)

Dhanaraj KR, Palaniswami V (2014) Firefly and bayes classifier for email spam classification in a distributed environment. Aust J Basic Appl Sci 8(17):118–130

Díaz-Uriarte R, De Andres SA (2006) Gene selection and classification of microarray data using random forest. BMC Bioinform 7(1):3

Fette I, Sadeh N, Tomasic A (2007) Learning to detect phishing emails. In: Proceedings of the 16th international conference on world wide web. ACM, pp 649–656 (2007)

Gang S (2017) Email overload: research and statistics [with infographic]. https://blog.sanebox.com/2016/02/18/email-overload-research-statistics-sanebox/

Gangavarapu T, Patil N (2019) A novel filter-wrapper hybrid greedy ensemble approach optimized using the genetic algorithm to reduce the dimensionality of high-dimensional biomedical datasets. Appl Soft Comput. https://doi.org/10.1016/j.asoc.2019.10553

Gangavarapu T, Jayasimha A, Krishnan GS, Kamath SS (2019a) TAGS: towards automated classification of unstructured clinical nursing notes. In: Métais E, Meziane F, Vadera S, Sugumaran V, Saraee M (eds) Natural language processing and information systems. Springer, Cham, pp 195–207

Gangavarapu T, Jayasimha A, Krishnan GS, Kamath S (2019b) Predicting ICD-9 code groups with fuzzy similarity based supervised multi-label classification of unstructured clinical nursing notes. Knowl Based Syst. https://doi.org/10.1016/j.knosys.2019.105321

Gangavarapu T, Krishnan GS, Kamath S (2019c) Coherence-based modeling of clinical concepts inferred from heterogeneous clinical notes for icu patient risk stratification. In: Proceedings of the 23rd conference on computational natural language learning (CoNLL), pp 1012–1022 (2019)

Gansterer WN, Pölz D (2009) E-mail classification for phishing defense. In: European conference on information retrieval. Springer, pp 449–460 (2009)

Geurts P, Ernst D, Wehenkel L (2006) Extremely randomized trees. Mach Learn 63(1):3–42. https://doi.org/10.1007/s10994-006-6226-1

Guerra PHC, Guedes D, Meira JW, Hoepers C, Chaves M, Steding-Jessen K (2010) Exploring the spam arms race to characterize spam evolution. In: Proceedings of the 7th collaboration, electronic messaging, anti-abuse and spam conference (CEAS), Redmond

Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009) The weka data mining software: an update. ACM SIGKDD Explor Newsl 11(1):10–18

Hamid IRA, Abawajy JH (2014) An approach for profiling phishing activities. Comput Secur 45:27–41

Hand DJ (2007) Principles of data mining. Drug Saf 30(7):621–622

Han J, Liu Y, Sun X (2013) A scalable random forest algorithm based on mapreduce. In: 2013 IEEE 4th International conference on software engineering and service science. IEEE, pp 849–852 (2013)

Hassan D (2015) On determining the most effective subset of features for detecting phishing websites. Int J Comput Appl 122(20):0975–8887

Hassanpour R, Dogdu E, Choupani R, Goker O, Nazli N (2018) Phishing e-mail detection by using deep learning algorithms. In: Proceedings of the ACMSE 2018 conference. ACM, p 45

Hastie T, Rosset S, Zhu J, Zou H (2009) Multi-class adaboost. Stat Interface 2(3):349–360

Idris I, Abdulhamid SM (2014) An improved ais based e-mail classification technique for spam detection. arXiv preprint arXiv:1402.1242

Issac B, Jap WJ (2009) Implementing spam detection using bayesian and porter stemmer keyword stripping approaches. In: TENCON 2009-2009 IEEE region 10 conference. IEEE, pp 1–5 (2009)

Jayasimha A, Gangavarapu T, Kamath SS, Krishnan GS (2020) Deep neural learning for automated diagnostic code group prediction using unstructured nursing notes. In: Proceedings of the 7th ACM IKDD CoDS and 25th COMAD, pp 152–160 (2020)

Jiao J, Venkat K, Han Y, Weissman T (2015) Minimax estimation of functionals of discrete distributions. IEEE Trans Inf Theory 61(5):2835–2885

Karthika R, Visalakshi P (2015) A hybrid aco based feature selection method for email spam classification. WSEAS Trans Comput 14:171–177

Khonji M, Jones A, Iraqi Y (2011) A study of feature subset evaluators and feature subset searching methods for phishing classification. In: Proceedings of the 8th annual collaboration, electronic messaging, anti-abuse and spam conference. ACM, pp 135–144 (2011)

Klein D (2004) Lagrange multipliers without permanent scarring. University of California at Berkeley, Computer Science Division, Berkeley, pp 1–11

Kosinski M, Wang Y, Lakkaraju H, Leskovec J (2016) Mining big data to extract patterns and predict real-life outcomes. Psychol Methods 21(4):493

Kumar S, Arumugam S (2015) A probabilistic neural network based classification of spam mails using particle swarm optimization feature selection. Middle-East J Sci Res 23(5):874–879

Laorden C, Ugarte-Pedrero X, Santos I, Sanz B, Nieves J, Bringas PG (2014) Study on the effectiveness of anomaly detection for spam filtering. Inf Sci 277:421–444

Louppe G, Wehenkel L, Sutera A, Geurts P (2013) Understanding variable importances in forests of randomized trees. In: Advances in neural information processing systems, pp 431–439

Lueg CP (2005) From spam filtering to information retrieval and back: seeking conceptual foundations for spam filtering. Proc Am Soc Inf Sci Technol. https://doi.org/10.1002/meet.14504201146

Ma L, Yearwood J, Watters P (2009) Establishing phishing provenance using orthographic features. In: eCrime researchers summit, eCRIME'09. IEEE, pp 1–10

Mendez JR, Fdez-Riverola F, Diaz F, Iglesias EL, Corchado JM (2006) A comparative performance study of feature selection methods for the anti-spam filtering domain. In: Industrial conference on data mining. Springer, pp 106–120

Michalski RS, Carbonell JG, Mitchell TM (2013) Machine learning: an artificial intelligence approach. Springer, New York

Mohammad RM, Thabtah F, McCluskey L (2015) Phishing websites features. (Unpublished). http://eprints.hud.ac.uk/24330/6/RamiPhishing_Websites_Feature.pdf

Mousavi A, Ayremlou A (2011) Bayesian spam classifier. http://cs229.stanford.edu

Nagelkerke NJ et al (1991) A note on a general definition of the coefficient of determination. Biometrika 78(3):691–692

Nazario J (2018) Phishing corpus. https://drive.google.com/open?id=0B3rX15hRO_71Tl9iOHRkdlEwZVE. Accessed 12 Oct 2018

Netsanet S, Zhang J, Zheng D (2018) Bagged decision trees based scheme of microgrid protection using windowed fast fourier and wavelet transforms. Electronics 7(5):61

Norte Sosa J (2010) Spam classification using machine learning techniques-sinespam. Master's thesis, Universitat Politècnica de Catalunya

Ott M, Choi Y, Cardie C, Hancock JT (2011) Finding deceptive opinion spam by any stretch of the imagination. In: Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies, vol 1. Association for Computational Linguistics, pp 309–319

Palanisamy C, Kumaresan T, Varalakshmi S (2016) Combined techniques for detecting email spam using negative selection and particle swarm optimization. Int J Adv Res Trends Eng Technol 3(2)

Pan Y, Ding X (2006) Anomaly based web phishing page detection. In: Null. IEEE, pp 381–392 (2006)

Pearson K (1901) Liii on lines and planes of closest fit to systems of points in space. Lond Edinburgh Dublin Philos Mag J Sci 2(11):559–572

Pearson K (1920) Notes on the history of correlation. Biometrika 13(1):25–45

Pelletier L, Almhana J, Choulakian V (2004) Adaptive filtering of spam. In: Proceedingsm second annual conference on communication networks and services research. IEEE, pp 218–224 (2004)

Peng H, Long F, Ding C (2005) Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. IEEE Trans Pattern Anal Mach Intell 27(8):1226–1238

Rajamohana SP, Umamaheswari K, Abirami B (2017) Adaptive binary flower pollination algorithm for feature selection in review spam detection. In: 2017 International conference on innovations in green energy and healthcare technologies (IGEHT). IEEE, pp 1–4 (2017)

Renuka DK, Visalakshi P, Sankar T (2015) Improving e-mail spam classification using ant colony optimization algorithm. Int J Comput Appl 22–26

Rossi F, Lendasse A, François D, Wertz V, Verleysen M (2006) Mutual information for the selection of relevant variables in spectrometric nonlinear modelling. Chem Intell Lab Syst 80(2):215–226

Sah UK, Parmar N (2017) An approach for malicious spam detection in email with comparison of different classifiers

Saito T, Rehmsmeier M (2015) The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. PloS ONE 10(3):e0118432

Sakkis G, Androutsopoulos I, Paliouras G, Karkaletsis V, Spyropoulos CD, Stamatopoulos P (2001) Stacking classifiers for anti-spam filtering of e-mail. arXiv preprint cs/0106040

Sanz EP, Hidalgo JMG, Pérez JCC (2008) Email spam filtering. Adv Comput 74:45–114

Sculley D, Wachman GM (2007) Relaxed online svms for spam filtering. In: Proceedings of the 30th annual international ACM SIGIR conference on research and development in information retrieval. ACM, pp 415–422 (2007)

Shams R, Mercer RE (2013) Classifying spam emails using text and readability features. In: 2013 IEEE 13th international conference on data mining (ICDM). IEEE, pp 657–666

Sharma AK, Prajapat SK, Aslam M (2014) A comparative study between naïve bayes and neural network (mlp) classifier for spam email detection. Int J Comput Appl

Sharma A, Suryawanshi A (2016) A novel method for detecting spam email using knn classification with spearman correlation as distance measure. Int J Comput Appl 136(6):28–35

Shrivastava JN, Bindu MH (2013) E-mail classification using genetic algorithm with heuristic fitness function. Int J Comput Trends Technol 4(8):2956–2961

Silipo R, Adae I, Hart A, Berthold M (2014) Seven techniques for data dimensionality reduction. Report, KNIME. com AG. Accessed 12 Jan 2018

Symantec (2018) Internet security threat report. http://images.mktgassets.symantec.com/Web/Symantec/%7B3a70beb8-c55d-4516-98ed-1d0818a42661%7D_ISTR23_Main-FINAL-APR10.pdf?aid=elq_. Accessed 09 Mar 2018

Toolan F, Carthy J (2009) Phishing detection using classifier ensembles. In: eCrime researchers summit, eCRIME'09. IEEE, pp 1–9

Toolan F, Carthy J (2010) Feature selection for spam and phishing detection. In: eCrime researchers summit (eCrime). IEEE, pp 1–12 (2010)

Turner CR, Fuggetta A, Lavazza L, Wolf AL (1999) A conceptual basis for feature engineering. J Syst Softw 49(1):3–15

Tyagi A (2016) Content based spam classification-a deep learning approach. Ph.D. thesis, University of Calgary

Vergara JR, Estévez PA (2014) A review of feature selection methods based on mutual information. Neural Comput Appl 24(1):175–186

Vorobeychik Y, Kantarcioglu M (2018) Adversarial machine learning. Synth Lect Artifi Intell Mach Learn 12(3):1–169

Wang XL et al (2005) Learning to classify email: a survey. In: 2005 International conference on machine learning and cybernetics, vol 9. IEEE, pp 5716–5719 (2005)

Wang Z, Josephson WK, Lv Q, Charikar M, Li K (2007) Filtering image spam with near-duplicate detection. In: CEAS (2007)

Wold S, Esbensen K, Geladi P (1987) Principal component analysis. Chem Intell Lab Syst 2(1–3):37–52

Wu J, Deng T (2008) Research in anti-spam method based on bayesian filtering. In: 2008 IEEE pacific-asia workshop on computational intelligence and industrial application, vol 2. IEEE, pp 887–891 (2008)

Yang HH, Moody J (2000) Data visualization and feature selection: new algorithms for nongaussian data. In: Advances in neural information processing systems, pp 687–693 (2000)

Zavvar M, Rezaei M, Garavand S (2016) Email spam detection using combination of particle swarm optimization and artificial neural network and support vector machine. Int J Mod Educ Comput Sci 8(7):68

Zhuang W, Jiang Q, Xiong T (2012) An intelligent anti-phishing strategy model for phishing website detection. In: 32nd International conference on distributed computing systems workshops (ICDCSW). IEEE, pp 51–56

Zhang D, Yan Z, Jiang H, Kim T (2014) A domain-feature enhanced classification model for the detection of chinese phishing e-business websites. Inf Manag 51(7):845–853

Zhao W, Zhang Z (2005) An email classification model based on rough set theory. In: Proceedings of the 2005 international conference on active media technology. IEEE, pp 403–408

Zhong N, Liu J, Yao Y, Wu J, Lu S, Qin Y, Li K, Wah B (2006) Spam filtering and email-mediated applications. In: International workshop on web intelligence meets brain informatics. Springer, pp 1–31 (2006)