UNIVERSITY OF PIRAEUS
School of Information and
Communication Technologies

Department of Digital Systems

Hybrid Machine Learning Architecture for Phishing Email Classification

M.Sc. Digital Systems Security Thesis

Author: Koutroumpouchos Konstantinos

Supervisor: Prof. Xenakis Christos

Piraeus, February 2020

Abstract

In parallel to the increase of internet usage worldwide, phishing attacks incidents have increased analogically. One of the ways with which phishing attacks are initiated is through sending phishing emails. In this thesis an attempt at creating a novel technique for detecting phishing emails is presented. For that purpose, a solution using machine learning algorithms is explored. In order to design a machine learning model that properly identifies emails as phishing or benign, the following directions were studied: (a) The most widely used machine learning algorithms, with their representative parameters, advantages and disadvantages that their usage provides, (b) Developing an algorithm to test different combinations of machine learning feature inputs, (c) Different performance evaluation metrics of the created machine learning models and developing an algorithm for them and (d) The structure of the emails, their different characteristics and anything that can be used as a feature input for the machine learning models to be able to efficiently detect the pattern of phishing emails. To that end, two different fundamental feature categories were created: (i) Properties-based features, which are retrieved from the different characteristics of the emails, such as the number of URLs or attachments in the emails and (ii) Text-based features, which are retrieved from the text part of the email which is to be read by the receiver. Today, with the increase of computer processing power, efficient machine learning solutions are developed for an increasing number of problems. Although there is related work towards phishing email classification using machine learning techniques, this work proposes a novel hybrid technique using the two aforementioned types of features. To that end, two architectures were proposed: (a) A hybrid "assembled" architecture, which assembles the properties-based and the text-based features as one consistent feature vector which is then used as input for the classification process and (b) A hybrid "stacked" architecture, which has two classifiers: the first one classifies the email using text-based features and the second one (which outputs the final classification) uses the properties-based features and one additional feature which is the classification output of the first classifier for the to-be-classified email. The most widely used tools for developing machine learning based programs were explored and Apache Spark with its MLlib library were used. All the algorithms were written using the Python programming language. After developing an algorithm for testing every combination of classifier, their different hyperparameter values and the different architectures, it was found that, compared to the classification using only one type of features, the hybrid "assembled" architecture has an improved performance but the hybrid "stacked" architecture has a slightly reduced performance.

Keywords: machine learning, classification, hybrid, phishing, email

Table of Contents

List of Figures

List of Tables

1 Introduction

With the consistent and of exponential rate increase of internet activity and active users, it has helped people to connect with each other and have the whole information of the world in their personal computers and organizations to improve on their services and product distribution. However, where there is constant and consistent movement of capital, there is also interest from frauds and thieves. One of the ways that stealing of valuable information can be achieved is through phishing attacks. Phishing attacks are social engineering-based attacks and have increased in incident numbers with the increase of the internet users. One of the ways of initiating a phishing attack is through sending phishing emails. These are fraud emails which attempt to persuade the receiver that the email is from a trusted source but in reality they have links or code through which valuable information of the receiver is sent to the attacker.

This work proposes a hybrid machine learning based classifier for phishing email classification in order to reduce the victims of cyber-attacks from phishing emails. More specifically, there are two fundamental types of information which can be used as features for the machine learning classifier to "learn" the pattern of phishing emails from:

- The **properties** of the email, which are the characteristics of the email, such as the number of URLs or the number of attachments in the email.
- The **text** of the email, which includes the text of the email that is meant to be read by the receiver of the email.

These two types of inputs were used as features for machine learning classification, and two hybrid architectures are proposed:

- A hybrid **"assembled"** classifier: With this architecture it was attempted to use the features from the properties and the text of the email together by assembling them into one consistent feature vector. This vector will then be used as input for the machine learning classifier.
- A hybrid **"stacked"** classifier: With this architecture two classifiers are used for the classification of emails as phishing or legitimate. The first classifier uses the text-based features to learn the patterns of phishing and legitimate emails. The second classifier uses the properties-based features and an additional feature which is the prediction of the first classifier for the to-be-classified email.

As it will be shown in the thesis, the hybrid assembled classifier has an improved performance when compared to the "simple" classifiers which use only the properties-based or the text-based features, but the hybrid stacked classifier has a slightly worse performance than them.

The structure of this thesis is as follows:

Firstly, there is the theoretical background chapter, which consists of two parts. One is the theory part, where every theoretical information which was considered as necessary or helpful towards understanding the reason for building a tool for marking phishing emails is presented. This includes what a phishing attack is, historic information about phishing attacks, different types of phishing attacks, the structure and the different types of phishing emails and already established phishing countermeasures which include the classification of emails as phishing or legitimate using machine learning techniques. Since this thesis is focused on developing a machine learning based tool, afterwards there is information about machine learning in general including the types of machine learning algorithms, definitions relevant to machine learning, different metrics which are used for the performance evaluation of the built machine learning classifiers, and lastly some of the most widely used machine learning models with the advantages and disadvantages of their usage. The second part of the theoretical background includes related work of machine learning techniques for classification of phishing emails.

The next chapter is the problem statement, where the problem that is attempted to be solved with this thesis is briefly presented, as well as the direction that is followed towards the solution of that problem.

The fourth chapter is the approach chapter, where the problem statement is analyzed further, with information that is to be used for its solution. This chapter includes information about the structure of an email, which will help finding proper features/inputs of the machine learning classifiers and will result in lower error rates. Then, different file formats are presented. Since it is attempted to build a machine learning classifier, the first step towards building it is having enough data, or a data set of a large enough number of emails. This will ensure that the classifier will “learn” and be able to properly classify the emails as phishing or legitimate. For that reason, the different email file formats are presented, so that the email data sets that are found in the literature can be used properly, independently of their file formats. Afterwards the email data sets that were found in the literature and related research work are presented. This chapter then has different tools for building machine learning models with advantages and disadvantages and a few criteria for which one to choose. Apache Spark was chosen for building this work’s machine learning classifier, so information about it is presented afterwards. Lastly, a generic machine learning model building process is briefly described.

The next chapter is the implementation chapter, where the process of building the proposed classifiers is presented. As it is described there, there are two proposed hybrid architectures, which use in different ways features coming from the properties and the text of the email. The properties of the email consist different characteristics of it such as the number of the URLs or the number of attachments included in it. The tools used for the development of the model and the different processes of the model are presented. These include processes for creating the DataFrame which is used by Spark, properly cleaning the email text so that it can be used by the next processes, properly retrieving both the properties-based and the text-based features, as well as the proposed architectures and the process of testing the different classifiers, their parameters, the different feature types and the two architectures.

The next chapter includes the results of the implementation. This includes performance evaluation metrics of the different classifiers, graphs for better comparison and processing times required for building the different classifiers.

The following chapter includes proposed ideas as potential future work which could be based on this work.

The last chapter is the conclusion which includes what has been and what has not been achieved by this work.

2 Theoretical Background

2.1 Phishing

Phishing is a fraudulent act of stealing personal and sensitive information from a user. As the name suggests, Phishing as a word originates from the actual "Fishing" in the water, where a bait is used to trick the fish into biting the hook and getting caught. The "Ph" was first found in the first hackers who were called "Phreaks". Phishing as a cyber-attack is executed through internet and mobile technologies, such as email, instant messaging applications, SMS and Social Networking applications.

Phishing employs both social engineering and technical subterfuge to succeed as an attack. Social engineering attempts to fool the unsuspecting user into believing that the message they have received (email, SMS etc.) or the website they have visited is from a trusted and legitimate organization/source. On the other hand, technical subterfuge techniques involve installing malware directly onto the victims' computers in order to directly steal the personal information as well as blocking the user from being able to access it in an attempt of a ransomware.

As mentioned before, the purpose of a phishing attack is to steal personal information from the unsuspecting user, such as passwords and credentials from banking applications for financial gain, or credentials from social applications for a type of ransomware, where the attacker asks for money in order to give the account back to the user. Other benefits that a successful phishing attack may bring include impersonating the identity of the victim in order to execute other illegal activities and gaining the fame and the dignity as criminals who have succeeded in stealing certain sensitive information or getting into a certain system.

The first known phishing attack on a financial operator was in 2001 against e-gold [1]. According to the Anti-Phishing Working Group (APWG) [2] and its latest report for Phishing Activity Trends, the total number of phishing sites detected by APWG in the third quarter of 2019 was 266,387. This was up 46 percent from the 182,465 seen in Q2 and almost double the 138,328 seen in Q4 2018. The total number of unique phishing reports received according to APWG, were 173,063 in 2005 and 1,040,654 in 2018. Note that these numbers represent phishing reports, with each one resulting in a multitude of actual phishing attacks.

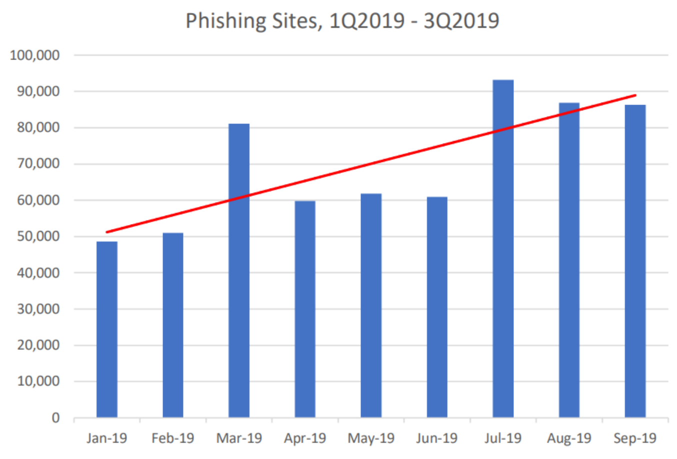


Figure 1 APWG Phishing Activity Trends for 2019

The top targeted industry sectors in the third quarter of 2019 are reported to be the following.

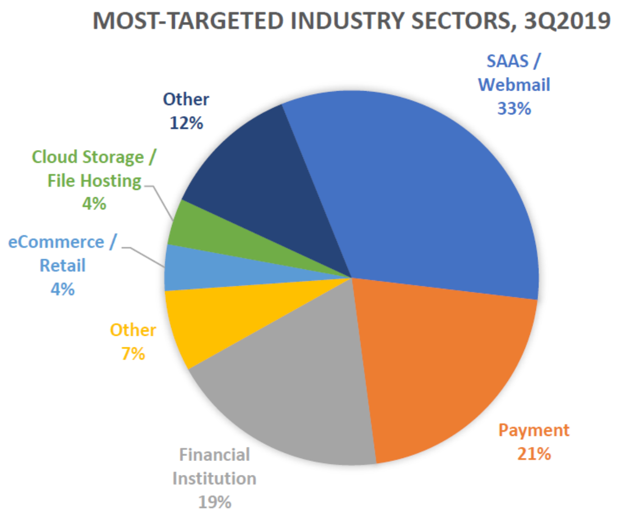


Figure 2 Most-Targeted Industry Sectors by Phishing Attacks

2.1.1 Phishing Techniques

The different types of phishing attacks are the following [3].

Spear Phishing: The traditional phishing attack has the logic of attacking as many targets as possible, by mass-sending the masqueraded message to them. Spear Phishing, on the other hand, attempts to learn as much information as possible about one or a few targets, in order to forge a much more legitimate-looking message. **Whaling** is a subcategory of spear phishing targeting the "whales" or senior and upper management positions in organizations.

Vishing (Voice Phishing): The attacker calls the victim on the phone and imitates to be from a legitimate organization. They then request personal information in order to "verify" that the user is the correct one, in an attempt to steal that information and then use it to their advantage.

Smishing (SMS Phishing): This type of phishing is executed using the Short Message Service (SMS). Following the principles of email phishing, the attacker sends a seemingly legitimate SMS from a trusted organization and requests the victim to provide with certain information. With the addition of smartphones in the last decade, these phishing SMS messages may also include URLs leading to malicious websites.

Email/Spam: One of the most widely used methods is sending a phishing email to many users with a request of personal information, such as credentials to a banking application. **Clone phishing** is a subcategory of email phishing, where a previously delivered email from a trusted source is cloned and malicious URLs or attachments are inserted into it and it is resent.

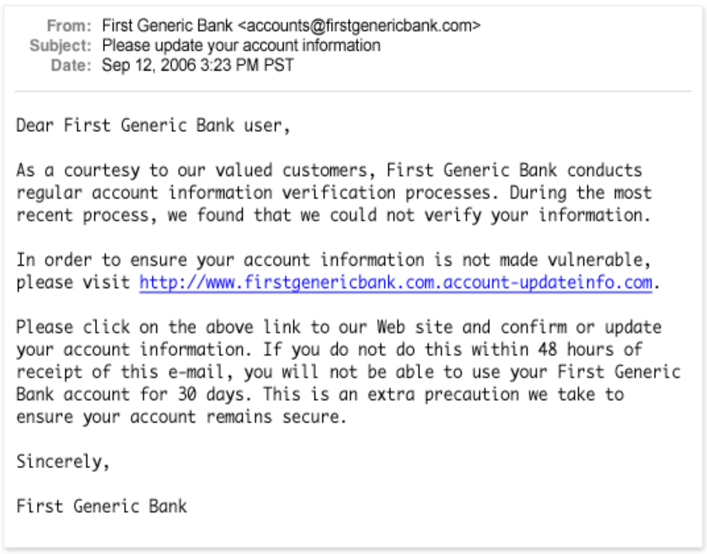
2.1.2 Lifecycle of phishing attack

The lifecycle of a phishing attack consists of the following steps:

1. **Planning and Setup**: This step includes the attacker collecting information about the target or targets, by identifying the organization's characteristics and anything that could help the attacker in executing a successful attack.
2. **Construction**: In order to bait the victims into believing that the website they are visiting is legitimate, the attacker creates a website that is a clone of the legitimate website that is to be exploited. The only difference is the functionality when entering sensitive information, for example when the victim tries to enter their banking credentials in order to log in. While at the legitimate website the result is that the account information is provided, at the phishing website, the credentials are stored and sent to the attacker. This attack could also include having the phishing website ask for additional information such as location, secret questions etc. in order to steal as much as possible. In order for the victim to be baited into giving the additional information, the trick could be that this information is needed in order to verify that it is the actual user.
3. **Spreading**: The attacker then chooses the method of phishing distribution (email, SMS, instant messaging etc.) and starts mass-sending phishing messages which include malicious links.
4. **Installation**: The fake link redirects to the phishing website. By clicking the link, there might also be an attempt to install malicious software into the victim's system, whether that is a smartphone or a personal computer.
5. **Data Collection**: The attacker receives sensitive information from the victims who are entering it through the malicious website and by any software that is installed and attempts to automatically send as much sensitive information as it can find.
6. **Break-Out**: The last step is for the attacker to remove all the evidence. This includes shutting down the phishing website's server, deleting any messaging/email account that was used to send the fake link and even getting to more extreme measures depending on the level of the attack (e.g. destroying the computers which were used to execute the attack).

2.2 Phishing Email

A phishing email is (as the name suggests) an email that is disguised so that it looks like it is coming from a trusted source. For example, it could be disguised as an email from a bank or an online store. Such emails are created by malicious users with the purpose of obtaining sensitive information from the receiver. The way this works is by inserting in the phishing email a URL which leads to a malicious website which, in its simplest form, will ask the potential victim to enter some sort of personal / sensitive information, such as their bank account credentials, credit card number, social security number etc. An easy and representative example of a phishing email is shown in the image below.



From: First Generic Bank <accounts@firstgenericbank.com>
Subject: Please update your account information
Date: Sep 12, 2006 3:23 PM PST

Dear First Generic Bank user,

As a courtesy to our valued customers, First Generic Bank conducts regular account information verification processes. During the most recent process, we found that we could not verify your information.

In order to ensure your account information is not made vulnerable, please visit http://www.firstgenericbank.com.account-updateinfo.com.

Please click on the above link to our Web site and confirm or update your account information. If you do not do this within 48 hours of receipt of this e-mail, you will not be able to use your First Generic Bank account for 30 days. This is an extra precaution we take to ensure your account remains secure.

Sincerely,

First Generic Bank

Figure 3 Phishing Email Example

There are a few characteristics which a phishing email has and can be used to help a user detect if they are a victim of a phishing attempt. A few of these characteristics are the following:

- **Generic Greeting**: As can be seen from the image of the phishing email example, there is a generic greeting, meaning that there is not a specific name of the receiver. Usually there is a "flat" greeting, like that one in the figure, where there is no variable name inserted, and another possibility is that as a name there is the local-part of the receiver's address (the part of the email name before the @ sign).
- **Reply address different than the claimed sender**: In an attempt to impersonate a trusted and legitimate source, the attacker will forge the sender name and change it into the trusted organization's name and hide the actual sender address [4].
- **Forged Link**: At the links provided in the email, even if they seem to lead to a safe and well-known address, the text shown could be different than the actual address that is to be visited if the link is clicked. That is handled by inserting HTML elements into the email. For example, the following HTML code will result in a text of the trusted yahoo website but clicking the link will lead to the phishing website.
  <a href=www.phishingwebsite.com> https://www.yahoo.com </a>
- **Domain Names**: In general, when an email is actually coming from a trusted source such as a bank, if there are any domains included in URLs in the email, they should be from the domain of the bank. Otherwise, the email becomes suspicious.
- **Number of URLs**: Trusted sources generally hesitate to include links in their emails, because, as shown at the previous bullets, links are not a very trustworthy way of visiting addresses. If there is a need to visit a website, it is preferred to ask the user to visit the website on their own, instead of providing the link.
- **Using URL redirecting services**: In order to hide the actual website that is to be visited by clicking on the link, the attacker could compress it into a smaller URL using services such as TinyURL [5]. That way the visited website's domain name is seen only after visiting it, which might lead into the user not checking it for legitimacy.
- **SSL Certificates are not safe**: According to the latest APWG's report on phishing trends [2], and their contributor PhishLabs, in the third quarter of 2019, 68 percent of all phishing sites

were using SSL certificates. That was up from 54 percent the prior quarter. This is a clear indicator that users cannot rely on SSL certificates alone to indicate that a website is safe.

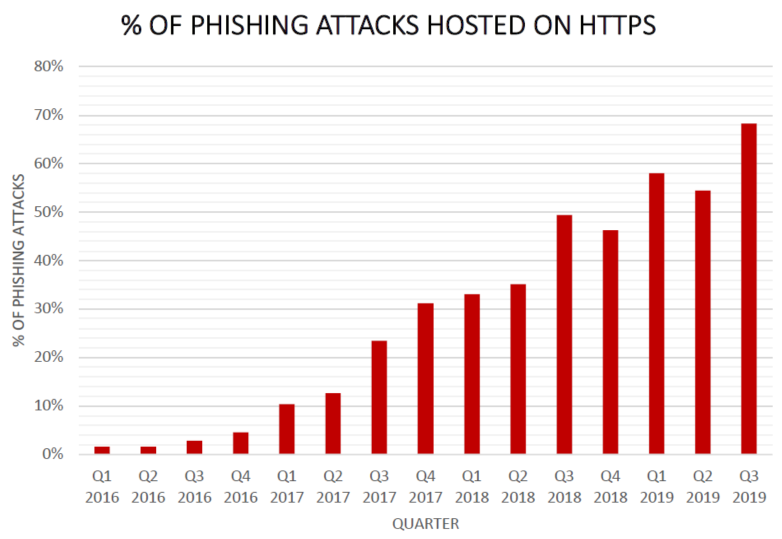


Figure 4 Phishing Attacks Hosted on HTTPS

- **Using the @ symbol**: When visiting a website and the URL has the @ symbol in it, the actual website that is visited is the one after the @ symbol. That way the attacker can have a trusted website before the @ symbol and then add the phishing website after it.
- **Using hexadecimal character codes**: When writing a URL, one could either use the ASCII characters which are the normal characters that are most widely used, or their hexadecimal representations can be used. For every character there is an equivalent hexadecimal number. This can be exploited by the attacker and they can include characters which cannot be easily read by the unsuspecting user.
- **Requesting information**: A legitimate email should never request any personal information. If there is such an attempt in the email, this is a clear indicator that it is a phishing attempt.
- **Sense of Urgency**: It is generally wanted by the malicious users to get as much information as possible before their malicious website is either taken down or marked as malicious by popular agencies (also known as becoming “blacklisted”). This reason, in addition to adding a psychological factor in order to persuade the victim, results in the phishing emails having a sense of urgency in their text. For example, the email could mention that there is a limited time window to provide the information or otherwise there might be consequences such as termination of the victim’s account.

The ones mentioned above are a few characteristics of a phishing email which, if a user goes through every email, they can be used as tools to detect an email as phishing or legitimate.

When it comes to the content of the phishing email, or the way that the attacker attempts to convince the victim that it is a legitimate source, there are different types of phishing emails [6].

1. <u>The Government Maneuver</u>: In this phishing email the attacker is trying to impersonate a federal body, in an attempt to intimidate the victim and persuade them in giving any information.
2. <u>The Friend Tactic</u>: The attacker, in this case, impersonates a friend of the victim. While the attacker most likely does not have any relevant information to impersonate the victim’s

friend properly (unless there is a spear phishing attack), they try to use a generic friend and count on the chance that the victim will have a friend who would send such an email.

3. The Billing Problem: This phishing email states that a recent online purchase could not be completed for different reasons (credit card expired, incorrect billing address etc.) and it is required to visit the website provided for the problem to be dealt with. The link provided leads to a phishing website.
4. The Expiration Date: With this type of phishing email there is a statement that the victim's account with a certain service or company is about to expire. However, the expiration can be extended by visiting the phishing website provided in the email and providing the credentials.
5. The Virus or Compromised Account Scare: In this case, the attacker attempts to scare the victim by stating that their computer or phone has been compromised or an account of theirs has been breached. In order for the problem to be solved, it is asked to visit the phishing website or download a malicious attachment.
6. The Contest Winner: This is an attempt to persuade the victim into thinking that they have won a prize. In order to receive the prize, they have to visit a website and enter certain information such as location, which is then used against them.
7. The Friendly Bank: Banks offer account notifications when certain amount of money has been withdrawn. This phishing email impersonates a bank and they mention that the withdraw cap has been exceeded, and for verification purposes certain pieces of information need to be entered.
8. The Victim: The attacker in this case impersonated a customer who has return a certain amount of money to the supposedly company which happens to be the victim. There is a threat involved of involving authorities if there is no answer.
9. The Tax Communication: This phishing email mentions that the victim is either enabled to receive a tax refund or has been selected to be audited. In both cases there is a requirement to provide with certain sensitive information.
10. The Checkup: In this case, a company is supposedly having a security checkup and ask the victim to provide their personal credentials for the checkup to be completed.

2.3 Phishing Countermeasures

There is already a plethora of proposed phishing countermeasures [7]. In the following chapter a description of each type of countermeasure is presented.

User Education

Both cyber security experts and cybercriminals make an effort to be one step ahead of one another. This results in both of the parties being constantly educated with every new technology and vulnerability that appears and are ready to either use it and secure it or exploit it respectively.

While the cyber experts are quickly aware of any change that happens in the cyber security sector, the regular user is their base target. And the regular user is the one that will eventually be a victim of a phishing attack. Even though there might be vulnerabilities and exploits which happen because of a technical imperfection, the human is the weakest link to the cyber security chain.

The fact that the average user is the reason that phishing attacks succeed (because they are the ones that will have to click on the phishing website link) results in a new relation of attacker-defender, where the attackers now also educate themselves in way to be more persuasive to that user. On the defender side, in order for the users to be able to recognize phishing attempts, there needs to be relevant education. There are already established education programs and services [8] [9]. The user education is very effective as a countermeasure because enhancement is provided to the base of the problem (the human factor from which the attacks start).

Machine Learning Techniques

Machine learning is the process of training a classifier by inputting an adequate number of examples. The examples are related to the classification that is desired. For example, if a phishing email classifier is to be designed, which will be able to determine whether an email is phishing or not, then the examples will be emails hand-classified as phishing or legitimate. The result is a trained classifier with the use of which it is possible to detect (classify) new examples for which a classification has not been done yet. The proposed tool in this thesis is based on Machine Learning Techniques. More details will be provided in the related work, the approach and implementation chapters.

Search engine based techniques

The idea of this way of determining a website as phishing or not is that certain keywords and pieces of information (images) are extracted from the website and are queried into search engines. If the domain name of the organization is returned within the first certain number of search results, then the website can be considered as legitimate. However, if the to-be-determined website is a phishing one, then the domain will not appear within these first results. The drawback to this approach is that there is a high false positive / alarm rate because newly created legitimate websites have a high chance of not being high in the search results.

List based anti-phishing solutions

This type of protection approach used blacklists and whitelists in order to protect against visiting a phishing website. A blacklist is a list of websites which have been identified and confirmed as phishing websites. On the contrary, a whitelist is a list of websites which have been identified as legitimate websites.

The advantage of the list based approach is that the true positive (correct alarm) when using a blacklist approach is 100 percent, because every website which is determined as phishing will be in the blacklist which means that it has been confirmed as phishing. When using a whitelist, the true negative rate is a 100 percent, because every website which will be determined as legitimate will actually be included in the whitelist, so it is confirmed as legitimate. On the other hand, these methods have their drawbacks. Firstly, the maintenance of the lists is a service in itself, meaning that it requires processing power. Secondly, while the aforementioned alarms are 100 percent correct, the false negative rate for the blacklists and the false positive rate for the whitelists will be significant. That is because, for the blacklists, if there is a phishing website which has not yet been included in the blacklist it will be wrongfully categorized as a legitimate one. With whitelisting, a website that is not in the whitelist will be categorized as phishing, regardless if it actually is a phishing or a legitimate one.

There are list based services which provide the ability to use for applications and normal use (Google Safe Browsing [10], PhishTank [11])

Visual similarity based approaches

This technique employs visual features to compare a webpage with verified authenticated webpages. The visual features include textual content, text formatting, presence and usage of CSS, screenshots of the website and images included in it. The base of this approach is the assumption that two pages cannot have similar looks, so that when a website is compare based on these visual features and it ends up being similar to another verified webpage, then it can be considered that this is an attempt to make a clone for a phishing website.

2.4 Machine Learning

The proposed solution towards detecting phishing emails uses Machine Learning technologies to help in the classification process. In this chapter there will be a description of the machine learning fundamentals, the types of machine learning techniques and the different most used Machine Learning classifiers found in the literature.

Machine Learning is considered to be a subset of the studies of Artificial Intelligence. Machine Learning consists of algorithms which are designed to be used by computers in order to complete certain tasks without a specific set of commands, but rather by detecting patterns from given examples.

The reason that machine learning has been successful in the recent years is the following. The machine learning algorithms generally require a lot of processing power because they rely not on just following specific commands and executing them, but rather on iterating over a large number of examples and adapting certain parameters and weights in order to be able to detect a certain pattern based on these examples. The end result should be a model/classifier which has their weights adjusted so that they are able to "generalize" and identify new examples (which have not yet been classified) with a minimal error. The processing power of computers has increased significantly, making the whole process more feasible [12].

Machine learning as a programming and problem solving technology is useful for the following reasons:

- **Less time programming**: When designing a traditional algorithm and program, the developer needs to take into account every possible case that might occur and provide a piece of code to solve them. However, with the use of a machine learning algorithm, the only basic requirement is to collect or find and already built data set with the desired examples and use them against the classifier for it to be able to train for them. The only additional requirement is the tuning of additional hyperparameters that the machine learning algorithm might have.
- **Customization and scaling of final products**: Because the machine learning algorithms are trained against certain examples, a developed design which is design for a specific type of examples could be easily adjusted for another set of examples just by using the machine learning algorithm from scratch and passing the new examples through it. For example, if a program is made for grammar or text based functionality, it can be instantly transported to another language by getting enough data examples in that language.
- **Solution to previously unsolvable problems**: With the addition of machine learning, a new direction towards problems like face recognition and voice recognition is provided.

2.4.1 Types of Machine Learning algorithms

The different types of machine learning are the following:

Supervised Learning

A supervised machine learning algorithm trains and learns the desired pattern from examples which have already been classified. The algorithm uses the example as an input and the already determined output of each example as a guideline to calculate an error and adapt the parameters.

A common way to evaluate the performance of such a classifier (more information about ways of performance evaluation will be provided in a later chapter) is to split the classified data set into two parts: one part to train the classifier (training set) and one part to evaluate its performance (testing test). First the classifier is trained using the training set and then the classifier attempts to classify the examples of the testing set without using the information regarding their already established classification. The next step is to compare the predictions of the classifier with the actual established classifications for performance evaluation purposes.

Supervised learning algorithms are categorized into classification and regression algorithms. In the case of classification algorithms, the purpose of those is to choose a value from a predetermined set for every example. For example, in the case of classifying an email as phishing or legitimate, the classification algorithm will give a value of either zero for a legitimate email prediction and a one for a phishing email prediction. Regression algorithms are used when outputs have a numerical value within a range.

Unsupervised Learning

Unsupervised algorithms take a set of input examples that are not classified and attempt to find a pattern by grouping them together or clustering them into neighbors of data points. The purpose of these algorithms is to find information or patterns which are unknown at the start of the operation, in order to provide insight for the data provided, for example by highlighting outliers which could not be grouped into any cluster or showing certain trends depending on different inputs.

2.4.2 Definitions

When choosing a machine learning model to use for a design or for a detection of a pattern, there is an overwhelming number of parameters and hyperparameters to choose and finetune in order to design and produce the best classifier. Also, the term “best” classifier has a lot of different interpretations and ways to calculate. In this chapter the fundamentals of machine learning performance evaluation as well as certain needed definitions are presented.

The case that is attempted to be solved through this thesis is the classification of emails as phishing or legitimate. This is a classification problem and the learning is supervised. This means that there will be an already classified data set which will be used to train a model in order to be able to classify new unclassified emails.

In supervised machine learning inputs are provided to train a model to predict outputs. In order to be able to explain the different models and how machine learning works in a more detailed manner, a few relevant definitions are presented.

- **Label** is the variable *y* that is to be predicted. In the case of emails, the label is a variable which could be called "Phishy" and has a value of one if the email is phishing and zero if it is legitimate.
- **Example** is one particular instance of data (row). For the purposes of the thesis, an example is one email that is in the data set. Examples can be labeled or unlabeled, depending on whether they have already been classified by a previous source (most times hand-classified).
- **Features** are the input variables x_i, i = 1, 2, …, n, where n is the total number of features, of the machine learning model. Features are characteristics or properties of the emails (in our case) which are retrieved from each example that is to be classified. Each feature provides a certain piece of information in its own way and in total they provide a whole picture of the email, something that will hopefully highlight the pattern that is to be detected.
  In the case of the emails, features may include the properties of the email (number of URLs in the email, number of attachments, whether the email has HTML content etc.) as well as textual features (frequency of specific words, similarity of sentences between each email etc.). A detailed presentation of the features that were retrieved is at the approach chapter.
- A **model** is the resulting classifier from the machine learning process and maps each example to a prediction *y'*.

For better understanding, the Linear regression model will be used at this point as an example. In this model, the n features x_i are retrieved from each example with a feature retrieving algorithm, there are the provided labels/outputs *y* for each example, and base model has that following formula:

$$\mathbf{y' = b + w_1x_1 + w_2x_2 + \ldots + w_nx_n,}$$

where *b* is the bias and w_i, *i = 1, 2, …, n* are the weights for each respective feature. In supervised learning, the algorithm builds a model by changing the values of the bias and the weights after every example is passed through the model so that an error is minimized using a specific loss function.

A widely used loss function is the Mean Square Error (MSE) which is the average of the sum of squared errors of every prediction.

$$MSE = \frac{1}{n}\Sigma\underbrace{\left(y - \widehat{y}\right)^2}_{\text{The square of the difference between actual and predicted}}$$

- **Overfitting** is the process of training the model so that the error is exceptionally low using the training data, but the resulting model is bad at generalization, meaning that although the error from the training and testing data is very low, the model is not good at predicting new examples. The proper way of training is to fit the training data well (have a relatively small value from the loss function) but also fitting it in a simple manner, so that the resulting model is able to "generalize", meaning that it is able to detect the pattern that is desired to be detected.

- **Test and Training sets**: In order to train the model and properly evaluate its performance, the available already classified data set needs to be split into two sets: a test set and a training set. These two sets must be independent from each other and randomly picked from the original data set. This will ensure that the model is trained without any bias and that the performance evaluation is properly executed.
- **Cross Validation**: Is a training and performance evaluation technique with the purpose of removing any random bias that might appear because of the test and training sets being a random choice of examples. The process is as shown in the figure below [13]:
    - The data set is split in n equal, independent and random subsets.
    - Choose one of the subsets as a test set and merge the other subsets into one training set.
    - Train the classifier, make predictions and evaluate performance.
    - Repeat the process by setting each time the next subset as a training set and merging the other subsets into a training set.
    - In the end, there are performance results for every subset of the data set. The total performance results consist of the mean values of every subset performance.

  This will ensure that, if there is any bias in any of the test subsets, that will be minimized after calculating the mean values with the other subsets' performance results.

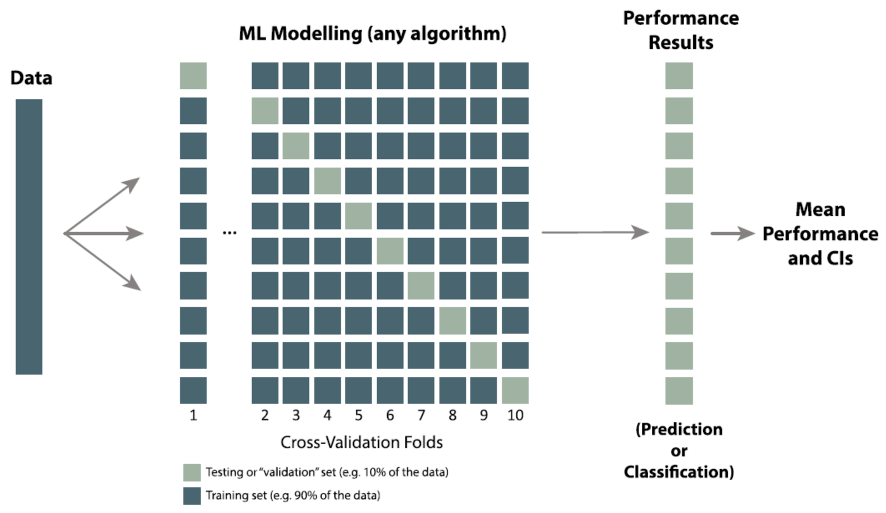

ML Modelling (any algorithm)
Performance Results
Data
Mean Performance and CIs
1
2
3
4
5
6
7
8
9
10
Cross-Validation Folds
(Prediction or Classification)
Testing or "validation" set (e.g. 10% of the data)
Training set (e.g. 90% of the data)

Figure 5 Cross Validation

- **Validation set**: The validation set is used when optimizing the hyperparameters of the machine learning model. It is an additional set which is retrieved from the original data set, meaning that if it is used, the original data set will be split into 3 subsets: test, training and validation. The process when using the additional validation set is as follows:
    - The training data set is used to train the model.
    - The validation data set is used to evaluate the performance of the model and use that information to tune any hyperparameters of the training process and of the model itself.

- After all the tuning is finished, then the test data set is used to evaluate one last time the performance of the machine learning model.

The purpose of the additional data set is to reduce the chance of overfitting which might occur while tuning every parameters too much so that it performs exceptionally well in the validation set, but when the model is evaluated using the test set, it will become apparent that overfitting has indeed occurred.

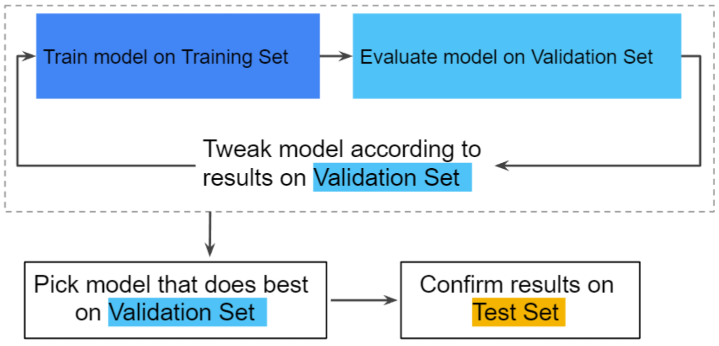

Train model on Training Set
Evaluate model on Validation Set
Tweak model according to results on Validation Set
Pick model that does best on Validation Set
Confirm results on Test Set

Figure 6 Training Process using Training, Validation and Test Sets [12]

- **Feature Engineering** is the process of retrieving features and choosing what features to retrieve from a data source (from raw data – emails in our case). Depending on the type of feature that is chosen to be retrieved from the data source, there needs to be some after-processing.
  - Numerical values can be passed as is.
  - Strings need to be encoded for the classifier to be able to interpret them properly. A frequently used algorithm for encoding strings is called One-hot encoding. Using this algorithm, each different string value that the feature can get is represented as an index in a boolean vector. The vector has a value of zero at every index and a value of one only at the index that represents the string value that is found at the specific example.

  When it comes to choosing a feature, there is a set of good methods that should be followed in order to provide the classifier with a set of features which will highlight the pattern in a clearer manner. A good feature should have the following properties:
  - It should have a value in most examples. There are cases in which a feature cannot be retrieved from the example, which might occur for several reasons one of which is that the data set is slightly corrupt. For example, in an email data set, there might be an email in which the receivers' addresses were removed. In that case, a feature cannot be retrieved. There are methods with which missing values could be filled. In the case of a numerical type feature, the missing value could be replaced by the average of the other examples' values, the minimum, maximum etc. However, features that end up not being able to be retrieved from most examples and end up being replaced by artificial values provide little information and are not useful for pattern detection.
  - It should have a meaningful value. When choosing features for detecting a certain pattern, there should be a critical thinking process, in which the features that actually provide relevant information should be chosen. For example, when choosing features for classifying an email as phishing or legitimate, choosing a feature like how many URLs are present in the email, or how many of the URLs have a domain name or are plain IPs provide valuable information for the classifier, because a IP URL usually means that the

respective web page is created in rush and there is a possibility that it will lead to a phishing website.

- Following the mentality of the previous method, unique identifiers (IDs) should never be used as features, because, although they are useful when it comes to searching the database for certain examples, they are different for each example and do not provide information that helps in terms of pattern recognition.
- Features should not have values that have a different meaning in a certain range and another meaning in another certain range. For example, a feature that represents the years that a house has been on a market should just be that. If a house has never been a market, it is not good practice to give a value of, for example, -1 to show that fact. It is good practice to set an additional feature with boolean value which will represent whether the house has been on the market or not.
- Features' represented values should not change over time. For example, if a feature describes a time value and the source model happens to have certain changes, it should not change the unit that the time is outputted as. If the source model used to output that time value in seconds, it should not change to, for example, minutes, as that would change the way that the time is represented and disrupt the ability of the classifier to detect and learn the pattern that is set to learn.
- Features should not have extreme outliers. That is because examples with extreme values will result in the loss function having an extreme output value. This value will result in an extreme tuning to the parameters and possibly disrupt the almost trained model.

2.4.3 Performance Evaluation Metrics

In order to evaluate the performance of a trained machine learning model, there is a plethora of formulas to calculate metrics which represent that [14]. These a presented below.

Confusion Matrix

A confusion matrix is a 2x2 matrix in which all the possible combinations of predictions and actual outcomes are listed. The values for each cell of the confusion matrix are equal to the number of the respective predictions which were executed on a test set. For example, if the model is built to detect phishing emails, the confusion matrix is like follows:

True Positive (TP): • Prediction: Phishing Email • Reality: Phishing Email	**False Positive (FP):** • Prediction: Phishing Email • Reality: Legitimate Email
False Negative (FN): • Prediction: Legitimate Email • Reality: Phishing Email	**True Negative (TN):** • Prediction: Legitimate Email • Reality: Legitimate Email

Table 1 Confusion Matrix

Accuracy is the ratio of correct predictions:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

Precision is the ratio of the correctly predicted positive examples over the total positive predictions:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Recall or probability of detection (PD) is the ratio of the amount of correctly predicted phishing emails over the total amount of phishing emails:

$$\text{Recall} = \text{PD} = \text{Proportion of actual positives was identified correctly} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

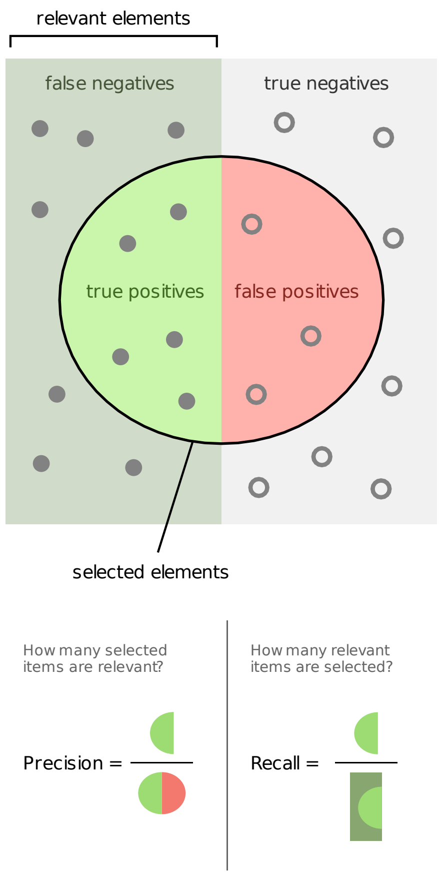


Figure 7 Precision and Recall

Probability of false alarm (PF) is the probability that a positive prediction is incorrect:

$$\mathrm{PF} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}$$

F1 score is the harmonic mean of the precision and recall. F1 score has value range from 0 to 1, where 1 is equal to having perfect precision and recall both equal to 1. F1 score is useful for providing a relation metric between precision and recall, as these two values have a reverse analogy in a way, which is represented with the ROC curve which is described below.

$$\mathrm{F1} = \frac{2}{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

A **ROC curve** (Receiver Operating Characteristics curve) is a plot of sensitivity vs. (1-specificity) or the probability of false alarm (false positive rate) on the X-axis and the probability of detection (true positive rate) on the Y-axis. This curve passes through the points (0,0) and (1,1). Descriptively, this curve plots how well a model classifies correctly different examples compared to how many false alarms will happen. A perfect ROC curve would have a value of 1 for every X in (0,1] and is shown at the following figure.

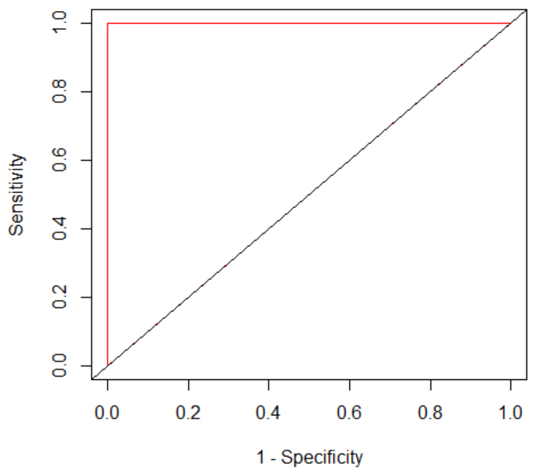


Figure 8 ROC curve of a perfect classifier

A more usual case of a ROC curve is the following figure [15]. As shown, the different points of the ROC curve are created from different decision thresholds of the classifier. The important regions of the ROC curve are shown in the next figure. If the ROC curve is a straight line from the points (0,0) and (1,1), that is when the false positive rate and the true positive rate are equal in value for every decision threshold. That means that the classifier does not recognize any pattern and does not go through any operation in order to predict examples. A negative curve is an unacceptable curve, and something is wrong on a fundamental level. A ROC curve is a monotonically increasing function.

The **Area under the ROC curve** (**AUC**) is a widely used performance metric for imbalanced datasets. It is a useful numerical representation of the ROC curve itself, which provides the relevant needed information that a ROC curve provides in cases where a curve is not possible to be presented or cannot be handled (e.g. in a program as input). Since the perfect ROC curve creates a 1x1 square, the AUC has values from 0 to 1.

The ROC curve and the produced AUC metric provide a more generalized metric for the classifier. For example, if in an email data set there were 99 legitimate emails and 1 phishing and the classifier just outputted a value of zero (legitimate email) for every email, then the accuracy would be 99% for this data set. However, the ROC curve will display that fact by it being a straight line from (0,0) to (1,1).

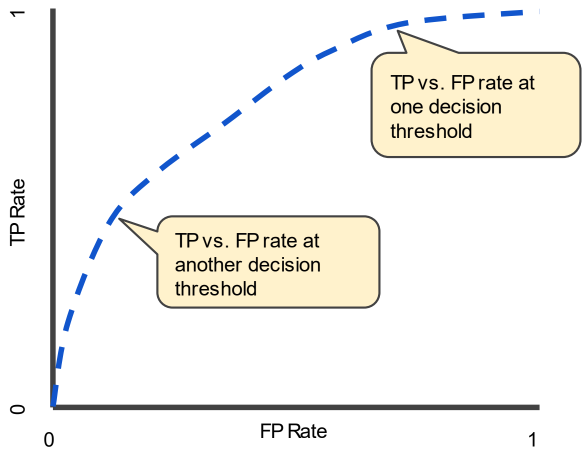


Figure 9 ROC curve

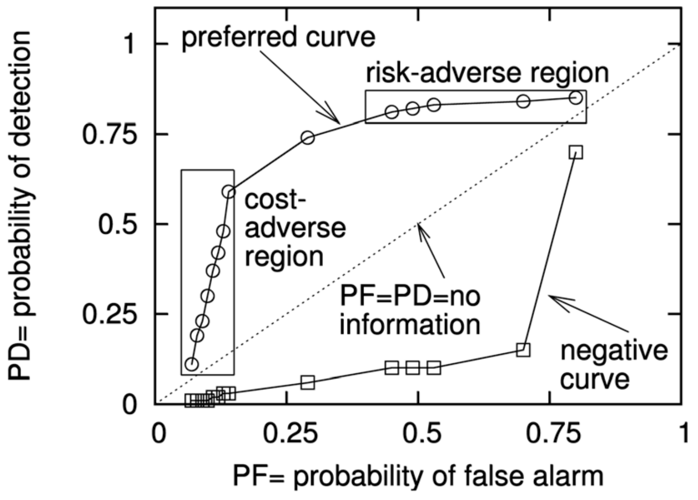


Figure 10 Regions of ROC curve

The **Precision-Recall (PR) Curve** is a curve with the precision value on the y-axis and the recall values on the x-axis on different classification thresholds. Similar to the ROC curve, the PR curve is useful because, while in every classification problem it is desired to have both high precision and high recall, there is a tradeoff between the two. The PR curve gives a visual and more generalized representation of the precision and recall performance of the corresponding classifier. Note here that a classifier with a better ROC curve will have a better PR curve, too. Opposite to the ROC curve, a PR curve is monotonically decreasing. The following figure is a typical PR curve [16].

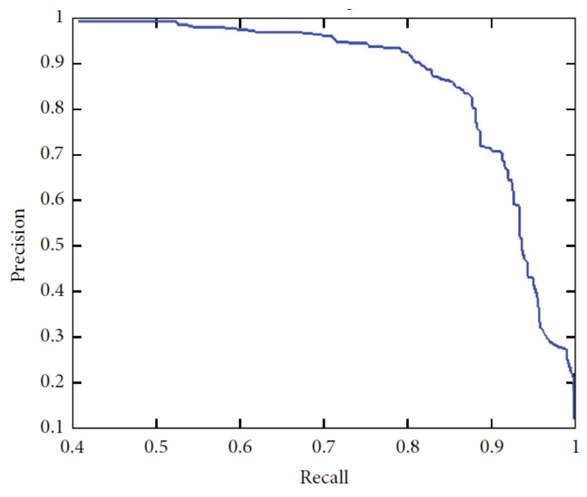


Figure 11 PR Curve

The **False Positive Rate (FPR)** and **False Negative Rate (FNR)** are statistical representations of the false positive and false negative numerical results:

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}$$

$$\mathrm{FNR} = \frac{\mathrm{FN}}{\mathrm{FN} + \mathrm{TP}}$$

The **Error** is the ratio of wrong predictions over the total number of examples:

$$\mathrm{Error} = \frac{\text{Wrong Predictions}}{\text{Total number of examples}} = \frac{\mathrm{FN} + \mathrm{FP}}{\mathrm{TP} + \mathrm{FP} + \mathrm{FN} + \mathrm{TN}}$$

The previous three metrics should be minimized, but there is a tradeoff between the FPR and FNR. When it comes to the classification of emails as phishing or legitimate, there is, on the one hand, the idea that the minimization of FPR is more important because the chance of wrongfully classifying legitimate emails as phishing emails is the main reason for users' resistance to such types of emails filters [17]. That is because the users will feel like they will lose useful information from legitimate emails because they were wrongfully classified as legitimate. On the other hand, there is also the idea that there should be an attempt towards minimization of the FNR in the expense of a relatively higher FPR, because it is more important to let as few phishing emails as possible pass classified as legitimate than a few legitimate emails be classified as phishing. That is because these few phishing emails that will pass as legitimate might cause more problems than the few legitimate ones that will be classified as phishing.

Average Absolute Error (AAE) is the average of the absolute differences of predictions and labels:

$$\mathrm{AAE} = \frac{1}{n}\sum_{i=1}^{n} |\mathrm{prediction}_i\text{-}\mathrm{label}_i|$$

2.4.4 Machine Learning Models

In this chapter there will be a description of the base machine learning models with certain advantages and disadvantages.

Logistic Regression

As mentioned in a previous chapter, regression is the type of machine learning algorithm the produced model of which is aimed to predict values not from a finite set of values, but rather a continuous space of values. However, regression models can be used for classification, too, by using the continuous outputted value and set various thresholds. Depending on the thresholds and the continuous output, a value from the set classification values is set as the final output. For example, in the case of phishing emails, one threshold is set and if the continuous output is higher than the threshold then the email is classified as phishing and legitimate otherwise.

As described before, linear regression attempts to adapt a linear function so that it represents the pattern as well as possible and minimize errors. However, linear regression is limited to numerical values only. Logistic Regression is a type of non-linear regression that can accept categorical data and predicts the probability of occurrence using logit function. Logistic Regression does not consider linear relationship between variables/features.

A logistic regression base logit function is shown at the figure below and has the following formula.

$$P(y = 1) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n)}}$$

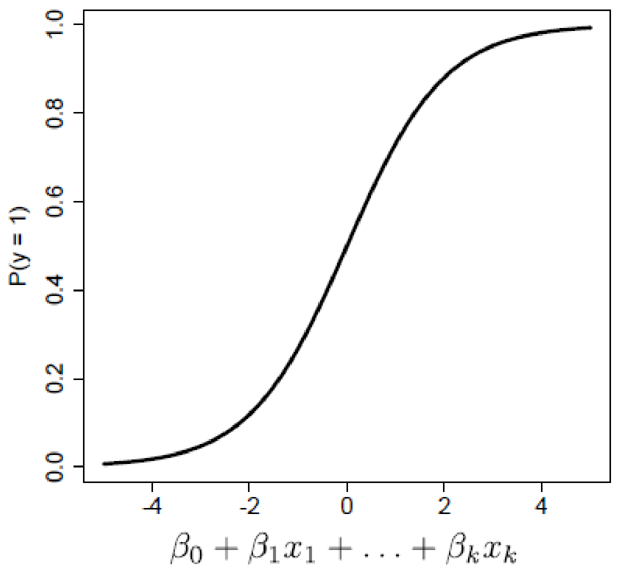


Figure 12 Logit Response Function

The bias and the feature weights are tuned to minimize error.

The **advantages** of logistic regression are [18]:

- Good performance with small datasets
- Its output can be interpreted as a probability

Disadvantages of logistic regression are:

- Data assumptions are needed to be complied
- It can only provide linear solutions

K-Nearest Neighbors

K-nearest neighbor is a non-parametric, instance-based learning method [19]. Instance based classifiers are also called lazy learners, because they do not train but rather just store all the training examples. The examples are store as points in a feature space, each coupled with its respective label value (for example 0 for legitimate email and 1 for phishing email). When a new example is to be classified, its feature values are used to place it in the feature space. Then its k nearest neighbors are considered and the classification value is equal to the label value which is the majority. For example, assume that k=5. A new email is put on the feature space where its 5 closest neighbors consist of 3 legitimate emails and 2 phishing emails. The new email will be classified as a legitimate one. This process is shown at the figure below [20].

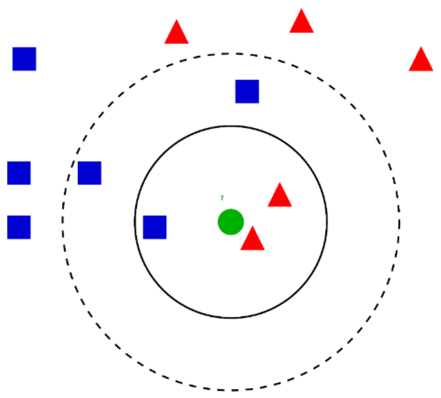


Figure 13 k-Nearest Neightbors Classifier

The choice of k affects the performance of the k-nearest neighbor algorithm. If the value of k is too small and there is noise, then the noise might be the majority in specific regions resulting in wrong predictions. On the other hand, if the value of k is too large, then there might be neighbors from other label values included, resulting again in wrong classifications.

The **advantages** with the k-nearest neighbors algorithm are:

- It is an easy to understand algorithm and to implement.
- Lazy learning methods have a shorter training time.
- It performs well on applications where a sample has many class labels.

However, this algorithm has several **disadvantages**:

- Even though the training time is short, this comes with a drawback. That is the classification of new examples is generally computationally expensive, especially if the value of k is high, because the distance between the new example and every neighbor needs to be calculated.
- It has large storage requirements, because every example during the training process is stored.
- There is not a specific way of choosing the optimal value for k. The only way is by trial and error.

Artificial Neural Networks

An artificial neural network is a set of connected input/output units where each connection has a weight and every node has a bias. Artificial Neural Networks consists of an input layer from which the inputs/features are fed into the ANN, one or more hidden layers, which help with the classification process and an output layer which outputs the desired classification value.

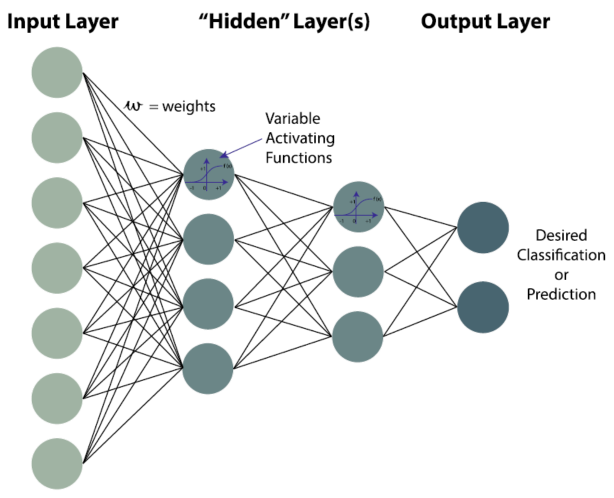

Input Layer
"Hidden" Layer(s)
Output Layer
w = weights
Variable Activating Functions
Desired Classification or Prediction

Figure 14 Artificial Neural Network

Every node calculates the output as follows (also shown at the figure below [21]):

$$\mathrm{Output} = \mathrm{f}(\sum_{\mathrm{i}} \mathrm{weight}_{\mathrm{i}} \cdot \mathrm{input}_{\mathrm{i}} + \mathrm{bias})$$

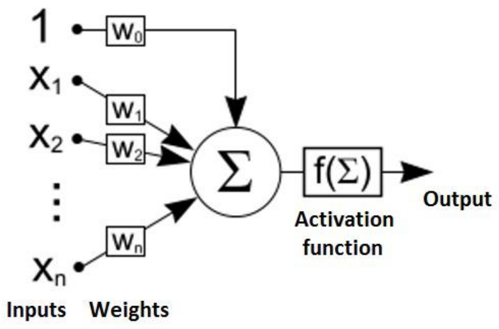


Figure 15 Node in ANN

The function f is called an activation function. There are different activation functions depending on the application. The activation functions output range is between 0 and 1. The simplest activation function is the step function [22].

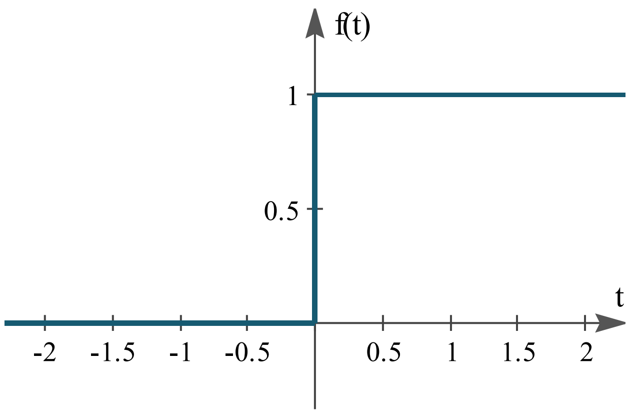


Figure 16 Step Function

More complex functions are used to produce more consistent and accurate results. A widely used activation function is the sigmoid function [23].

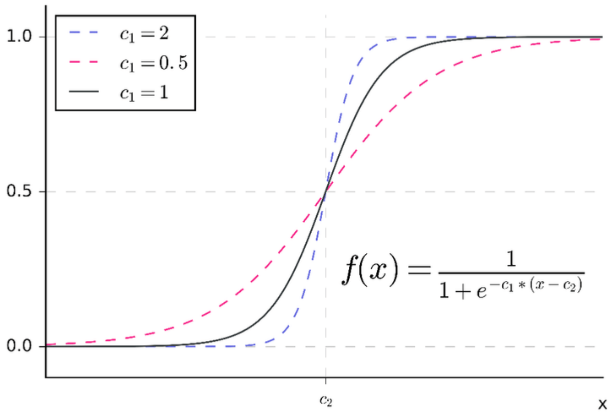


Figure 17 Basic Sigmoid Function

A popular way to train an ANN is called Back Propagation. The training process is as follows:

1. Initialize the weights and the biases.
2. Feed the training sample into the ANN.
3. Calculate the output and the error of it with the desired output.
4. For each neuron, calculate what the output should have been and the difference between it and the actual output (local error).
5. The error is propagated backwards by updating the weight and biases to reflect the error of the network's prediction.

The process is repeated until a specified error threshold is reached, or a specified number of epochs (iterations of the whole training process) is reached, or the error metric has not improved over the last certain number of epochs, or when there are signs of overfitting (error value on validation set is more than a certain amount than the error on the training set).

Advantages of artificial neural networks include:

- Able to tolerate noisy data.
- Can be used when there is not clear relation between the features and the label.
- Parallelization techniques can be used to speed up the process.
- Successful on otherwise hard tasks, such as handwritten text recognition, image and voice recognition etc.

Disadvantages of ANNs include:

- Long learning time
- Difficult to interpret (complicated network from which the weights and biases do not necessarily represent anything specific)

Recurrent Neural Network

RNNs are a type of ANNs which improve on the memory capabilities of traditional ANNs [24].

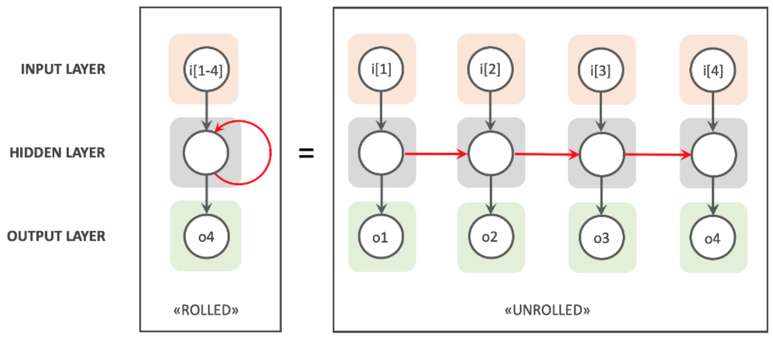


Figure 18 Recurrent Neural Network

The additional feature of RNNs is the feedback loop in the hidden layer. This provides additional insight in sequential data. For example, if the input in the neural network is a sentence of words and each input is a word, with the feedback loop in the hidden layer the RNN is able to take into account the whole sentence to train and update its nodes' weights and biases, something which would not be possible if there was no feedback loop.

Decision Trees

Decision tree classifies data into discrete ones using tree structure algorithms [19]. The idea behind these machine algorithms is to highlight the structure behind the data/features. The training set is recursively partitioned into subsets which are purer than the original, meaning that the subsets are split in an as equal manner as possible. Each node of the produced decision tree represents a feature. Each branch represents a certain value range of that feature. An optimal case of splitting the data set (which is practically never occurring) would be a feature which has a 1-1 relation with the label, meaning that with this one feature it would be possible to classify every example. However, this case is never apparent, and would also mean that there is no practical need of machine learning because it is just a case of if-else. The leaf nodes represent a classification result. The decision tree, as a classifying algorithm, works by getting each example through the tree, depending on the respective feature values, and classifying the example based on the leaf node value.

When it comes to splitting the data, on the root node and the internal nodes, the feature that is chosen to be used as the one to split the data is the one that best divides the training data. There are different metrics to find which feature does that including Information gain, Gain ratio, Gini index, Chi square, C-SEP, G-statistics and Minimum description length (MDL).

Overfitting can occur if the parameter of maximum tree depth is set to a too high value. To counter overfitting there is the tree pruning method, which includes two approaches:

- **pre-pruning**: The decision tree is pruned by halting its development early, when a certain threshold value is reached.
- **post-pruning**: The decision tree is pruned after the tree is complete, by remove certain sub-trees which do not help with the accuracy, or split the tree too much for slight gain in accuracy (which probably means that overfitting is happening at that sub-tree).

An example of a built decision tree for the classification of phishing emails is shown at the figure below.

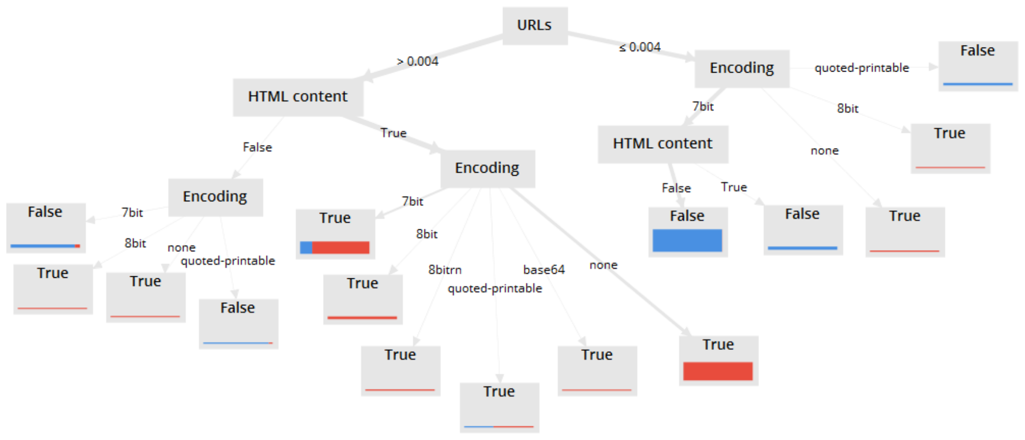


Figure 19 Decision Tree for phishing emails

The **advantages** of Decision Trees are the following:

- Decision trees are simple and fast on the classification (each new example that is to be classified only goes through the tree, which is basically a few if-else cases).
- They are able to handle high dimensional data.
- The built decision tree is easily comprehensible.
- They can handle features of different values, because the splitting happens based on how well each feature splits the training set and not on what value each feature has.

The **disadvantages** of Decision Trees are the following:

- High training time, because for each node branching of the tree, every example is needed to be used to decide which feature to choose.

- Lack of available memory, when dealing with large databases.

Support Vector Machines

SVMs are based on statistical learning theory and structural risk minimization principal and aim in determining the location of decision boundaries. The basic idea is that a support vector classifier attempts to find a straight line which separates the different classes with the maximum possible distance between the line and each class, as shown in the following figure.

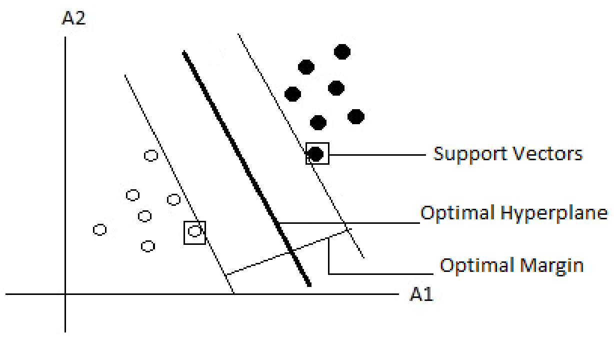


Figure 20 Support Vector Classifier

In the simplest case of linearly separable data, only one straight line is required to properly separate the data and determine the final classifier. In the case that the data is not separable with one straight line, the SVM finds the hyperplane that maximizes the margin while minimizing the number of misclassification errors.

Advantages of SVMs are:

- Robust and accurate method compared to the other classifiers
- Theoretical foundation, requires small training data set, operates well with high dimensions.
- Less prone to overfitting

Disadvantages of SVMs include:

- Requires high computational power and processing time.
- Long training time.
- Memory requirements are analog to the square of the number of training examples.
- The results are difficult to interpret.

Naïve Bayes

The Bayesian are statistical classifiers. They classify by calculating the probability that an example belongs to a specific class. The Naïve Bayes classifier assumes that all variables contribute towards classification and are mutually correlated. It is based on the Bayesian theorem, where if H is a hypothesis that certain data X belongs to a class C, then

$$P(H|X) = \frac{P(X|H)P(H)}{P(X)}$$

P(H|X) is the probability of hypothesis H if X is true, P(H) is the probability of H independent of X and P(X|H) is the probability of X if H is true.

The algorithm calculates the probability of each class – hypothesis and classifies the example with the class with the highest probability.

The **advantages** of the Naïve Bayes classifier are the following [19]:

- Low training and classification time, easy to construct.
- Does not involve any iterative parameter which would have to go through every example, so operates well against big data sets.

The **disadvantages** of the Naïve Bayes classifier are the following:

- It is generally less accurate than other classifiers.

Ensemble Classification

Ensemble employs more than one classifier, takes their outputs and combines them in different ways to produce the final prediction.

Random Forest

The Random forest is a classifier which combines many decision tree classifiers, where each tree depends on the values of a random vector sampled independently. During training, a number of decision trees are constructed which are then used for the class prediction. This is achieved by considering the voted classes of the decision trees and the class with the highest vote is considered to be the output [25].

AdaBoost

Boosting is a type of ensemble training using decision trees, where each new tree is trained on a modified version of the original data set [26]. AdaBoost trains the first decision tree with each observation being assigned a weight value. At the first tree every example has the same weight value. After the evaluation of the first tree, the examples which were difficult to correctly classified have their respective weight values increased and the easily classified examples weights are reduced. The process is repeated, and an ensemble tree model is built from which the classification output is the weighted sum of the previous created sub-trees.

Gradient Boosted Trees

Gradient boosting machines operate the same way as AdaBoost, with the only difference being how the adaptation of the newly created trees is done. Instead of adapting weight values for the different examples, GBM uses gradients in the loss function. The loss function could be a user specified cost function instead of a generic uncontrollable loss function which offers less control.

Majority Voting

With majority voting, the classifiers are trained on independently, and then a voting process is executed, where the class that most of the classifiers predicted that the example is will be the final output [27].

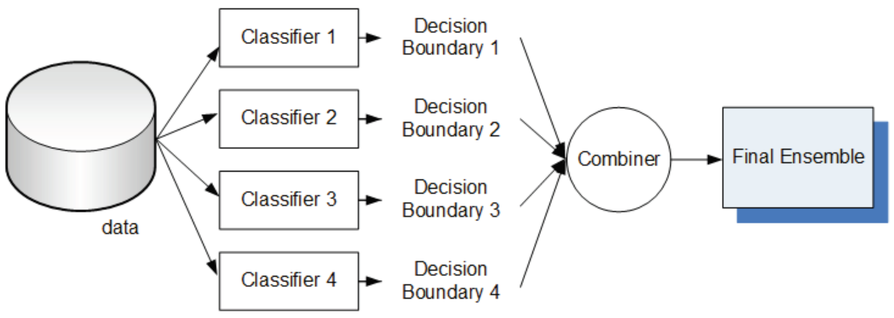


Figure 21 Majority Voting

Stacking

Stacking is a meta classifier, where the first step includes training a variety of different simple classifiers, and then using their outputs as features to train a last classifier which will output the final prediction [28].

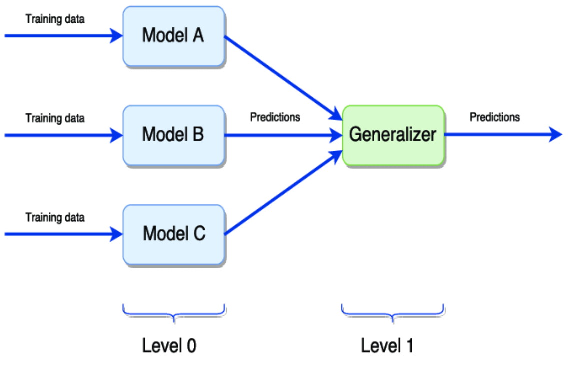


Figure 22 Ensemble Stacking

2.5 Related Work

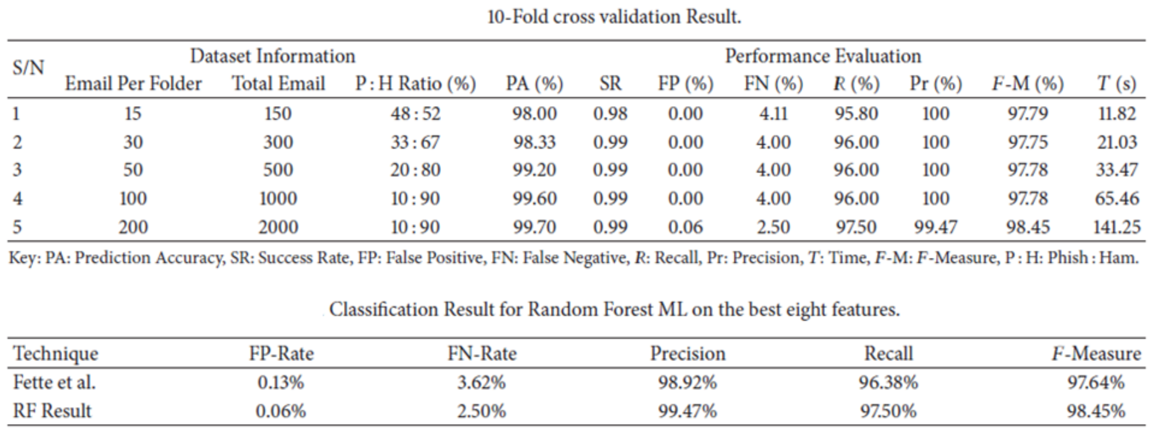
In this chapter relevant work will be presented in the field of phishing email and website detection with machine learning techniques.

2.5.1 Classification of Phishing Email Using Random Forest Machine Learning Technique [25]

This paper investigates the use of the random forest technique machine learning algorithm in classifying emails as phishing or legitimate. In the proposed classification method, the features are extracted directly from each email. There are no queries sent over the network or large data stored, thus reducing the memory and processing time required. The features retrieved are based on the properties of the email and do not take into account the text itself of the email. 15 features were identified:

- URLs containing **IP address**
- **Disparities** between “href” attribute and link text: When writing HTML code, it is possible to have the text of a link be different than the actual URL that is to be visited.
- **Presence of the words** “link”, “click” and “here” in the link text
- **Number of dots** in domain name: The number should not be more than three. If it is, a binary value of 1 is recorded.
- **HTML email**: The MIME standard defines the type of content contained in the email. If it is “text/html” then a binary value of 1 is recorded.
- **Presence of Javascript**
- **Number of links**
- **Number of linked to domain**: The number of distinct domain names present in the URLs is counted.
- **Word list features**: This accounts for the only textual features of this classifier. It does not take into account the whole text of the email, but rather the counts the presence of specific words which indicate a possible attempt of persuasion of the email receiver. They are separated into different subcategories:
    - Update, confirm
    - User, customer, client
    - Suspend, restrict, hold
    - Verify, account, notif
    - Login, username, password, click, log
    - SSN, social security, secur, inconvinien

Before the decision trees of the random forest algorithm were constructed, the information gain (IG) from all the 15 features is calculated and the eight features with the best IG are selected for the training and classification process.



10-Fold cross validation Result.

S/N	Dataset Information			Performance Evaluation							
	Email Per Folder	Total Email	P : H Ratio (%)	PA (%)	SR	FP (%)	FN (%)	*R* (%)	Pr (%)	*F*-M (%)	*T* (s)
1	15	150	48 : 52	98.00	0.98	0.00	4.11	95.80	100	97.79	11.82
2	30	300	33 : 67	98.33	0.99	0.00	4.00	96.00	100	97.75	21.03
3	50	500	20 : 80	99.20	0.99	0.00	4.00	96.00	100	97.78	33.47
4	100	1000	10 : 90	99.60	0.99	0.00	4.00	96.00	100	97.78	65.46
5	200	2000	10 : 90	99.70	0.99	0.06	2.50	97.50	99.47	98.45	141.25

Key: PA: Prediction Accuracy, SR: Success Rate, FP: False Positive, FN: False Negative, *R*: Recall, Pr: Precision, *T*: Time, *F*-M: *F*-Measure, P : H: Phish : Ham.

Classification Result for Random Forest ML on the best eight features.

Technique	FP-Rate	FN-Rate	Precision	Recall	*F*-Measure
Fette et al.	0.13%	3.62%	98.92%	96.38%	97.64%
RF Result	0.06%	2.50%	99.47%	97.50%	98.45%

Figure 23 Random Forest Performance

2.5.2 Analysis and detection of e-mail phishing using PySpark [29]

In this paper the Spark tool, and more specifically PySpark, which is a python-based implementation of Spark, is used with machine learning classifiers to classify phishing emails. The Spark Python API exposes the Spark programming model to Python. Spark applications run by having a Spark driver that runs the main function and executes parallel operations on a cluster.

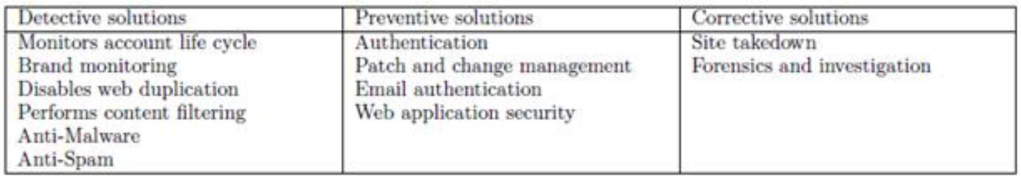
The components used for the machine learning algorithm are the following:

- **Natural Language Toolkit** (NLTK), which is a collection of libraries and programs for natural language processing (NLP) for English written in Python.
- **Virus Total API**, which allows to build scripts to access the information provided by Virus Total, an online tool which combines many antivirus products and online scan engines to check for viruses or to verify against any false positives [30].
- The Naïve Bayes classifier is used for the classification of the email, which is described in a previous chapter.

2.5.3 A Comparison of Machine Learning Techniques for Phishing Detection [31]

In this paper a variety of different machine learning algorithms were compared for detecting phishing emails. 2889 emails in total were user, 59.5% of which were legitimate, and the rest were phishing emails. 43 features were used for the classification process.

According to the Anti-Phishing Working Group (APWG), there are three categories of defense mechanisms against phishing: detective, preventive and corrective.



Detective solutions	Preventive solutions	Corrective solutions
Monitors account life cycle	Authentication	Site takedown
Brand monitoring	Patch and change management	Forensics and investigation
Disables web duplication	Email authentication	
Performs content filtering	Web application security	
Anti-Malware		
Anti-Spam		

Figure 24 Phishing and fraud solutions

Anti-phishing toolbars are mentioned as related work against phishing attacks, but the results are not satisfactory (50% - 75% detection rates).

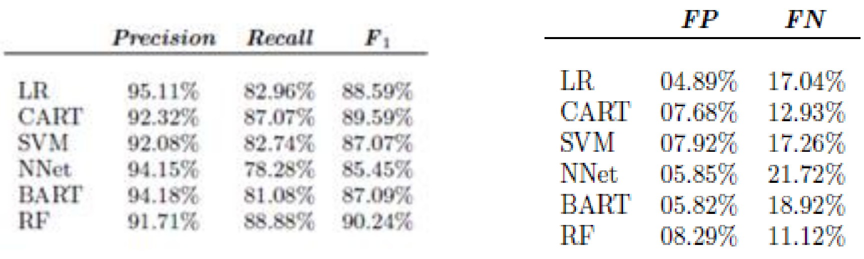
The machine learning algorithms which were tested are the following:

- Logistic Regression
- Classification and Regression Trees (a category of decision trees)
- Random Forests
- Neural Networks
- Support Vector Machines
- Bayesian Additive Regression Trees

The features used in this paper represent the frequency of the most frequent terms that appear in phishing and legitimate emails. This method of choosing words as features is widely applied in text mining and is called "bag-of-words" method. The features in this paper are words. The process that every email goes through in order to have their features retrieved is the following:

1. The attachments of the email are removed.
2. Header information, HTML tags and elements are extracted.
3. The stop words (I, you, the, is, there, in, on etc.) are removed from the text.
4. Stemming is applied to the text (removal of prefixes/suffixes, e.g. "running" becomes "run").
5. The most frequent terms are detected with TF-IDF (term frequency – inverse document frequency) and the features are created from them. TF-IDF works by giving set higher weight to words that appear frequently in one example and not frequently in the whole data.

The results are in the following tables and figure.



	Precision	*Recall*	F_1
LR	95.11%	82.96%	88.59%
CART	92.32%	87.07%	89.59%
SVM	92.08%	82.74%	87.07%
NNet	94.15%	78.28%	85.45%
BART	94.18%	81.08%	87.09%
RF	91.71%	88.88%	90.24%

	FP	*FN*
LR	04.89%	17.04%
CART	07.68%	12.93%
SVM	07.92%	17.26%
NNet	05.85%	21.72%
BART	05.82%	18.92%
RF	08.29%	11.12%

Figure 25 Performance of model of paper

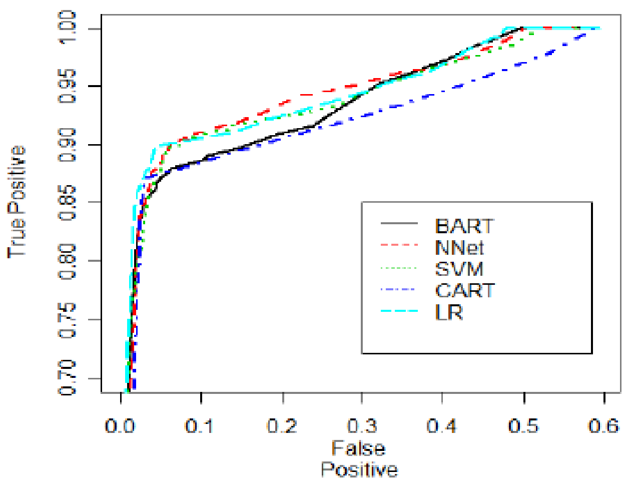


Figure 26 ROC curve for all classifiers of the paper

2.5.4 Deep Learning Based Phishing E-mail Detection [32]

The paper proposes a model using the Keras Word Embedding and Convolutional Neural Networks, followed by a Pooling layer, a fully connected layer, a non-linear activation function (sigmoid) and one output neuron for classifying the emails as legitimate or phishing, as shown in the following figure.

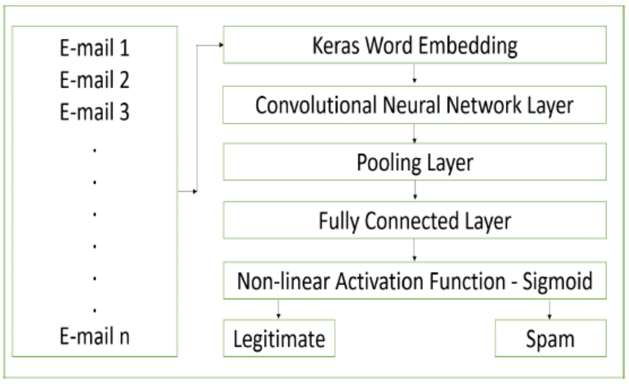


Figure 27 Deep Learning based model

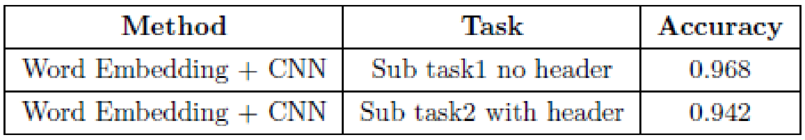
The components of this model are the following:

- **Keras word embedding**: Word embedding is function operating on texts. With it, every word is transformed into a vector, where the vector space axes represent different attributes/aspects of words. Every word has a value for each attribute and the final vector is produced. Each word being represented by a vector means that words with similar meanings will have neighboring vectors, thus making word embedding a more sophisticated version that a regular word frequency function.

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK or Theano.

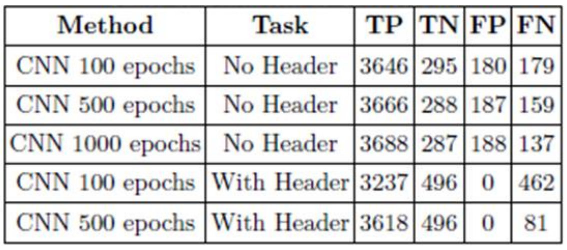
- **Convolutional neural network**: The emails are first passed through word embedding, so now the text is a series of numbers. This number-type file can be interpreted as an image (since images are also series of numbers as a file which represent the colors of pixels) and thus can be processes as one. First, they are convoluted, then pooled (by taking either the maxes or the averages) and then processes as vectors through a neural network classifier.

The results of this paper's proposed model are the following.



Method	Task	Accuracy
Word Embedding + CNN	Sub task1 no header	0.968
Word Embedding + CNN	Sub task2 with header	0.942

Figure 28 Accuracy of word embeding + deep learning model



Method	Task	TP	TN	FP	FN
CNN 100 epochs	No Header	3646	295	180	179
CNN 500 epochs	No Header	3666	288	187	159
CNN 1000 epochs	No Header	3688	287	188	137
CNN 100 epochs	With Header	3237	496	0	462
CNN 500 epochs	With Header	3618	496	0	81

Figure 29 Statistics of test result in relation to the number of epochs

2.5.5 An Approach to Detect Spam Emails by Using Majority Voting [33]

This paper proposes an ensemble model (majority voting) to detect spam emails. Note here that although the paper attempts to classify spam emails and not phishing ones, the majority voting method is a useful one and was considered useful for the purposes of this work.

The classification algorithms used in the paper are Naïve Bayes, Support Vector Machines, Random Forest, Decision Stump and k-Nearest Neighbors. As mentioned in a previous chapter, majority voting operates by first training several simple classifiers and then using their outputs as votes in order to make the final classification. The sample data set that was used is taken from the UCI [34] and the tool used to create and train the models was RapidMiner [35]. More about RapidMiner will be provided in the Approach chapter, where there is a description of the tools found in the literature. The proposed model is shown in the following figure. There are five inner classifiers and if four out of the five predict the email to be spam, then the final output labels the email as spam.

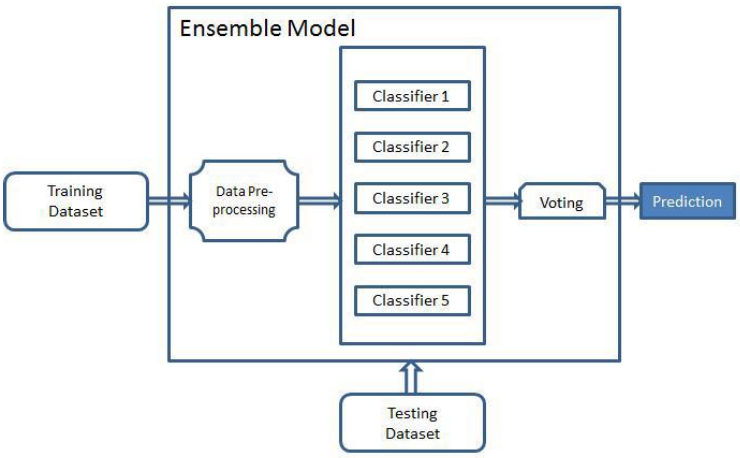
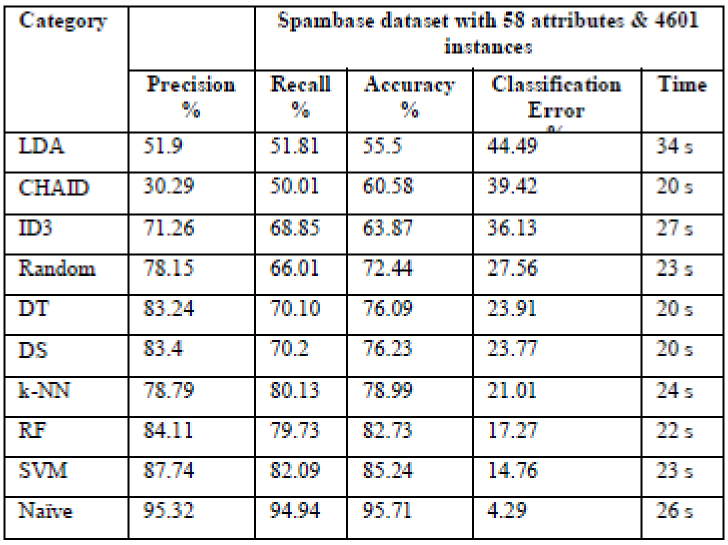


Figure 30 Majority Voting for spam email classification

Initially several different classifiers were trained and tested, with the following results.

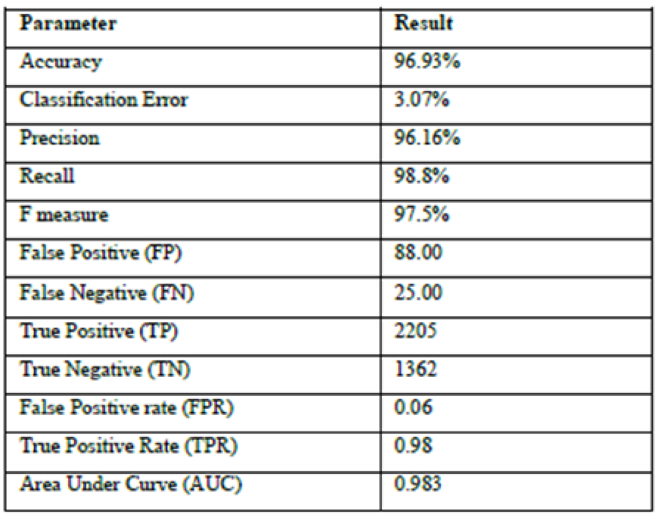


Category		Spambase dataset with 58 attributes & 4601 instances			
	Precision %	Recall %	Accuracy %	Classification Error	Time
LDA	51.9	51.81	55.5	44.49	34 s
CHAID	30.29	50.01	60.58	39.42	20 s
ID3	71.26	68.85	63.87	36.13	27 s
Random	78.15	66.01	72.44	27.56	23 s
DT	83.24	70.10	76.09	23.91	20 s
DS	83.4	70.2	76.23	23.77	20 s
k-NN	78.79	80.13	78.99	21.01	24 s
RF	84.11	79.73	82.73	17.27	22 s
SVM	87.74	82.09	85.24	14.76	23 s
Naïve	95.32	94.94	95.71	4.29	26 s

Figure 31 Classifier test results

The 5 best performing classifiers were selected (highest accuracy): Decision Stump, k-Nearest Neighbors, Random Forest, Support Vector Machine and Naïve Bayes.

Lastly, the ensemble model with the voting technique was built, trained and tested. The performance results proved that majority voting is a performance improvement.



Parameter	Result
Accuracy	96.93%
Classification Error	3.07%
Precision	96.16%
Recall	98.8%
F measure	97.5%
False Positive (FP)	88.00
False Negative (FN)	25.00
True Positive (TP)	2205
True Negative (TN)	1362
False Positive rate (FPR)	0.06
True Positive Rate (TPR)	0.98
Area Under Curve (AUC)	0.983

Figure 32 Majority Voting Model Results

2.5.6 Employing Machine Learning Techniques for Detection and Classification of Phishing Emails [36]

In this paper a neural network-based classifier is proposed, with the preprocessing steps being word embedding or vectorization. The model is made of six parts and uses five features. For the training process there was a ten-fold cross validation. The email data sets that were used were from Spam Assassin for the legitimate emails [37] and the Jose Navario phishing email corpus [38].

The process that every email goes through with the aforementioned six components is in the following figure.

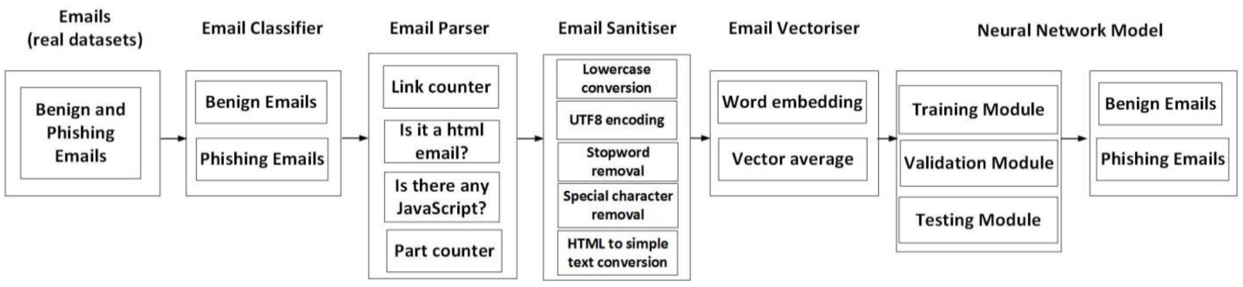


Figure 33 6 Components Neural Network Classifier

The five features that are retrieved from emails are the following:

- URL counter
- Whether the email has HTML content
- Whether the email has JavaScript code in it
- The number of the email's parts (attachments, HTML text, plain text etc.)

- Vector Average

For the Vector average: First the emails pass through the Email Sanitizer, which purifies the emails and makes them ready for vectorization. Vectorization or Word Embedding is a set of language modelling and feature learning techniques in Natural Language Processing (NLP) where words or phrases from the vocabulary are mapped to vectors of real numbers. Once all the words have been vectorized, the vectors for a message are summed and the average is calculated.

The resulting data set after retrieving the features from the emails is a csv file with 6 columns (5 features and 1 label) and 14370 rows/emails (6656 benign and 7714 phishing emails). Matlab [39] was used for the implementation of the Neural Network. 70% of the data was used for training, 15% for validation and 15% for testing. The confusion matrix with the results is the following.

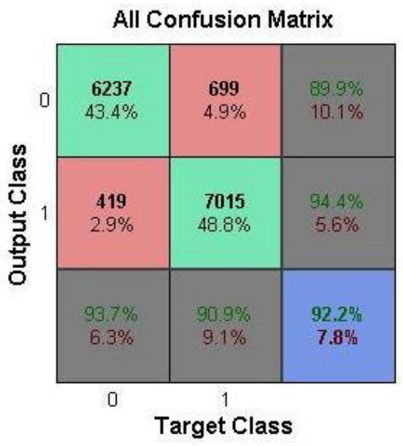


Figure 34 Confusion Matrix of 6 component NN classifier

2.5.7 Phishing Detection: A Case Analysis on Classifiers with Rules using Machine Learning [40]

This paper is not focused on phishing emails but rather on phishing websites. While this is not directly the purpose of the work of this thesis, it is a neighboring field of study which provides useful information about relevant methodologies.

The paper proposes the idea that a necessary step which should improve the performance of a predictive system is to pre-process the features in order to select the most effective ones. Feature effectiveness can be measured using different algorithms such as information gain, correlation analysis, Chi Square etc.

The paper is focused on rule-based classification systems, because it is believed that they are more suitable for phishing classification for the following reasons:

- The content of the models is straightforward human knowledge.
- The rules formed are usually in the "if-then" form, which is easily controllable by the users.
- Rule based classifications have performed well in many domains.

The machine learning classification algorithms used are eDRI (Thabtah et al., 2016A), RIPPER (Cohen, 1995), C4.5-Rule (Quinlan, 1993) and RIDOR (Gaines and Compton, 1995). The following figures show the error rate of each algorithm and the runtime required to build the respective models.

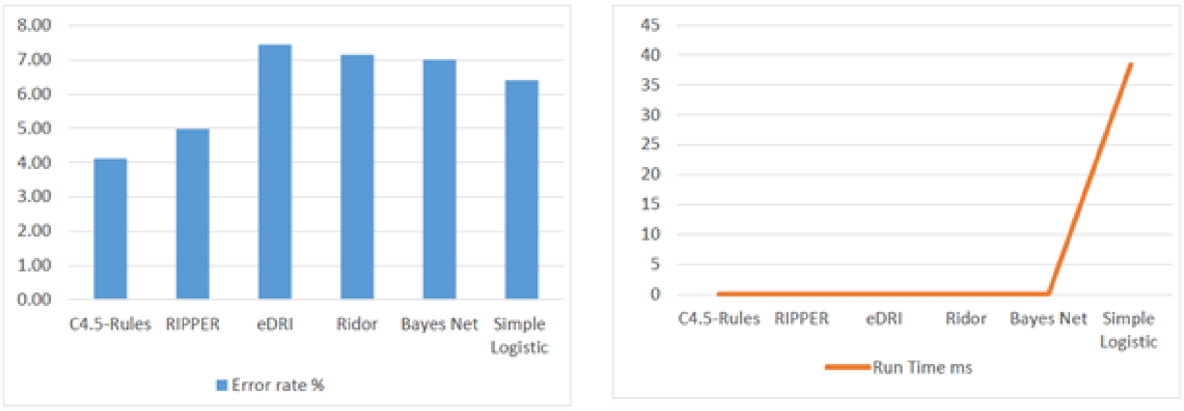


Figure 35 Error rate and runtime required to build the predictive models of the considered algorithms

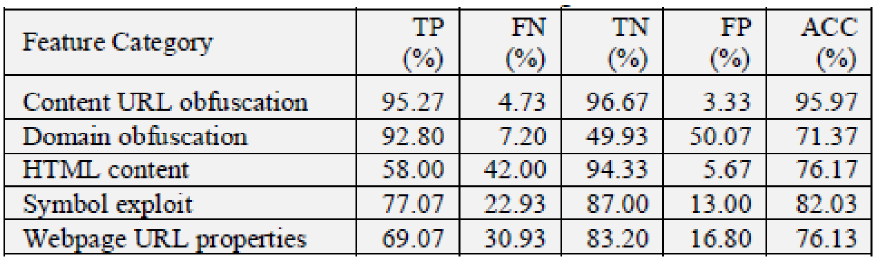
2.5.8 Identifying the Most Effective Feature Category in Machine Learning-based Phishing Website Detection [41]

In this paper, a variety of different features are retrieved from websites and are tested for accuracy (which set of features results in the best accuracy). The features were selected from related work on machine learning based phishing website detection. They are based on 18 research papers published between 2006 and 2016. The features were split into five different categories to determine which category yields the best results. These features are the following:

- Features related to **content URL obfuscation**:
    - Count percentage of external and resource URLs in the HTML code.
    - Check if the favicon is loaded from a different domain name.
    - HTML forms the actions of which lead to a different domain name.
    - Count the percentage of hyperlinks which contain empty values, self-direct value, the URL of a current webpage or some abnormal value.
    - JavaScript code with onMouseOver to display a fake URL in the status bar.
    - Check if any HTML form has abnormal actions.
- Features related to **domain obfuscation**:
    - Number of “-“ characters in hostname part of URL
    - URL is an IP address.
    - Check if the TLD (Top-level domain) or ccTLD (country code TLD) is used as part of the subdomain.
    - Check if the TLD or ccTLD are used as the path in the URL.
    - Obfuscated HTTPS
    - Check if the most frequent domain name in the HTML source code does not match with the webpage’s URL domain name.
- Features related to **HTML content**:
    - HTML form actions containing URLs without HTTPS protocol.

    - Form action contains a relative URL.
    - Form action contains “#”, “about:blank”, empty string or “javascript:true”.
    - JavaScript code that disables right-clicking capabilities.
    - JavaScript code that launched pop-ups.
    - HTML code containing the HTML “mailto” function.
    - Usage of iframe or frame.
    - Check if the title tag is empty.
    - Check if the form scope contains no text but only images.
- Features related to **symbol exploit**:
    - Number of dots in the URL
    - Number of “-“ in the URL
    - Check if the “@” symbol exists in the URL.
    - Check if the “~” symbol is in the URL.
    - Number of “_”, “%”, “&”, “#” and numeric characters are in the URL (different feature for each).
    - Check if “//” is in the URL.
    - Number of sensitive words (i.e. “secure”, “account”, “confirm” etc.) in the URL.

The results showed that the URL obfuscation creates the best performing classifier.



Feature Category	TP (%)	FN (%)	TN (%)	FP (%)	ACC (%)
Content URL obfuscation	95.27	4.73	96.67	3.33	95.97
Domain obfuscation	92.80	7.20	49.93	50.07	71.37
HTML content	58.00	42.00	94.33	5.67	76.17
Symbol exploit	77.07	22.93	87.00	13.00	82.03
Webpage URL properties	69.07	30.93	83.20	16.80	76.13

Figure 36 Performance of different feature categories on C4.5 classifier

2.5.9 Feature Extraction or Feature Selection for Text Classification: A Case Study on Phishing Email Detection [42]

In this paper a test was performed on which feature dimensionality reduction method yields the best results in terms of classification performance. The classification on which these experiments were completed was phishing email classification. The classifier which was used was a bagging classifier which is a combination of classifiers trained by randomly generated training sets to form a final prediction. As a base classifier the J48 decision tree algorithm was used.

There are two basic categories for dimensionality reduction: feature extraction and feature selection.

- **Feature Extraction**: The original feature space is converted to a more compact new space.
    - **Principal Components Analysis (PCA)**: The idea is to derive new variables that are combinations of the original variables and are uncorrelated. Firstly, the mean of the dataset is calculated and then the covariance matrix of the original attributes is calculated. Then the eigenvectors are extracted. They are a linear transformation

from the original attribute space to a new space in which attributes are uncorrelated. The best n eigenvectors are selected as new features and the rest are discarded.

    - **Latent Semantic Analysis (LSA)**: useful to overcome problems with multiple words with similar meanings and words that have more than one meaning.

- **Feature Selection**: A subset of the original features is selected, and only this subset is used to train the Machine Learning model.
    - **Chi-Square**: This produces a score for each attribute depending on the dependency between the term and the class.
    - **Information Gain**: The amount of information gained about a random variable or signal from observing another random variable. The result is a score for each feature, and the features with the highest scores are chosen.

The model architecture of the paper is the following:

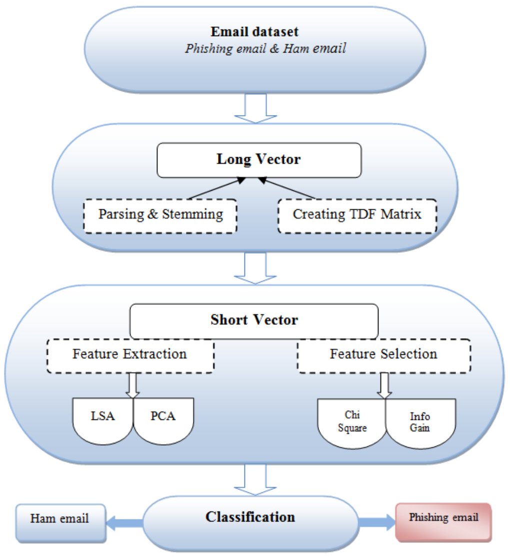


Figure 37 Feature Extraction/Selection Architecture

The steps taken towards the classification of the emails are the following:

1. The **data set is prepared** by collecting emails from public emails corpora, labeled as phishing or legitimate.
2. **Tokenization** is performed to separate the words of the email into an array.
3. **Stop word removal** is applied to remove words that do not have any significant importance in building the classifier.
4. **Stemming** is applied to remove the flexional ending from the words.
5. The **Term-Document-Frequency (TDF)** matrix is created where the rows represent different examples (emails) and the columns represent different terms (words) and their frequency that they appear in the corresponding example.
6. **Dimensionality reduction** is applied to each row, which is practically a vector, from the aforementioned possible techniques.
7. **Classification** is done on the emails.

The paper's model considers the whole email itself, so the feature vector contains different kinds of features like header-based, URL-based and body-based features. The performance results show that there is a maximum performance at about 1500 features.

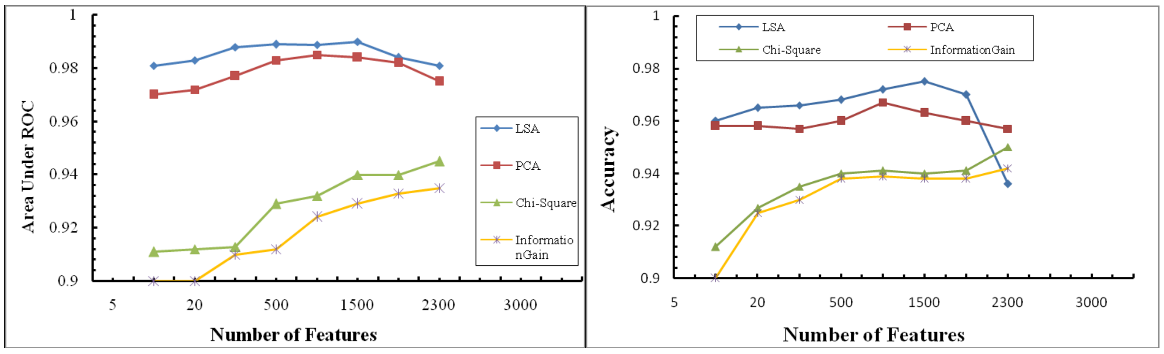


Figure 38 Feature Dimensionality Reduction Performance Results

2.5.10 Using Syntactic Features for Phishing Detection [43]

This paper reports on the comparison of the subject and object of verbs in their usage between phishing email and legitimate emails. The email data sets used are one phishing from 2005, one phishing from 2014 and one legitimate. The purpose of using two different phishing email data sets was to perform comparisons between phishing emails throughout the years as well as between phishing and legitimate emails in general.

This paper attempts to find if there is a difference in the subject and objects of the same verbs between phishing and legitimate emails. While there has been research related to using the words themselves for phishing classification, this should provide additional insight and a more sophisticated classification. For example, in the case of the verb "update", there could be the following two cases depending on the legitimacy of the email:

- Legitimate email: "I'll update the document."
- Phishing email: "Update your account."

The first experiment calculated the syntactic similarity using the parse tree paths. For example in the figure below, the parse tree path from the frame element 'He' to the target word 'ate' is represented as ↑VB↑VP↑S↓NP.

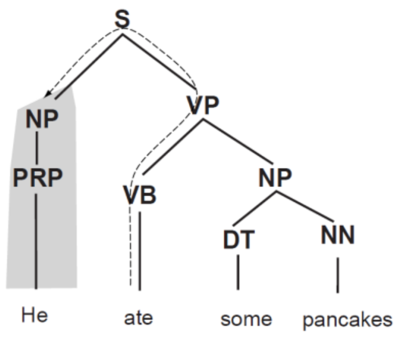


Figure 39 Parse tree path from 'He' to 'ate'

The abbreviations used are part of part-of-speech tags. A few of them are:

- PRP: personal pronoun
- NP: noun phrase
- S: sentence
- VB: verb
- VP: verb phrase
- DT: determiner
- NN: common noun

The top parse tree paths for various verbs were explored. As a similarity measure for the created parse tree paths from the phishing and the legitimate corpus the cosine similarity coefficient was used. This metric works by plotting the two sets which are to be compared as vectors, and the cosine of the angle between them is equal to the similarity metric. The results showed that the syntactic structures were not drastically different between the two types of emails. Therefore, the syntactic structure of a sentence was not a good indicator.

The next experiment compared the subjects and objects of target verbs between sentences in the phishing and legitimate data. The idea is that the subjects and objects used for the same verbs are different for the two different categories of emails.

The most frequent subjects between the two corpora were relatively similar. The distribution of the subjects showed that the most frequent subjects were even more frequent in the phishing emails than in the legitimate emails.

Unlike the subjects, the most frequent objects between the two corpora were all different except for the verb “click”.

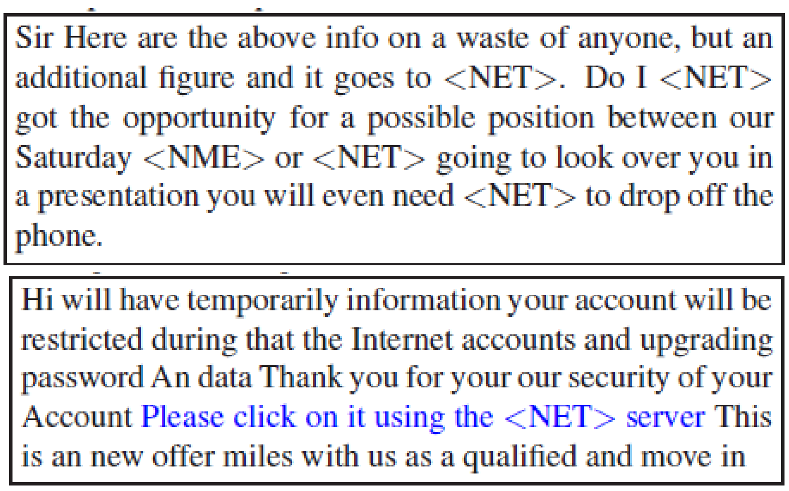
The same experiment was concluded between the phishing data sets of 2005 and 2014 to explore any differences that may have happened throughout the ages. Most verbs have similar subject and objects between the two data. It was concluded that the patterns in phishing emails have not been substantially altered in terms of text contents.

Concluding, the similarity scores were not consistent, and their characteristics seemed to depend on the verbs themselves.

2.5.11 Automated email Generation for Targeted Attacks using Natural Language [44]

This paper proposes a system for email masquerading attacks using Natural Language Generation (NLG) techniques. The proposed deep learning system uses legitimate and a varying amount of malicious content to generate fake emails with malicious content. The deep learning model is a Recurrent Neural Network (RNN) for automated text generation. The focus is on the performance of these generated emails against statistical detectors.

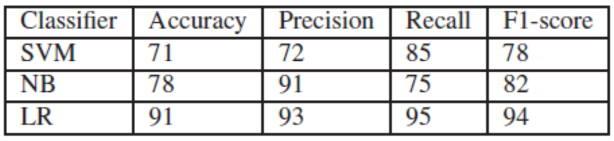
The email texts from both legitimate and phishing emails were used to train the model. The emails were cleaned by removing special characters and replacing URLs, email addresses and names with <LINK>, <EID> and <NET> respectively. For that purposes the NLTK NER (Natural Language Toolkit and Name Entity Recognition) Python library was used. Different data sets were used in terms of the percentage of phishing emails in them to train the neural network (5%, 10%, 30% and 50% phishing emails). Two examples of created emails are the following:


Sir Here are the above info on a waste of anyone, but an additional figure and it goes to <NET>. Do I <NET> got the opportunity for a possible position between our Saturday <NME> or <NET> going to look over you in a presentation you will even need <NET> to drop off the phone.

Hi will have temporarily information your account will be restricted during that the Internet accounts and upgrading password An data Thank you for your our security of your Account Please click on it using the <NET> server This is an new offer miles with us as a qualified and move in

Figure 40 Emails generated using RNN

The blue colored letters represent the injected malicious part. These examples show that there are small substrings which make sense, but the whole string does not make much sense. While they do not make sense to a human, statistical classifiers were used to classify the generated emails, and the results were the following:



Classifier	Accuracy	Precision	Recall	F1-score
SVM	71	72	85	78
NB	78	91	75	82
LR	91	93	95	94

Figure 41 Statistical Classifiers on Generated Emails

2.5.12 Catching the Phish: Detecting Phishing Attacks using Recurrent Neural Networks (RNNs) [17]

This paper uses Recurrent neural networks and the email's text structure to classify emails as phishing or not. The reason for using RNNs is that they are specialized NNs for processing sequential data and account the order of the data. In the case of emails, the order of the words in the text will be taken into account thanks to the RNNs. The paper proposes a sophisticated word tokenization as the first step towards classification:

1. **Lowercase** all the characters
2. Remove all characters that the RFD 3986 standard does not allow in URLs, i.e. only keep the unreserved a-z, 0-9, -, ., _, ~ and the reserved :/?#[]@!$&'()*+,;= and %. But not " which is used in emails.

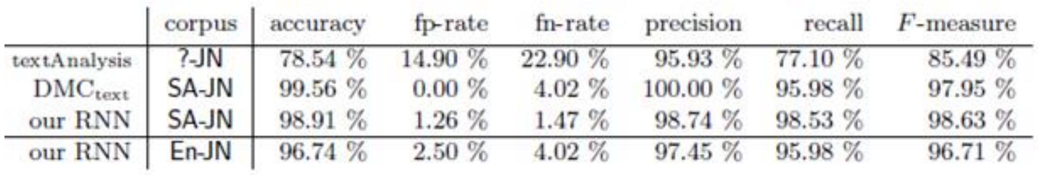
3. URLs starting with http:// or https:// are replaced by the indicator word <url>. URLs starting with www are replaced by <www>. Three or more consecutive non-alphanumeric characters are replaced by <threespecial>.

The final text includes only lowercase alphanumeric words and tokens.

The size of the dictionary used by the model is limited to the 4995 most common words in the data sets' emails. Stemming is done using the Snowball Stemmer. Five more tokens are added: <unkalpha>, <unknum>, <unk> for unknown words to the dictionary, such as those that consist of only alphabetical or numerical values, or fit none of the first two, respectively.

The other two tokens which are introduced are <cuts> and <cute>, which represent the "cut start" and "cut end". These tokens are used for too long emails, the length of which causes training inefficiency. The solution that is proposed by the paper is to keep the beginning, most middle and ending parts of the emails. For the model to understand the anomaly that is introduced by that transformation these tokens are replacing the parts where the text was removed.

The performance of this architecture is compared with the textAnalysis sub-classifier of the PhishNet-NLP email classifier by Verma et al. [45] and the Dynamic Markov Chain (DMC) model proposed by Bergholz et al. [46] and the results are the following:



	corpus	accuracy	fp-rate	fn-rate	precision	recall	F-measure
textAnalysis	?-JN	78.54 %	14.90 %	22.90 %	95.93 %	77.10 %	85.49 %
DMC_{text}	SA-JN	99.56 %	0.00 %	4.02 %	100.00 %	95.98 %	97.95 %
our RNN	SA-JN	98.91 %	1.26 %	1.47 %	98.74 %	98.53 %	98.63 %
our RNN	En-JN	96.74 %	2.50 %	4.02 %	97.45 %	95.98 %	96.71 %

Figure 42 Results of paper's model compared to related work

2.5.13 A Survey of Phishing Email Filtering Techniques [47]

This survey explored related work in filtering techniques against phishing emails. Amongst the different techniques, a few machine learning based filters were mentioned. These are the following:

- Bergholz et al. [48] proposed a classifier with features obtained from a dynamic Markov chain and class-topic models. The features extracted from the emails are from the following categories:
    1. Four structural (total number of body parts)
    2. Eight link features (total number of links)
    3. Four element features (HTML or JavaScript code)
    4. Two spam-filter features (passing the email through established spam filter and having their output as a feature)
    5. Nine word list features (words: account, update, confirm, verify, secure, notify, log, click, inconvenient)

  The second step is the features retrieval by the dynamic Markov chain, which depend on the likelihood of capturing a message belonging to a specific class. Third, 50 latent topic model features are retrieved (clusters of words appear together in emails).

- Ma, Ofoghi et al. [49] proposed a model which uses only seven features after ranking many features using the information gain algorithm. These features represent the most powerful ones. The steps followed to build the model are the following:
    1. A feature generator which includes the seven features
    2. A machine learning selection method by an adaptive five-machine learning algorithm
    3. Information gain is created by induction
    4. A feature evaluation method which selects a smaller vector space of features
    5. A refined feature matrix to optimize feature sets

  Decision trees algorithm was used as a classifier and the accuracy reached was equal to 99.8%.
- Castillo et al. [50] proposed a hybrid system called FRALEC. A hybrid system are multi-layered systems, which are based on combining different algorithms together to enhance results. The system consists of the following:
    1. A Naïve Bayes classifier for textual content of emails
    2. A rule-based classifier for separation of non-grammatical features of emails into three categories: fraudulent, legitimate and suspicious
    3. An emulator-based classifier that classifies the responses from URL websites.
        - The email is assigned as phishing if the body of a previously legitimate-classifier email includes forms.
        - The email is assigned as legitimate if the body of a legitimate-classifier email does not include forms, links or images.

  The model was tested against 1038 emails, 10 of which were legitimate and the rest phishing, and a precision of 96% was achieved.
- Islam et al. [51] proposed a multi-tier classification method. The email features are extracted and classified in a sequential fashion by an adaptive multi-tier classifier, and the outputs are sent to a decision classifier. If the message is misclassified by any of the tiers of the classifiers, the third tier will make the final decision in the process. The models used in the three tiers were SVM, AdaBoost and Naïve Bayes. The model achieved an accuracy of 97%.

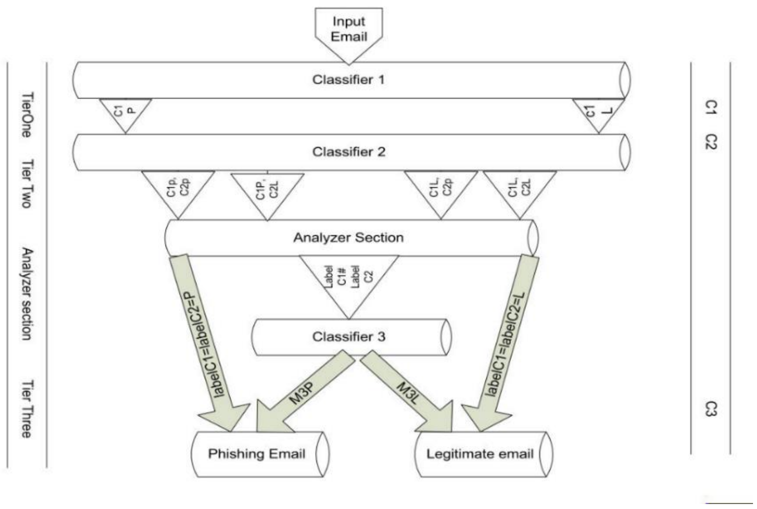


Figure 43 Multi-tier classification method

- Almomani et al. [52], proposed a framework called Phishing Evolving Neural Fuzzy Framework (PENFF) for detecting and predicting zero day phishing emails. It is based on adaptive Evolving Fuzzy Neural Network (EFuNN) [53]. The architecture is shown below.

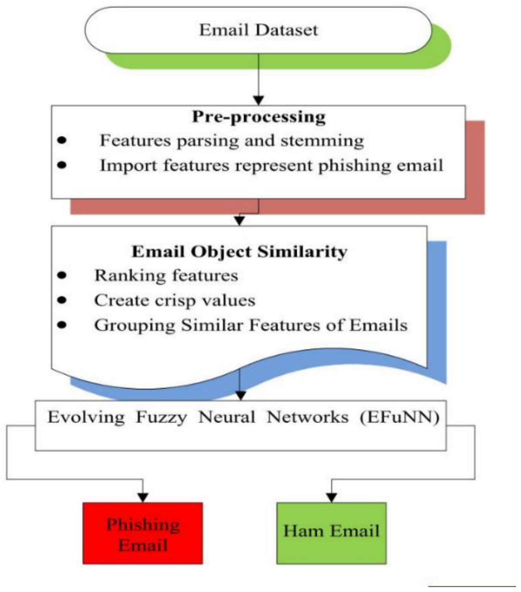

Email Dataset
Pre-processing
Features parsing and stemming
Import features represent phishing email
Email Object Similarity
Ranking features
Create crisp values
Grouping Similar Features of Emails
Evolving Fuzzy Neural Networks (EFuNN)
Phishing Email
Ham Email

Figure 44 Phishing Evolving Neural Fuzzy Framework (PENFF)

- Almomani et al. [54] [55] proposed a framework called PDENFF (Phishing Dynamic Evolving Neural Fuzzy Framework) that adapts the "evolving connectionist system". It is an improvement on the PENFF and has achieved a 3%-13% improvement compared to the previous work. The architecture is shown in the figure below.

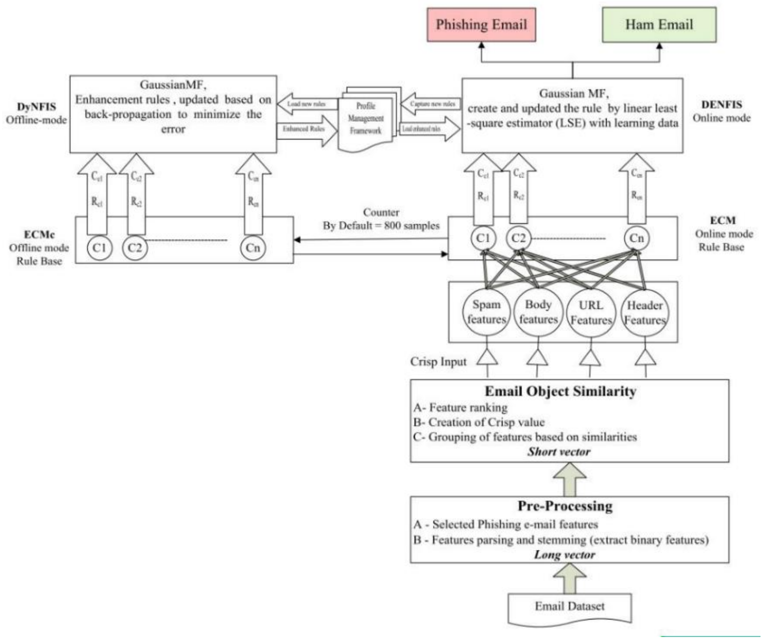

Phishing Email
Ham Email
DyNFIS Offline-mode
GaussianMF, Enhancement rules , updated based on back-propagation to minimize the error
Load new rules
Enhanced Rules
Profile Management Framework
Capture new rules
Load enhanced rules
Gaussian MF, create and updated the rule by linear least -square estimator (LSE) with learning data
DENFIS Online mode
ECMc Offline mode Rule Base
Counter By Default = 800 samples
ECM Online mode Rule Base
C1
C2
Cn
Spam features
Body features
URL Features
Header Features
Crisp Input
Email Object Similarity
A- Feature ranking
B- Creation of Crisp value
C- Grouping of features based on similarities
Short vector
Pre-Processing
A - Selected Phishing e-mail features
B - Features parsing and stemming (extract binary features)
Long vector
Email Dataset

Figure 45 Phishing Dynamic Evolving Neural Fuzzy Framework (PDENFF)

Having a thorough understanding of the theory behind machine learning classifiers and having presented several relevant works, the next step towards building the proposed hybrid classifier is a brief but clear problem statement and the approach that it is attempted to be solved.

3 Problem Statement

As mentioned in the previous chapter, there are already established countermeasures against phishing emails including:

- User education
- Search-engine-based techniques
- List based anti-phishing solutions
- Visual-similarity-based solutions
- Machine Learning techniques

When it comes to the machine-learning-based tools and relevant research work, there are two fundamental categories which are used as input features of the machine learning classifier:

- **Properties**: This category includes any characteristic of the email which has a specific value, e.g. number of URLs in the email, content transfer encoding of the email, number of attachments, whether the email has any part with HTML etc.
- **Text**: This category includes every part of text of the email that is to be read by the receiver of the email, or the body of the email. This does not include any header field value or the contents of any webpage the URL of which is in the email. It only includes the text that is to be presented as plain text as the email body.

Although there is relevant work for classifying emails as legitimate or phishing using machine learning techniques for both properties-based and text-based solutions, there is no work found which combines the two.

It is attempted in this work to build a hybrid model using both feature categories together as input. The purpose of experimenting with a hybrid architecture is to test whether there is improvement of the performance compared to simple classifiers when it comes to detecting phishing emails.

In order to build a hybrid classifier of this type, the following directions were studied:

- **Emails**: In order to understand the email and properly retrieve both the properties-based and the text-based features, a proper understanding of how an email is structured, its components and different characteristics is required. The structure of the email was studied thoroughly and presented in this thesis. Afterwards the data sets which include legitimate and phishing emails were discovered from the relevant work and the web.
- **Machine Learning**: After learning the email's components, the next step is to start building an algorithm to retrieve the best features and a proper machine learning classifier which will detect phishing emails in the most efficient and best performing manner. For that purpose, different machine learning tools are presented as well as criteria as to which one to choose and the one that was chosen. For the chosen tool there is detailed information for what it provides and how it can be used for the purposes of phishing email classification.
- **Architecture**: On a higher level, the architecture which will be used for involving both the properties-based and the text-based features was searched upon. Two architectures were tested, and they are called "assembled" and "stacked" hybrid architectures. They are presented in the following chapters.
- **Performance**: To thoroughly test whether there is an improvement in performance in classifying emails as phishing or legitimate, all the combinations of feature inputs, machine

learning classifiers and their corresponding hyperparameters were tested, as well as the different proposed architectures.

The findings of these directions are presented in the following approach and implementation chapters. The results are then presented in the numerical results chapter and the conclusion and potential future work are presented next.

4 Approach

In this chapter the approach method will be presented, including the steps taken to determine the email data sets that were chosen, the classification procedure of the emails as phishing or legitimate, the criteria that were used to choose the most suitable for this work's purposes machine learning tool of the literature. Lastly, a high-level description of a generic machine learning training and testing architecture will be presented.

4.1 Email Structure

The main focus of this project is to classify emails as phishing or benign. The first step towards the classification of an email is a proper preprocessing process, which will analyze the email and retrieve all the useful information which will be needed by a machine learning classifier in order for it to identify emails with the minimum error rate. In order to properly preprocess the email, in this part of the thesis the fundamental and needed characteristics of an email will be presented.

The following text is a simple form of an email as a plain text.

```
From: Mendez, Nicole
Sent: Friday, March 30, 2001 10:51 AM
To: Rance, Susan
Subject: RE: REMINDER: Please fill our your time sheets today


Great.  Thanks.  Here's how:

Tools
Options
Mail Format

Send in this message format:    HTML
(Under Stationery & Fonts)
    Use this stationery by default (use dropdown to pick your stationery)

Nicole Mendez
```

Table 2 Email Example

The Internet Message Format (IMF) is a syntax for text messages that are sent between computer users as "electronic mail" messages (emails) and is defined by RFC 5322 (Request For Comments) standard [56]. Non-ASCII data and multimedia content attachments are defined in RFC 2045-2049. In total they constitute the Multipurpose Internet Mail Extensions (MIME).

Emails are made up of two sections, the message's headers and the message body. The email headers are written in a specific format. They are lines beginning with a field name, followed by a colon (":"), followed by a field body (a value for the field name) and terminated by CRLF (Carriage Return – ASCII value of 13 or "\r", Line Feed / New Line – ASCII value of 10 or "\n"). In the email shown before a header example is the "Sent" header field, followed by the colon and the value of the date that the email was sent at. The body of a message is written after the headers and simply includes US-ASCII characters, with the following limitations: The CR and LF characters must be written only together, the lines of characters must be limited to 998 characters and should be limited to 78, excluding the CRLF. The aforementioned MIME documents extend this specification to allow for different sorts of message bodies.

Multipart emails are messages with multiple parts, in which one or more different sets of data are combined in a single body. In this case a "multipart" Content-Type filed must appear in the entity's header. The body will contain one or more "body parts", with each one having a preceding and a closing boundary. The multipart emails' parts follow a tree structure, where there is a root part which branches into other parts which could either be leaf nodes or be roots for other multiple parts. A multipart email's structure is shown in the following figure.

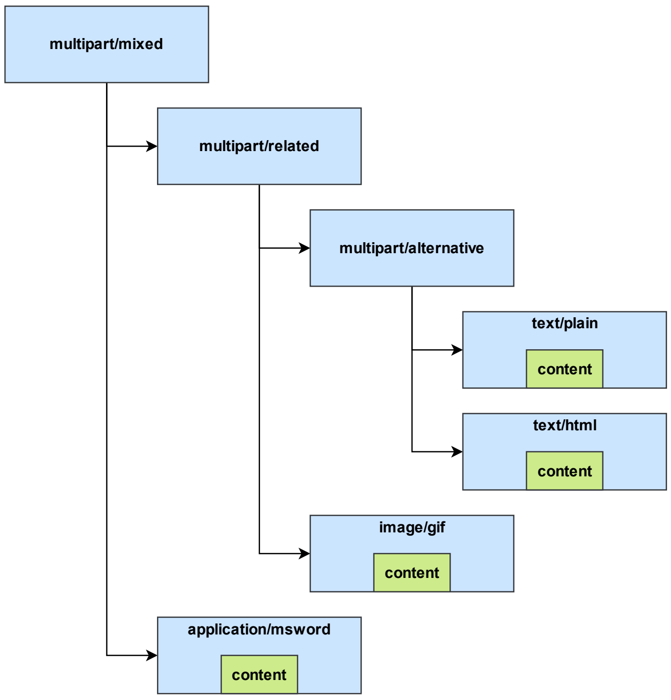


Figure 46 Multipart email structure

An example of a multipart email is shown in the following [57].

```
// Example Multipart Email:
From: sender@example.com
To: recipient@example.com
Subject: Multipart Email Example
Content-Type: multipart/alternative; boundary="boundary-string"
```

```
--your-boundary
Content-Type: text/plain; charset="utf-8"
Content-Transfer-Encoding: quoted-printable
Content-Disposition: inline

Plain text email goes here!
This is the fallback if email client does not support HTML

--boundary-string
Content-Type: text/html; charset="utf-8"
Content-Transfer-Encoding: quoted-printable
Content-Disposition: inline

<h1>This is the HTML Section!</h1>
<p>This is what displays in most modern email clients</p>

--boundary-string--
```

Table 3 Multipart Email

4.1.1 Email clients (mail user agent)

An email client, email reader or maul user agent (MUA) is a computer program used to access and manage a user's email. There are web applications and specific computer hardware or software pieces with a primary role of working as email clients.

In an email the header which states the program that was used to draft and send the original email is the "X-Mailer" header.

4.1.2 Email Headers

As mentioned before, the email headers are written in a specific format and determine a certain characteristic of the email. They can be interpreted as variables of the email. There is a wide plethora of different email headers and a user can create custom headers for their specific needs.

The **"Content-Type" header** indicates the media type of the message content. The header value consists of a type and a subtype in the format "type/subtype" (e.g. "text/plain" or "text/html"). This mechanism supports the following types:

- **multipart**: This is the type which is used as a root when there are other parts of the email. This part must be included in any case where the email has multiple parts as the root part of the email. Multiple multipart parts can be used in one email to complicate the tree structure. The subtypes are the following:

  ```
  multipart/mixed (the primary subtype)
  multipart/alternative (used for representing the same data in multiple formats)
  multipart/related (used by MHTML (HTML mail))
  multipart/parallel (for parts which are to be presented simultaneously)
  ```

multipart/digest (for multipart parts in which each part is of type “message)

- **text**: This type includes any type of textual information. The base subtype is “text/plain” which includes unformatted text. The other subtypes are used for enriched text, where an application could enhance the appearance of the text. The subtypes are:

text/css
text/csv
text/html
text/javascript (obsolete)
text/plain
text/xml

- **message**: This type is an encapsulated message. A part of type “message” is an RFC 822 conformant message and may include its own different type header field. The subtypes are:

message/rfc822 (primary subtype)
message/partial (used for partial message, for sending messages in parts)
message/external-body (for defining large bodies by reference to an extrernal data source)

- **image**: As this name indicates, this type includes image data. To view the information, a display device is need (monitor, printer etc.). The subtypes include image file formats:

image/jpeg
image/gif
image/png
image/bmp
image/svg+xml

- **audio**: This type includes an audio file. The primary subtype is “audio/basic” and more complex subtypes include the file format, such as “audio/mpeg”. Audio requires an audio device to reproduce the contents.

- **video**: This type includes a video file. The primary subtype Is “video/mpeg”. Video requires a display device to display the contents.

- **application**: This includes data that cannot be included in the other types. A typical type of data is uninterpreted binary data or other data which are to be processed by a mail-based application. Other uses include spreadsheets and data for mail-based scheduling systems. The application subtypes are the following:

application/octet-stream (for uninterpreted binary data)
application/ODA
application/PostScript

The **“Content-Disposition” header** describes the that a mail message is presented. A MIME part can have:

- **Inline content-disposition**: the message part should be automatically displayed when the message is displayed.
- **Attachment content-disposition**: the part is not displayed automatically but requires action to open it (traditional attachment which has to be clicked to either open/show or download).

The **"Content-Transfer-Encoding" header** defines the method for representing the binary data that an email (and any file for that matter) has.

If there is some sort of binary to text encoding used for the email, this header defines what that transformation algorithm was used, so that the mail client can decode it and present the information of the email. If there is not any transformation, this header provides a descriptive label for the format of the content.

The most widely used values of the Content-Transfer-Encoding header, and correspondingly the most widely used encodings of emails are:

- **7bit**, which is the traditional format of email encoding. The format allows up to 998 characters per line of the ASCII number range from 1 to 127, with the CRLF characters appearing only together and only at the end of the line.
- **quoted-printable**: This encoding is used to encode arbitrary byte sequences so that the rules of 7bit encoding are satisfied. This encoding is useful when most of the characters are of US-ASCII type and none to a few characters are outside that range, and the purpose is to convert the sequence of bytes to a mostly human readable form.
- **base64**: Used to encode bytes to a form that satisfies the 7bit rules. This encoding is more useful for bytes which do not consist of US-ASCII characters (non-text 8 bit and binary data).
- **8bit**: This is for use with SMTP servers that support the 8BITMIME SMTP extension, is the same type as the 7bit. The reason of the 8bit's existence is that, prior to it, the original SMTP supports transport of only textual data. Therefore, any binary data needed to be encoded into a text. The 8bit encoding was developed to alleviate this limitation.
- **binary**: This is any sequence of bytes.

Other email headers which do not require detailed analysis are listed in the following extensive table [58].

Table 4 Mail Header Fields

Header	Description
Data	Message date and time
From	Mailbox of message author
Sender	Mailbox of message sender
Reply-To	Mailbox for replies to message
To	Primary recipient mailbox
Cc	Carbon-Copy recipient mailbox
Bcc	Blind-carbon-copy recipient mailbox
Message-ID	Message Identifier
In-Reply-To	Identify replied-to message(s)
References	Related message identifier(s)

Subject	Topic of message
Comments	Additional comments about the message
Keywords	Message key words
Resent-Date	Data and time message is resent
Resent-From	Mailbox of person for whom message is resent
Resent-Sender	Mailbox of person who actually resends the message
Resent-To	Mailbox to which message is resent
Resent-Cc	Mailbox(es) to which message is cc'ed on resend
Resent-Bcc	Mailbox(es) to which message is bcc'ed on resend
Resent-Reply-To	Resent reply-to
Resent-Message-ID	Message identifier for resent message
Return-Path	Message return path
Received	Mail transfer trace information
Encrypted	Message encryption information
Disposition-Notification-To	Mailbox for sending disposition notification
Disposition-Notification-Options	Disposition notification options
Accept-Language	Language(s) for auto-responses
Original-Message-ID	Original message identifier
PICS-Label	PICS rating label
Encoding	Message encoding and other information
List-Archive	URL of mailing list archive
List-Help	URL for mailing list information
List-ID	Mailing list identifier
List-Owner	URL for mailing list owner's mailbox
List-Post	URL for mailing list posting
List-Subscribe	URL for mailing list subscription
List-Unsubscribe	URL for mailing list unsubscription
Message-Context	Type or context of message
DL-Expansion-History	Trace of distribution lists passed
Alternate-Recipient	Controls forwarding to alternate recipients
Original-Encoded-Information-Types	Body part types in message
Content-Return	Return content on non-delivery
Generate-Delivery-Report	Request delivery report generation
Prevent-NonDelivery-Report	Non-delivery report required
Obsoletes	Reference message to be replaced
Supersedes	Reference message to be replaced
Content-Identifier	Message content identifier
Delivery-Date	Message delivery time
Expiry-Date	Message expiry time
Expires	Message expiry time
Reply-By	Time by which a reply is requested
Importance	Message importance
Incomplete-Copy	Body parts are missing
Priority	Message priority
Sensitivity	Message content sensitivity
Language	X.400 message content language
Conversion	Conversion allowed?

Conversion-With-Loss	Lossy conversion allowed?
Message-Type	Message type: delivery report?
Autosubmitted	Automatically submitted indicator
Autoforwarded	Automatically forwarded indicator
Discarded-X400-IPMS-Extensions	X.400 IPM extensions discarded
Discarded-X400-MTS-Extensions	X.400 MTS extensions discarded
Disclose-Recipients	Disclose names of other recipients?
Deferred-Delivery	Deferred delivery information
Latest-Delivery-Time	Latest delivery time requested
Originator-Return-Address	Originator return address
X400-Content-Identifier	X400-Content-Return
X400-Content-Type	X400 content type
X400-MTS-Identifier	X400 MTS-Identifier
X400-Originator	
X400-Received	
X400-Recipients	
X400-Trace	

Permanent MIME Header fields are the following.

Table 5 Permenent MIME Header Fields

Header	Description
MIME-Version	MIME version number
Content-ID	Identify content body part
Content-Description	Description of message body part
Content-Transfer-Encoding	Content transfer encoding applied
Content-Type	MIME content type
Content-Base	Content-Location
Content-features	Indicates content features of a MIME body part
Content-Disposition	Intended content disposition and file name
Content-Language	Language of message content
Content-Alternative	Alternative content available
Content-MD5	MD5 checksum of content
Content-Duration	Time duration of content

4.1.3 Email File Formats

There are many different file formats when it comes to storing emails**.** The most popular file formats are the following [59]:

- **EML** (E-Mail Message File): emails saved using Outlook and other relevant applications. Microsoft Outlook is the default software for opening EML files.
- **VCF** (Virtual Card Format) is a digital file format for storing contact information.

- **TNEF** (Transport Neutral Encapsulation Format) is a Microsoft-based format for encapsulating email attachments based on the Messaging Application Programming Interface (MAPI).
- **PST** (outlook Personal Storage File) is a file that stores a variety of user's information.
- **OST** (Offline Storage File) consist of a user's mailbox data on a local machine when registered with Exchange Server using Microsoft Outlook.
- **EMLX** is a file format implemented and developed by Apple. The Apple mail application uses the EMLX file format for exporting the emails.
- **MBOX** (Mail box) is a file format for collection of emails. The emails are stored inside the container along with their attachments. The MBOX file is practically a text file with emails, one written after another and are separated by a few new lines. In this file a new email is started right the "From" header field, and that is the line that applications and programs are searching when they access and MBOX file. This project operates on email data sets with mbox format.
- **Maildir** (mdir) is a format for storing emails, where each message is kept in a separate file with a unique name and each mail folder is a file system directory. The Maildir directory usually has three subdirectories: tmp, new and cur. The maildir file format structure is show in the following figure. A few data sets found in the literature of machine learning against phishing emails were of this file format but were converted to mbox for concistency.

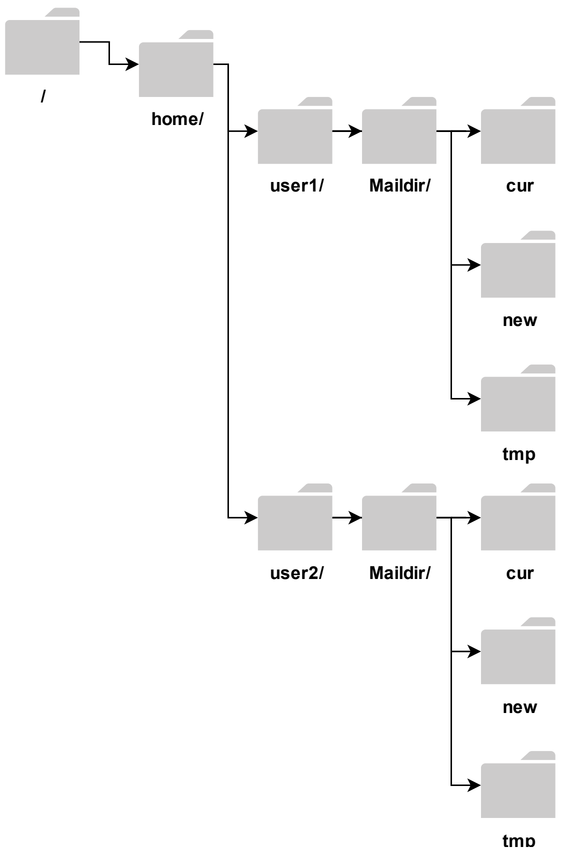


Table 6 Maildir Email File Format

4.1.4 Email Data Sets

The first step towards building a machine learning classifier is having a proper data set. After research in relevant work, the following data sets were found:

- **Spam Assassin public mail corpus** [60]: Spam Assassin is an open source anti-spam platform that gives a filter to classify email and block spam [61]. The data set provided by Spam Assassin includes legitimate and spam emails with the purpose of building classifiers for spam email classification and, more generally, to test spam filtering systems. Even though the purpose of this project is not to classify spam emails but rather detect phishing emails, this data set's legitimate emails can be of use for building an email data set which will include phishing and legitimate emails.
- **Jose Nazario Phishing email Corpus** [62]: This data set includes phishing emails in different mbox format files. The emails are classified by the creator of the data set and they are emails which were sent to them. It is not meant to be an exhaustive representation of phishing emails, but rather a representation.
- **Enron Email Dataset** [63]: Enron was one of the largest integrated natural gas and electricity companies in the world [64]. Before its bankruptcy in 2001, Enron employed 29,000 employees with a revenue of approximately $101 billion during 2000. At the end of 2001 it was revealed that Enron's reported financial condition was sustained by systematic and planned accounting fraud, known since then as the Enron scandal. Because of the scandal, all the emails of the corporation were made public by the Federal Energy Regulatory Commission during its investigation for said scandal. This data set contains data from about 150 users, mostly senior management of Enron, organized into folders. The corpus contains about 500,000 emails in total. These emails are used by the literature as a legitimate email data set for machine learning algorithm training and testing purposes.
- **Data sets with already retrieved features**: There are two data sets which were used with their features already retrieved. These are:
    - Diego Ocampo Machine learning phishing data set [65]: This GitHub project includes a process of training and testing different machine learning algorithms on phishing emails. It has an already built data base of phishing and legitimate emails, with twelve features for each one.
    - UCI Spambase Data Set [66]: The University of California, Irvine (UCI) Machine Learning Repository provides a plethora of different data sets for building machine learning and intelligent systems. The UCI Spambase data set is an already built data set that contain spam and legitimate emails. Although its purpose is towards spam email filtering, the data set can be used as a quick test for other applications. It contains 57 features, including word-type features and properties-type features.

4.2 Machine Learning Tools in the Literature

To develop a machine learning algorithm for classifying emails as phishing or not, there needs to be a relevant tool for such purpose. The tools used in the literature for training and testing machine learning algorithms are presented in this chapter.

4.2.1 Rapid Miner

Rapid Miner is a data science software platform that provides an environment for all the steps towards data science: data preparation, machine learning, deep learning, text mining and predictive analytics. It supports all the steps of the machine learning process as well as results visualization, model validation and optimization. The RapidMiner Studio has different versions:

- Free Edition is limited to 1 logical processor and 10,000 data rows (examples) in the data set. This edition does not include the more sophisticated capabilities that RapidMiner Turbo Prep, Auto Model and Model Ops provide, which are included in the commercial and the education editions.
- Commercial version pricing starts at $5,000
- Education Licenses are available for certain institutions, which provide the full version for free and educational purposes.

The operation of the Rapid Miner Studio tool is with a graphical user interface, where the user puts boxes and connects them, which represent processes and data pipes respectively, in order to create the whole system process to train, test classifiers and output results. Firstly, this graphical interface will be presented and then the additional features including the Auto Model, Model Ops and Turbo Prep will be presented.

Rapid Miner Studio: To better understand the interface of the Studio, a process example follows.

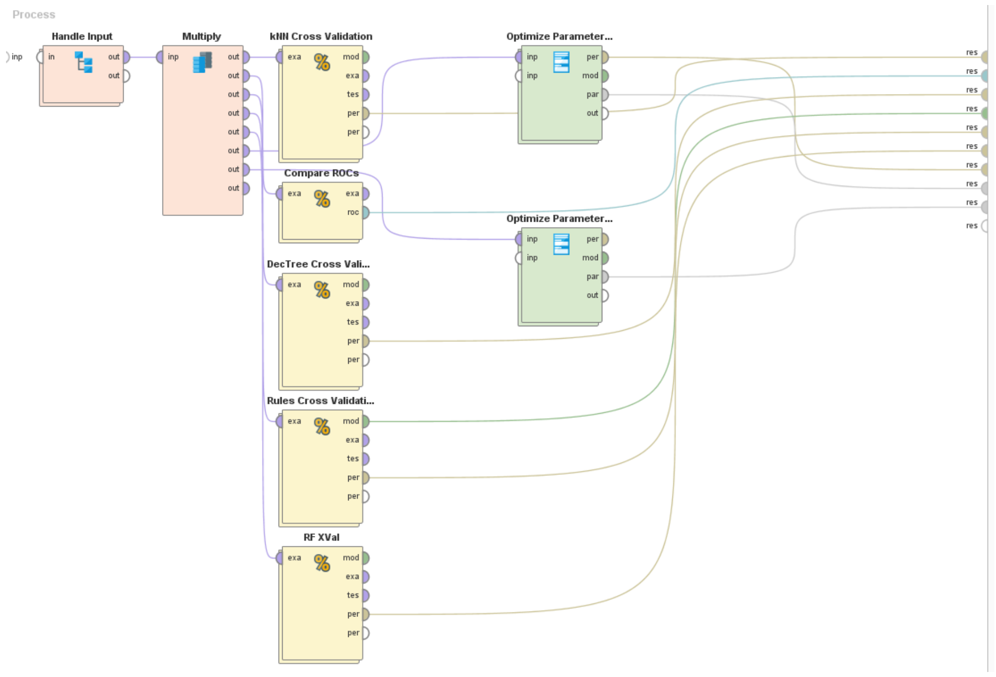


Figure 47 Rapid Miner Studio

As can be seen, there are processes represented as boxes. These processes handle the data which is inputted to them in some way and output the transformed values. The inputs and outputs are represented by the connections between the processes. Note here that Rapid Miner handles already created data sets and does not provide functionality for retrieving features from the data sources (from the emails in the case of this project). The processes of Rapid Miner Studio include the following:

- **Import Data**
- **Filter examples**: Filter out the example set depending on a parameter value.
- **Sort**: Sort by an attribute by giving its name
- **Join**: Joins two input sources
- **Aggregate** attributes
- **Generate Attributes** as combinations of the original ones.
- **Select Attributes** as a subset of the original ones.
- **Replace Missing Values**, with a defined function (e.g. average of the other values)
- **Detect Outlier**, to identify examples which are the farthest away from all the other values. These can be then filtered out because they do not help in training the classifier but rather promote overfitting.
- **Loop values**: Just like in programming, provides a method for looping a process until a condition is met.
- **Branch**: This is the process equivalent to the if-then-else logic.
- **Classifier**: In order to train a classifier, its corresponding process is used and has as input the cleaned data set.
- **Apply Model**: The now trained classifier, as a process, has two outputs: the predictions and the model itself. The model can be used to make predictions on new examples with the "Apply Model" process.
- **Performance**: This process takes the predictions and evaluates the performance of the corresponding trained classifier.
- **Cross Validation** divides the example set into equal parts and rotates through all parts, through all parts, always using one for testing and all others for training the model. At the end, the average of all testing accuracies is delivered as a result.
- **Compare ROCs**: Takes as inputs an arbitrary number of models and produces a single graph with one ROC curve for each model, as shown in the figure below.

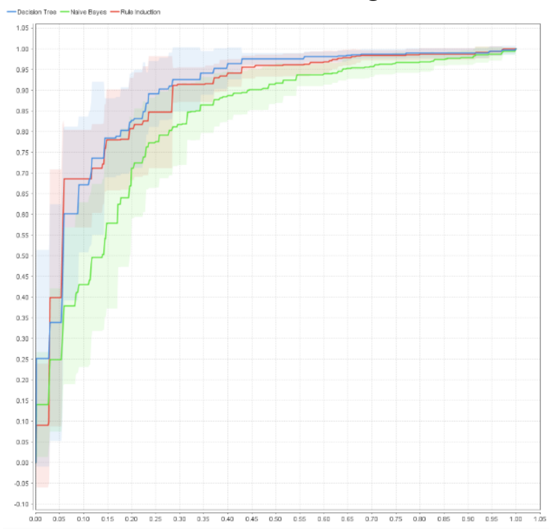


Figure 48 Compare ROCs process of Rapid Miner Studio

A Rapid Miner Studio model was created as a test, using some previous relevant work [65]. The process is the following.

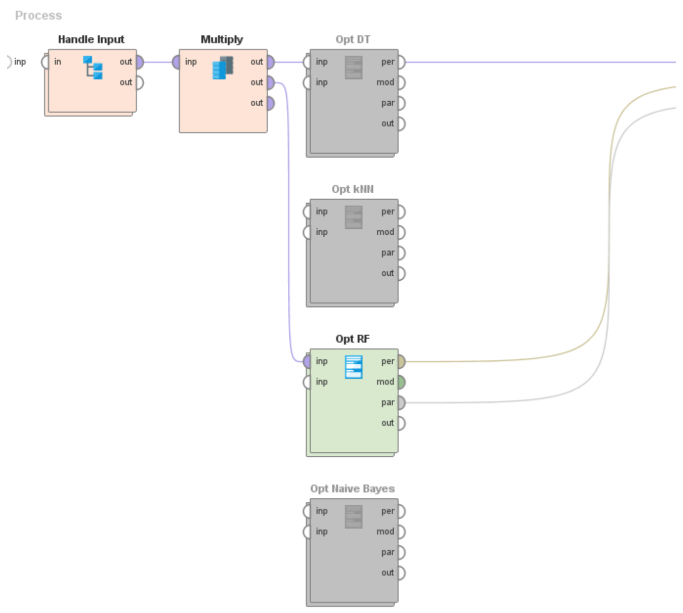


Figure 49 Rapid Miner Studio created process

The data set was the already created csv file from the aforementioned relevant work. There is the "Handle Input" Process which will take the csv data and prepare it for training and testing, the "Multiply" part is just a system to get multiple outputs of the same input, and the "Opt RF" is a Process which trains and tests a Random Forest model and optimizes (hence the Opt) its parameters to maximize the performance. Note here that the other boxes are grayed out because they were inactive to test one of them at a time.

The "Handle Input" process, when opened, has the following process in it.

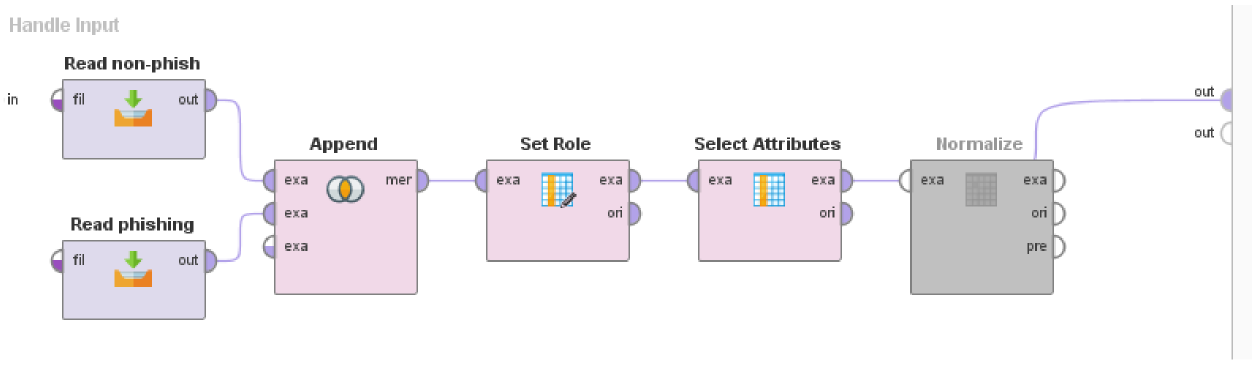


Figure 50 "Handle Input" Process

First the csv files which have the attributes are read and appended so we have one csv file with all the emails (both non-phishy and phishy) totaling at 4511 emails at a ratio of 50 phishing - 50 non-phishing. Then the "Set Role" box sets a role for a specified attribute/column. In our case, we set the attribute "Phishy" as "label", which is the column that label column and determines whether the email is phishing or not. This is used when training and testing the models to let them know that this is the attribute that we want to train for. By using all the other attributes, we want to predict what value the "Phishy" attribute will have. Then the "Select Attributes" just removes the attribute (column) of the unique identifier which was provided to each email (which is useless for training). The Normalize process, as the name indicates, normalizes the values of selected attributes so that they fit in a specific range. It did not change the results at all, so it is disabled.

Opening the “Opt RF” process, we have the following.

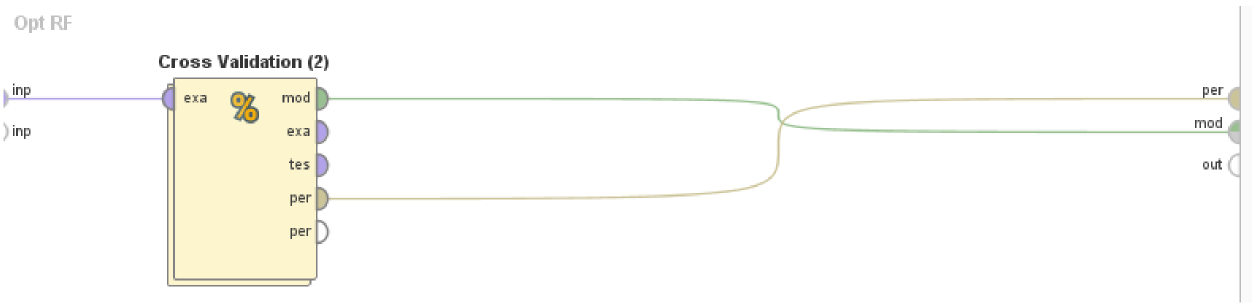


Figure 51 "Opt RF" process

We just have a “Cross Validation” process. Inside the Cross Validation we have the following.

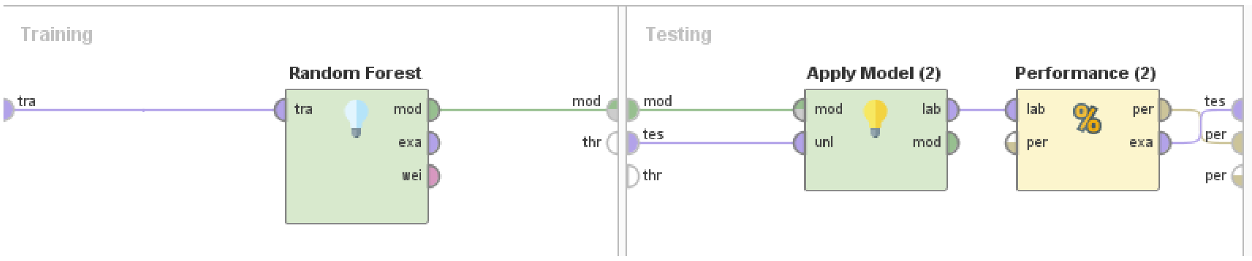


Figure 52 "Cross Validation" Process

For the Cross-Validation process, there is a parameter called “Number of folds”. We put, for example, a value of 10. What happens is that the data will be split at a ratio of 90-10 and the model that is inside the Cross-Validation process will be trained with the 90% and then tested with the 10%. This will be done 10 times, with each time using a different 10% of the data. This is done to have a more general performance information, removing the possibility of the first 10% of the data being too easy or too hard to predict.

Getting a bit more specific inside the Cross Validation, as it can be seen, there is a “Training” sub-process and a “Testing” sub-process. Firstly, the Training sub-process is run using the 90% of the data. In this case, a Random Forest prediction model is used. After the model is trained, the output of the “Random Forest” process called “mod” is the trained Model, which is then used as input at the “Apply Model” together with the 10% of the data. At the Apply Model process, the model is used to predict the outcome of the 10% of the data and afterwards it is sent to the “Performance” process, where the performance is checked there. We get information about the accuracy, the recall and the precision of the model.

As shown before, the Cross-Validation process is inside an Optimization process. This means that all this will be run for every combination of parameters specified at the “Opt RF” process, and the best performance combination will be selected.

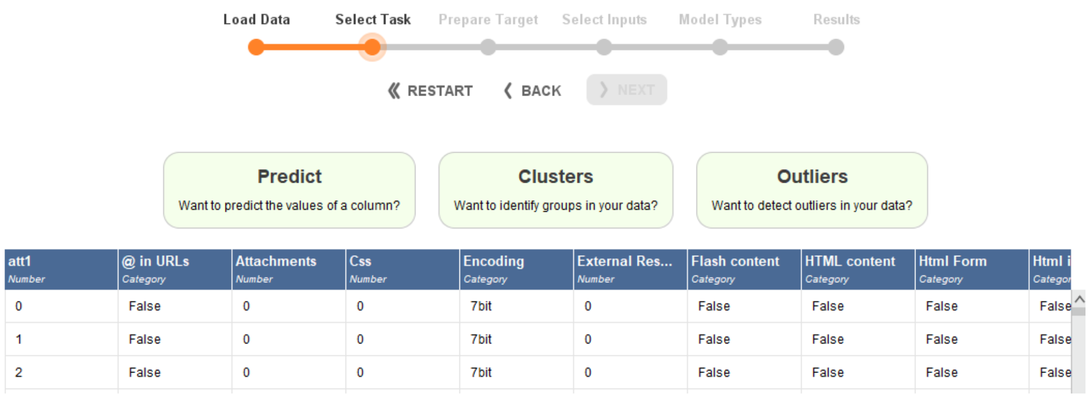
We ran this whole process for a few of Supervised Machine Learning algorithms and we have the following results.

	Accuracy	**Precision**	**Recall**

Decision Tree	96.28%+-1.04%	94.95%+-1.7%	97.78%+-1.2%
k-Nearest Neighbour	**97.07%**+-1.19%	96.27%+-1.54%	**97.96%**+-1.48%
Gradient Boosted Trees	97.01%+-0.8%	**96.4%**+-1.5%	97.7%+-1.43%

Rapid Miner Auto Model: Rapid Miner also provides with the “Auto Model” feature. This is an advanced high layer process to select the best model for the given data. The operation of this feature is very simple, where the user provides the data, selects the attributes to be trained and tested, select the Machine Learning algorithms to test and receive the performance results. Figures for each step are provided below.

1. **Load the data**
2. **Select Task**



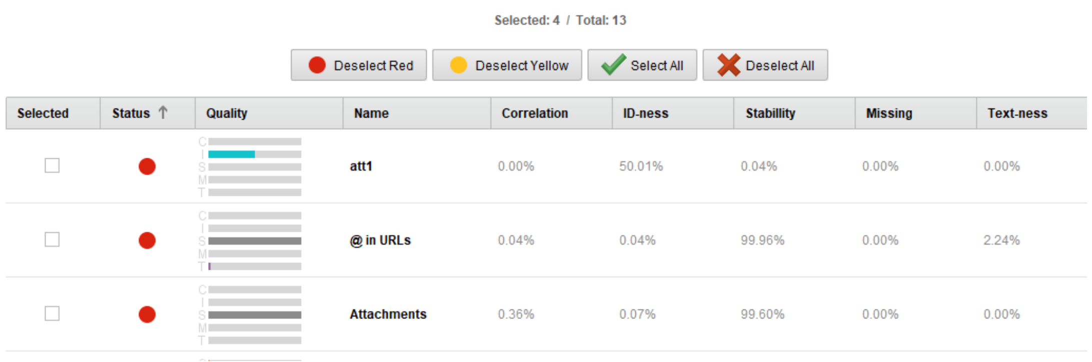
Predict Want to predict the values of a column?
Clusters Want to identify groups in your data?
Outliers Want to detect outliers in your data?

att1 *Number*	@ in URLs *Category*	Attachments *Number*	Css *Number*	Encoding *Category*	External Res... *Number*	Flash content *Category*	HTML content *Category*	Html Form *Category*	Html i *Categor*
0	False	0	0	7bit	0	False	False	False	False
1	False	0	0	7bit	0	False	False	False	False
2	False	0	0	7bit	0	False	False	False	False

Figure 53 Select Task

Here we choose what we want to do with the data. In our case we want to Predict the values of a column, so we choose that and then choose which column we want to predict, which is the “Phishy” column which specifies whether the email is phishing or not.

3. **Prepare Target**: In this step nothing was needed to be done. It is more useful for targets which have different values that are to be classified differently (for example if the column had values of -1 and 1 and it was needed to be changed to true/false).
4. **Select Inputs**:

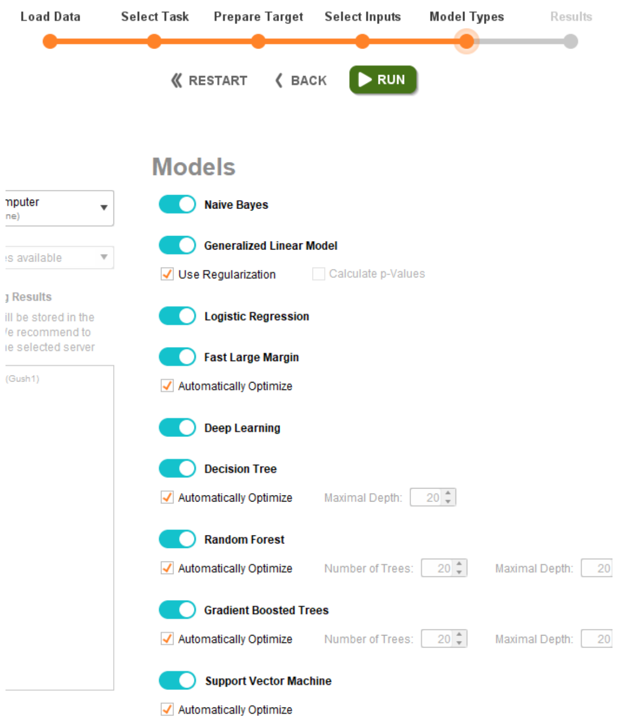

Selected: 4 / Total: 13

Selected	Status	Quality	Name	Correlation	ID-ness	Stability	Missing	Text-ness
☐	●	C I S M T	att1	0.00%	50.01%	0.04%	0.00%	0.00%
☐	●	C I S M T	@ in URLs	0.04%	0.04%	99.96%	0.00%	2.24%
☐	●	C I S M T	Attachments	0.36%	0.07%	99.60%	0.00%	0.00%

In this step the columns to be used for the training and testing are selected. There is information for every column about the correlation, ID-ness (how unique is each value), stability, if there are any missing values, text-ness (values which have free text, meaning that it is difficult to use them for training). The column "Quality" has a visual representation of each other column.

- (C) Columns that too closely mirror the target column,
- (I) Columns where nearly all values are different,
- (S) Columns where nearly all values are identical,
- (M) Columns with missing values,
- (T) Columns which look like they contain free text.

5. **Model Types:** In this step a variety of classifiers is provided as a checklist and the desired classifiers are chosen.

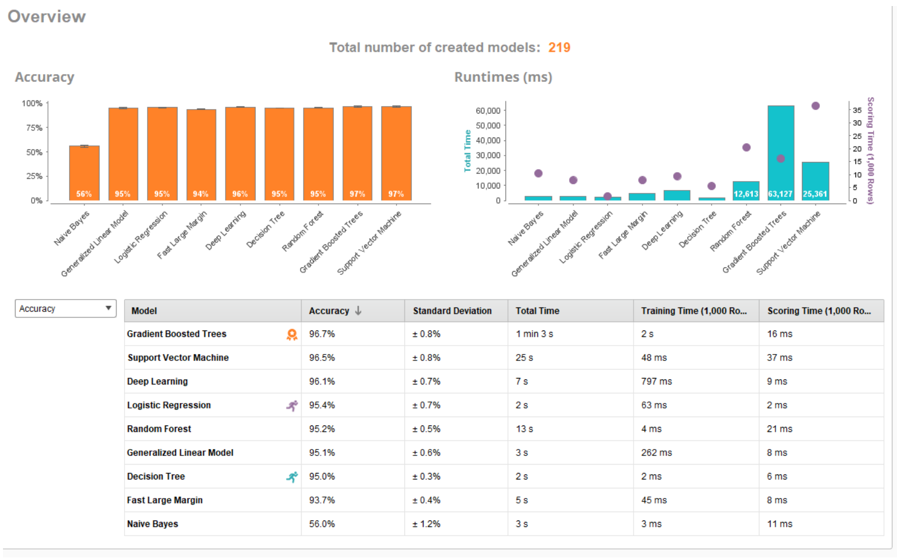


Models

- Naive Bayes
- Generalized Linear Model
  - ✓ Use Regularization
  - Calculate p-Values
- Logistic Regression
- Fast Large Margin
  - ✓ Automatically Optimize
- Deep Learning
- Decision Tree
  - ✓ Automatically Optimize — Maximal Depth: 20
- Random Forest
  - ✓ Automatically Optimize — Number of Trees: 20 — Maximal Depth: 20
- Gradient Boosted Trees
  - ✓ Automatically Optimize — Number of Trees: 20 — Maximal Depth: 20
- Support Vector Machine
  - ✓ Automatically Optimize

Figure 54 Rapid Miner Model Types

6. **Results**: After running the process the models are trained and performance results are printed for them.



Model	Accuracy ↓	Standard Deviation	Total Time	Training Time (1,000 Ro...	Scoring Time (1,000 Ro...
Gradient Boosted Trees	96.7%	± 0.8%	1 min 3 s	2 s	16 ms
Support Vector Machine	96.5%	± 0.8%	25 s	48 ms	37 ms
Deep Learning	96.1%	± 0.7%	7 s	797 ms	9 ms
Logistic Regression	95.4%	± 0.7%	2 s	63 ms	2 ms
Random Forest	95.2%	± 0.5%	13 s	4 ms	21 ms
Generalized Linear Model	95.1%	± 0.6%	3 s	262 ms	8 ms
Decision Tree	95.0%	± 0.3%	2 s	2 ms	6 ms
Fast Large Margin	93.7%	± 0.4%	5 s	45 ms	8 ms
Naive Bayes	56.0%	± 1.2%	3 s	3 ms	11 ms

Figure 55 Rapid Miner Auto Model Results

With the Auto Model tool, the optimal parameters can be found for each classifier, by testing the performance of parameter values combinations.

In conclusion, Rapid Miner is a useful tool when trying to choose a Machine Learning with the following advantages:

- Easy to build and comprehend processes with the provided graphical interface.
- Easy to test classifiers and optimize their parameters with the graphical interface and Auto Model Feature.

The disadvantages of Rapid Miner Studio are:

- The full functionality comes as a commercial licensed package.
- It does not provide functionality when it comes to retrieving features. It can only use an already built feature data set.

4.2.2 Weka

Weka [67] (Waikato Environment for Knowledge Analysis) is an open-source software of visualizing, data analysis and predictive modeling tools. It provides with graphical user interfaces for accessing these functions. It is written in Java and provides APIs for code writing for Java as well as for Python with a wrapper [68], R with the RWeka package [69] and for Spark with the distributed Weka for Spark package [70].

Weka has a few user interfaces, with the primary one being the Explorer. There is also the “Knowledge Flow”, a command line interface and the “Experimenter”, which enables an automated comparison of the performance of different machine learning algorithms on the provided dataset.

The “Explorer” interface has the following different sub-interfaces as panels:

- **Preprocess** provides functionality for importing data from a database, a csv file etc. and for preprocessing the data by filtering it, filling missing values, transforming values to a desired type etc.
- **Classify** is for applying classification algorithms to the preprocessed now data set and produce predictions and performance evaluation.
- **Associate** provides association rule learners for finding any possible relationships/(in)dependencies between features of the dataset.
- **Cluster** enables clustering techniques (k-means clustering).
- **Select attributes** includes algorithms for feature selection/extraction.
- **Visualize** show a scatter plot matrix of predictions and other relevant information.

The Weka’s advantages include:

- Open source – Free license (GNU General Public License)
- Easy testing of machine learning algorithms and comprehension of the process, with the graphical user interface.

However, Weka’s disadvantages are:

- No functionality when it comes to feature retrieving for a raw data source (e.g. an email data set). It requires as input the data set of feature values for the examples of the raw data source (e.g. the features retrieved from the emails and transformed to a csv file).
- Although it provides APIs for programming languages, there are already libraries, packages and modules for machine learning languages which will naturally be more efficient when it comes to processing time because there are less commands executed.

4.2.3 TensorFlow

TensorFlow [71] is an open-source platform for machine learning. It provides tools and libraries for building, testing and deploying machine learning powered applications. It was developed by the Google Brain team, a deep learning artificial intelligence research team at Google.

There are lower level APIs which can be used to build models by defining a series of mathematical operations, and there are high level APIs like the tf.estimator API which provides with already built machine learning models which can be used to train and make predictions.

TensorFlow has the following advantages:

- Can develop the whole machine learning process through it, because it is part of code so at first the preprocessing can happen with the result being the retrieved from the emails features, and then apache spark can be used for the classification.
- It is open source software.

Disadvantages of TensorFlow include:

- Since it is practically a software library, it can seem daunting at start.

4.2.4 Apache Spark

Apache Spark [72] is an open source general-purpose cluster computing system. It provides APIs in Java, Scala, Python and R. It supports high-level tools including Spark SQL for SQL data processing, MLlib for machine learning, GraphX for graph processing and Spark Streaming.

Spark offers the ability to combine discrete tasks on data such as selecting data, transforming the data in different ways and analyzing the resulting data with several methods. Spark jobs perform multiple operations consecutively in the memory of the computer. The way that Spark provides easy implementation methods of multiple operations on data is with the Data Pipelines. A data pipeline can include many different processes and, as a whole, be a complex transformer, combining large number of inputs and processing steps and produce the desired results.

Spark runs multi-threaded light tasks inside Java Virtual Machine (JVM) processes, enabling easy parallelization of them in a multi-core CPU environment. It caches the data in the memory of the computer, thus improving the processing times of parallelized processes.

Apache Spark has the following advantages:

- Fast processing time
- Rich variety of available processes and tools that facilitate the development and implementation of projects
- Open source
- Can develop the whole machine learning process through it, because it is part of code so at first the preprocessing can happen with the result being the retrieved from the emails features, and then apache spark can be used for the classification.
- It has a very active online community, from which solutions to any kind of problem can be found.

The Spark's disadvantages are:

- Could be difficult to comprehend at start, both as a cluster computing system (its architecture) and as a software library (the relevant pieces of code to use and run it)

4.2.5 Criteria for choosing the most suitable tool for the purposes of this work

In order to select the most appropriate tool for developing the phishing email classifier, the following criteria have to be considered:

- **User comprehensibility**: A basic criterion for development is how easily and how well a new developer and a generic user can understand the tool. This includes understanding the underlying architecture of the tool's processes and procedures, understanding the functions on a higher-level as well as the relevant code/building blocks when developing a program with the tool.
- **Distribution Method**: An important factor to determine the right tool is the way that a user can have access to the tool and use it. More specifically, an open-source licensed product is preferred as there is permission to use the source code, the design documents and the content of the product. This enables the free development, implementation and deployment of the tool with practically no restriction.
- **Open ended-ness/Restrictions**: For the proper development of a process like the machine learning classification that is the purpose of this thesis, there needs to be as few restrictions as possible when it comes to how the tool is used. This includes the ways that the tool's parts are allowed and meant to be used. A tool that might seem to be useful at start might end up not providing enough freedom to develop the intended project.
- **Basic functionalities**: This criterion is particularly important in the case of machine learning because there is a series of steps which need to be completed in order to build a model. This includes functionality for handling the raw data source, processing it and retrieving the features, as well as building, training and testing a machine learning model with the resulting features. Having these functionalities can be achieved by using different programs, tools and programming languages, but it is more efficient to be able to complete these steps in one seamless stream of procedures.
- **Online Community**: Last, but not least, it is important to have activity in the internet when it comes to the usage of the tool, because this means that there will be a higher chance that

any question that may come when using the tool is already answered in a relevant forum and that if there is a question without an answer it can be asked in the forums and have an answer relatively shortly.

Taking into account these criteria and applying them to the aforementioned tools that were found to be used in relevant work and literature resulted in the following table. A simple rating of 1-3 was assigned to each criterion for every tool.

Table 7 Machine Learning Tool Criteria

Tool	**Rapid Miner**	**Weka**	**TensorFlow**	**Apache Spark**
User Comprehensibility	3	3	2	2
Distribution Method	1	3	3	3
Open ended-ness	2	2	3	3
Functionalities	1	1	3	3
Online Community	1	1	2	3
Total	**8**	**10**	**13**	**14**

Following the table and the fact that Apache Spark is a competitive tool when it comes to processing speed (because of the parallelization techniques that it employs) the tool that is used for the implementation of this thesis is the **Apache Spark**. In the following chapter more details about this tool will be provided, including its underlying architecture and the libraries that are included. In the implementation chapter there will be more specific information at the parts of the code where it was determined necessary.

4.3 Detailed Presentation of Spark and its Features

Spark can run as a standalone program or in cluster mode. Cluster mode is where Spark application can run on multiple processing units, either on the same system or on different ones that are connected to a network. The architecture of spark running in cluster mode is shown in the following figure [73].

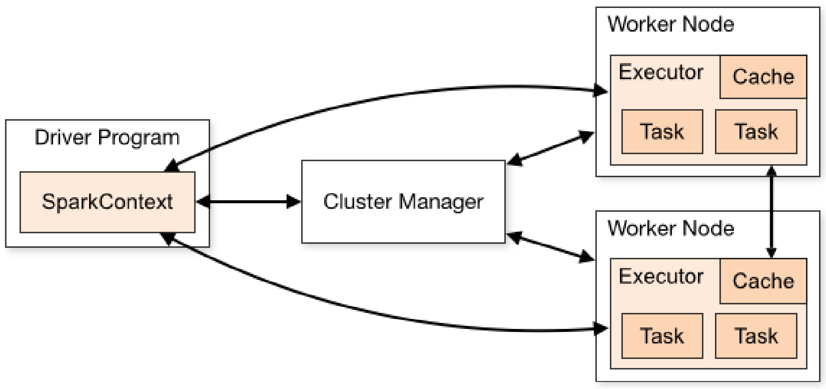


Figure 56 Apache Spark Cluster Mode

Spark applications run as sets of processes on a cluster. These processes are coordinated by the SparkContext object in the main program, which is the Driver Program in the figure. There needs to be Cluster Managers which will handle which process will run on which node of the network. Cluster Managers which are supported by spark are the following:

- **Standalone**: this is a simple cluster manager which is included in spark for easier and simpler implementations.
- **Apache Mesos** [74]
- **Hadoop YARN** [75]: the resource manager in Hadoop [76]
- **Kubernetes** [77]: an open source system for automating deployment and management of containerized applications

The process continues as follows:

1. Spark acquires Executors from the network, which are processes which will execute what SparkContext assigns them to and also store data in the memory.
2. Spark sends the application code to the executors.
3. SparkContext sends tasks to the executors to run.

The main abstraction that spark provides is the resilient distributed dataset (RDD). It is a collection of elements which are spread across the nodes of the cluster and they can be accessed in parallel from one another. RDDs can be created either with a file in the Hadoop file system, or an existing Scala collection in the driver program and transforming it to an RDD.

Each application gets its own executor processes for the duration of the whole application. That way each application is isolated from another. The isolation has the drawback that applications cannot directly share data with each other.

Spark does not take into account the type of cluster manager that is used. For spark what matters is that there is a cluster manager which will properly provide with executors that spark needs for its applications.

Because the cluster architecture requires a network for it to operate and exchanges to happen between driver and executors, the driver needs to have a known address/port for the executors to be able to directly send to it data, requests and results.

For efficiency reasons, the driver program should be in a local network with the executors, for the communications to be as short as possible.

As mentioned before, spark uses a stack of libraries including SQL and DataFrames, MLlib, GraphX and Spark Streaming.

- **Spark SQL and DataFrames**: Spark SQL is spark's module for working with structured data. This module allows querying structured data inside spark programs using SQL queries or a familiar DataFrame API. As a clarification:
    - A Dataset is a distributed collection of data.
    - A DataFrame is a Dataset organized into named columns. DataFrames can be created from a variety of sources, including structured data files (csv, json etc.), tablies in Hive, external databases and existing RDDs. The DataFrame API is available in all the languages of spark (Scala, Java, Python and R). The features array for this work's purposes will be stored as a DataFrame.
- **Spark Streaming**'s purpose is to build scalable steaming applications. It brings a language-integrated API for stream processing in Java, Scala and Python. It can read data from various sources including Kafka, Twitter and HDFS, as well as custom created data sources.
- **GraphX** is a Spark API for graphs and graph-parallel computation. In a single system one can use iterative graph computations, exploratory analysis and ETL (Extract, Transform, Load). It provides capabilities to displaying data as graphs and arrays, transform and join the graphs using RDDs and write iterative graph algorithms.
- **MLlib** is Spark's scalable machine learning library. Using Spark's iterative computation capabilities help MLlib train classifiers efficiently. More information about MLlib wil be presented in the following chapter.

4.3.1 Spark's MLlib library for Machine Learning Applications

MLlib is a library of Apache Spark for machine learning purposes. It provides the following functionality:

- Machine Learning algorithms, including classification, regressing, clustering, collaborative filter and deep learning with neural networks.
- Featurization, including feature extraction and selection, transformation of features and dimensionality reduction
- Pipelines are tools for merging series of processes for machine learning purposes in one seamless process.
- Persistence: can save and load algorithms, models and Pipelines.
- Utilities, including statistics, data handling and others.

4.3.1.1 Basic Statistics

In the following the functions of MLlib will be presented.

Correlation: It is useful to calculate the correlation between a feature and the label we want to predict, because if the correlation is close to 1 this means that the label can be found with almost 100% certainty just from this 1 feature. This means that there is nothing to be taught to the machine learning model which is supposed to be trained using all the features. We can just use a function which will take as input the feature and produce the label output. If, on the other hand, the correlation is close to 0, then there is no information provided by this feature, so it is pointless to include it in the training. The function *Correlation* of MLlib computes the correlation matrix for the input Dataset of vectors using a specified method (the currently supported methods are Pearson's and Spearman's correlations).

Hypothesis testing is a tool to determine if a result is statistically significant. Spark's *ChiSquareTest* executes Pearson's independence test for every feature against the label and a contingency matrix is made for each statistic.

Summarizer vector column summary statistics. The metrics that it provides are column-wise maximum, minimum, mean, variance, number of non-zeros and total number of examples.

4.3.1.2 Pipelines

ML Pipelines are a fundamental part of the Spark MLlib library and provide functionality on the DataFrames for building and tuning machine learning processes. The use of Pipelines helps with combining multiple transforming and machine learning algorithms into one single pipeline or workflow. The components that are used in building and operating Pipelines are the following.

- **DataFrame**: As mentioned before, a DataSet is a distributed collection of data. A DataFrame is a DataSet organized into named columns. This ML API uses the DataFrame from Spark SQL as a dataset for machine learning purposes. Such DataFrame can include any type of data in its columns including text, feature vectors, labels and predictions. The DataFrame is used as an input for the machine learning algorithms, which use practically any type of data and transform them into useful and informative (for the classifier) features.
- **Transformer**: A Transformer is the part of the Pipeline which will transform a DataFrame to another. The most common case would be to transform a DataFrame of features to one of predictions. Technically, a Transformer is a method called *transform* which will execute the transformation. For example, a transformer which builds features for a classifier can take a column of text, transform it to a numerical representation. Afterwards, it will take this column and any other feature column of numerical and boolean type values and assemble them all into one feature vector column to be used as input for the machine learning classifier. A second, follow-up of the first, transformer would be a classifier. One that takes as input these features and makes a prediction/classification, by adding to the DataFrame a column "*predictions*".
- **Estimator**: The Estimator "fits" in a DataFrame to produce a Transformer. Using the example of the Transformer, an Estimator could be used in the Pipeline prior to making the predictions with the classifier. The Pipeline would have the untrained classifier as its last stage. The DataFrame would be split into a training and a test set. The Pipeline would be

used first on the training set to fit on the DataFrame. In other words, the classifier would be trained using the training set and the label column which should be in the DataFrame. Then a transformer, or model/classifier, is produced, which can then be used to transform the test set and make predictions, with the purpose of testing the model's performance. When it comes to coding, the Estimator is a Pipeline method called *fit*, which takes a DataFrame as input and outputs a Model/Transformer.

- **Pipeline**: A Pipeline appends a series of Transformers and Estimators to build a series of processes which are to be executed on a DataFrame. When it comes to coding, a Pipeline consists of a series of stages. The stages are run in order on the input DataFrame and it is transformed at each stage. The Pipeline is an Estimator, so at first it is called with the *fit* method and a Transformer will be produced which is called Pipeline Model. The Pipeline Model will be called on the DataFrame with the *transform* method, and the DataFrame will be transformed each stage at a time.
- **Parameter**: Estimators and Transformers use the same API for setting parameters. The parameters can be set at the initialization phase of an Estimator or Transformer instance, by using the API to set parameters on already initialized ones or by passing pairs of parameters and values on *fit* and *transform* methods.

4.3.1.3 Feature retrieval, transformation and selection/extraction

A fundamental part of machine learning is to create an efficient and purposeful algorithm for transforming the raw input data which is to be classified into representative features. Spark's MLlib provides a wide plethora of feature related algorithms from the following categories:

- **Retrieval**: Retrieving feature from the raw input data.
- **Transformation**: These including feature modification algorithms including scaling and normalizing.
- **Selection/Extraction**: With feature selection a subset of the features is selected to be used as input for the classifier. With feature extraction the features which can be relatively dependent on each other are transformed into a set of independent features that will be used for the classification process.

The following algorithms are not every algorithm from this section of MLlib but rather the ones that were used in this work for classifying phishing emails:

Feature Retrievers

- **TF-IDF**: Term frequency-inverse document frequency is a feature retriever method for text classification. With it the resulting features represent words and words that appear frequently in one example but not frequently in the whole document of examples will have a higher weight. The methods in MLlib are *HashingTF* and *IDF*. HashingTF is a Transformer which takes sets of terms (words) and transforms them to feature vectors of fixed length. IDF, on the other hand, is an Estimator which is fit on the HashingTF's output and produces an IDFModel. This Model takes feature vectors and scales each feature. As the name "inverse document frequency" suggests, this Model will reduce the weight of features that appear frequently in the examples of the document.

- **Word2Vec**: This is an Estimator which implements the Word Embedding. Word Embedding is a text feature retrieving method with which every word is mapped to a vector. This is a more sophisticated text feature method in the sense that, in addition to providing frequency-based information (because if a word exists then the corresponding vector will exist, too), words with similar meanings will have neighboring vectors. Word2vec is a method created and published in 2013 by researchers at Google. In Spark, Word2Vec is an Estimator which takes sequences of words and trains a Word2VecModel. This Model will map each word to its corresponding vector and transforms each text examples into a vector using the average of all words in the document.
- Other types of features for the case of emails which are based on the properties of the emails, such as number of URLs in the email or number of attachments of the email, are specific to emails, so specific methods were created for these purposes. These methods are described in the Implementation chapter.

Feature Transformers

- **Tokenizer**: A Tokenizer takes a text as input and breaks it into individual terms, creating an array. There is a delimiter parameter which states the character which is the delimiter between the terms. A **RegexTokenizer** executes the same operation, with the difference that the delimiter is now a regular expression. There is a parameter which can be set to true and then the RegexTokenizer will instead use the regular expression to find the terms themselves instead of as a delimiter.
- **StopWordsRemover**: Stop words are words which do not provide useful information when it comes to the context of a text document, such as "I, you, the, in, as, on, a" etc. The method StopWordsRemover takes as input a vector of words, such as the output of the Tokenizer method, and removes the Stop Words. There are a few available options when it comes to language: "danish", "dutch", "english", "finnish", "french", "german", "hungarian", "italian", "norwegian", "portuguese", "russian", "spanish", "swedish" and "turkish".
- **StringIndexer**: This method takes as input a column of strings or numbers and transforms them to indices. The indices are number from 0 to the number of different values in the column. This method is particularly useful if a feature is of string type and has a limited number of different string values. Then it can be transformed to a numerical feature using the StringIndexer, and the numerical values could then be used for training the different machine learning algorithms.
- **OneHotEncoderEstimator**: One-Hot encoding maps a categorical feature (of string type value) which is represented as an index (typically using the output of StringIndexer) to a binary vector which has zeroes at every index except one which has one. This index represents the specific value of the index which was the input of the encoder.
- **Normalizer**: This is a Transformer which transforms a dataset of Vector rows so that every vector has a unit norm.
- **StandardScaler**: This method transforms a dataset of vector rows, typically features, so that each feature has unit standard deviation and/or zero mean.
- **VectorAssembler**: This transformer takes as input a list of columns of the input DataFrame and converts them into a single vector column. It accepts columns with types of numeric, boolean and vector type. This is a crucial step towards machine learning classification,

because the machine learning algorithms in Spark are designed to take as input one feature vector column and use it to either train or classify.

Feature Selectors/Extractors

- **PCA**: Although this method is categorized as a Feature Transformer in the spark documentation page, PCA is considered as feature extractor by relevant research work [42]. Principal Component Analysis is a feature extraction algorithm which takes a set of inputs and produces a new set based on the input, but the output is a set of linearly independent values. The PCA method is an Estimator which creates a Model that projects vectors to a low-dimensional space using the PCA algorithm. The number of dimensions can be set as a parameter.
- **ChiSqSelector**: This Estimator creates a Transformer which uses the Chi-Squared test of independence to select the most relevant features. The Chi-Square test of independence is a procedure to test if two features are related. The relation is found by comparing the frequency of each value of the first feature with all the values of the second feature. The number of the resulting features can be either set as a fixed number or as a percentage of the number of total input features.

4.3.1.4 Classification and Regression

This chapter includes the classification and regression algorithms which are included in the Spark's MLlib module.

Classification

- **Logistic regression**: This classifier has been mentioned in the Theoretical Background chapter. It supports both binomial and multinomial (multi-class) classification, with the same classifier class. The class definition, which showcases the parameters of this class, is the following:

  ```
  class pyspark.ml.classification.LogisticRegression(featuresCol='features', labelCol='label',
  predictionCol='prediction', maxIter=100, regParam=0.0, elasticNetParam=0.0, tol=1e-
  06, fitIntercept=True, threshold=0.5, thresholds=None, probabilityCol='probability', rawPre
  dictionCol='rawPrediction', standardization=True, weightCol=None, aggregationDepth=2, f
  amily='auto', lowerBoundsOnCoefficients=None, upperBoundsOnCoefficients=None, lower
  BoundsOnIntercepts=None, upperBoundsOnIntercepts=None)
  ```

  The parameters *featuresCol, labelCol* and *predictionCol* have the names of the corresponding columns of features and label which are taken as input from the DataFrame and the predictions column which is to be created after Transforming the DataFrame.
    - Binomial logistic regression
    - Multinomial logistic regression
- **Decision tree classifier**: The class definition is the following:

```
class pyspark.ml.classification.DecisionTreeClassifier(featuresCol='features', labelCol='label', predictionCol='prediction', probabilityCol='probability', rawPredictionCol='rawPrediction', maxDepth=5, maxBins=32, minInstancesPerNode=1, minInfoGain=0.0, maxMemoryInMB=256, cacheNodeIds=False, checkpointInterval=10, impurity='gini', seed=None)
```

- **Random forest classifier**

```
class pyspark.ml.classification.RandomForestClassifier(featuresCol='features', labelCol='label', predictionCol='prediction', probabilityCol='probability', rawPredictionCol='rawPrediction', maxDepth=5, maxBins=32, minInstancesPerNode=1, minInfoGain=0.0, maxMemoryInMB=256, cacheNodeIds=False, checkpointInterval=10, impurity='gini', numTrees=20, featureSubsetStrategy='auto', seed=None, subsamplingRate=1.0)
```

- **Gradient-boosted tree classifier**

```
class pyspark.ml.classification.GBTClassifier(featuresCol='features', labelCol='label', predictionCol='prediction', maxDepth=5, maxBins=32, minInstancesPerNode=1, minInfoGain=0.0, maxMemoryInMB=256, cacheNodeIds=False, checkpointInterval=10, lossType='logistic', maxIter=20, stepSize=0.1, seed=None, subsamplingRate=1.0, featureSubsetStrategy='all')
```

- **Multilayer perceptron classifier**: The MLPC is a classifier based on the feedforward artificial neural network. It is an ANN with multiple layers of nodes. Each layer is fully connected to the next layer. As is the case with other ANNs in the case of classification, the first layer is the input layer where the features are the inputs and the last layer is the output layer where the classification happens.

```
class pyspark.ml.classification.MultilayerPerceptronClassifier(featuresCol='features', labelCol='label', predictionCol='prediction', maxIter=100, tol=1e-06, seed=None, layers=None, blockSize=128, stepSize=0.03, solver='l-bfgs', initialWeights=None, probabilityCol='probability', rawPredictionCol='rawPrediction')
```

- **Linear Support Vector Machine**: As it has been already explained in the Theoretical Background chapter, the linear is the simplest version of the Support vector machines.

```
class pyspark.ml.classification.LinearSVC(featuresCol='features', labelCol='label', predictionCol='prediction', maxIter=100, regParam=0.0, tol=1e-06, rawPredictionCol='rawPrediction', fitIntercept=True, standardization=True, threshold=0.0, weightCol=None, aggregationDepth=2)
```

- **One-vs-Rest classifier (a.k.a. One-vs-All)**: This classifier performs multiclass classification given a base classifier that can perform binary classification efficiently. It takes the binary classifier and creates a binary classification problem for each of the k classes. The classifier for every class is trained to predict where the label is *i* or not. The corresponding predictions are done by evaluating each binary classifier and the classifier with the highest confidence in its prediction will be the final output classification.

```
class pyspark.ml.classification.OneVsRest(featuresCol='features', labelCol='label', predictionCol='prediction', classifier=None, weightCol=None, parallelism=1)
```

- **Naive Bayes**

```
class pyspark.ml.classification.NaiveBayes(featuresCol='features', labelCol='label', predictionCol='prediction', probabilityCol='probability', rawPredictionCol='rawPrediction', smoothing=1.0, modelType='multinomial', thresholds=None, weightCol=None)
```

Regression

- **Linear regression**

  class pyspark.ml.regression.**LinearRegression**(*featuresCol='features'*, *labelCol='label'*, *predictionCol='prediction'*, *maxIter=100*, *regParam=0.0*, *elasticNetParam=0.0*, *tol=1e-06*, *fitIntercept=True*, *standardization=True*, *solver='auto'*, *weightCol=None*, *aggregationDepth=2*, *loss='squaredError'*, *epsilon=1.35*)

- **Generalized linear regression**: The simple linear regression assumes that the output follows a Gaussian distribution. With the GLM, it is assumed that the output follows a specified distribution of the exponential family. The available families are: Gaussian, Binomial, Poisson, Gamma and Tweedie.

  class pyspark.ml.regression.**GeneralizedLinearRegression**(*labelCol='label'*, *featuresCol='features'*, *predictionCol='prediction'*, *family='gaussian'*, *link=None*, *fitIntercept=True*, *maxIter=25*, *tol=1e-06*, *regParam=0.0*, *weightCol=None*, *solver='irls'*, *linkPredictionCol=None*, *variancePower=0.0*, *linkPower=None*, *offsetCol=None*)

- **Decision tree regression**

  class pyspark.ml.regression.**DecisionTreeRegressor**(*featuresCol='features'*, *labelCol='label'*, *predictionCol='prediction'*, *maxDepth=5*, *maxBins=32*, *minInstancesPerNode=1*, *minInfoGain=0.0*, *maxMemoryInMB=256*, *cacheNodeIds=False*, *checkpointInterval=10*, *impurity='variance'*, *seed=None*, *varianceCol=None*)

- **Random forest regression**

  class pyspark.ml.regression.**RandomForestRegressor**(*featuresCol='features'*, *labelCol='label'*, *predictionCol='prediction'*, *maxDepth=5*, *maxBins=32*, *minInstancesPerNode=1*, *minInfoGain=0.0*, *maxMemoryInMB=256*, *cacheNodeIds=False*, *checkpointInterval=10*, *impurity='variance'*, *subsamplingRate=1.0*, *seed=None*, *numTrees=20*, *featureSubsetStrategy='auto'*)

- **Gradient-boosted tree regression**

  class pyspark.ml.regression.**GBTRegressor**(*featuresCol='features'*, *labelCol='label'*, *predictionCol='prediction'*, *maxDepth=5*, *maxBins=32*, *minInstancesPerNode=1*, *minInfoGain=0.0*, *maxMemoryInMB=256*, *cacheNodeIds=False*, *subsamplingRate=1.0*, *checkpointInterval=10*, *lossType='squared'*, *maxIter=20*, *stepSize=0.1*, *seed=None*, *impurity='variance'*, *featureSubsetStrategy='all'*)

- **Survival regression**: The implementation is the Accelerated failure time (AFT) model which is a parametric survival regression model for censored data. It is a model for the log of survival time. The AFT model is easier to parallelize because each instance for the objective function is calculated independently.

  class pyspark.ml.regression.**AFTSurvivalRegression**(*featuresCol='features', labelCol='label', predictionCol='prediction', fitIntercept=True, maxIter=100, tol=1e-06, censorCol='censor', quantileProbabilities=[0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95, 0.99], quantilesCol=None, aggregationDepth=2*)

- **Isotonic regression**: The Isotonic Regression is a problem where if *y_1, y_2, …, y_n* are observed responses and *x_1, x_2, …, x_n* are the unknown response values to be fitted, then the solution is finding the function that minimized the following (w_i are corresponding weights):

$$f(x) = \sum_{i=1}^{n} w_i (y_i - x_i)^2$$

```
class pyspark.ml.regression.IsotonicRegression(featuresCol='features', labelCol='label', pr
edictionCol='prediction', weightCol=None, isotonic=True, featureIndex=0)
```

Linear methods

Linear methods include logistic regression and linear least squares, with L_1 or L_2 regularization. L_1 and L_2 regularization penalize complex models. L_2 tries to make the weights of features have small value, while L_1 tries to bring the weights' values to zero. L_1, in a way, is more determinant. The implementation of these regularization in spark is with the Elastic net which is a hybrid of both regularizations. Given the weights $\mathbf{w_i}$ of the linear method, $\boldsymbol{\lambda}$ the learning rate and $\boldsymbol{\alpha}$ the elastic net parameter, the regularization function (which is to be minimized) is the following:

$$\alpha(\lambda(|w_1| + |w_2| + \cdots + |w_n|)) + (1\text{-}\alpha)\left(\frac{\lambda}{2}(w_1^2 + w_2^2 + \cdots + w_n^2)\right), \alpha \in [0,1], \lambda \geq 0$$

In the case of the linear methods in spark, there is a parameter *elasticNetParam* which represents the elastic net parameter $\boldsymbol{\alpha}$.

4.3.1.5 Clustering

Clustering is an unsupervised classification method, meaning that there is no specific class to predict (e.g. classify an email as phishing or not). However, the algorithm attempts to group the examples into neighboring groups, with each group's examples having similar features values.

K-means

This algorithm clusters the data points/examples into a defined number of groups/clusters.

```
class pyspark.ml.clustering.KMeans(featuresCol='features', predictionCol='prediction', k=2, initMo
de='k-means||', initSteps=2, tol=0.0001, maxIter=20, seed=None, distanceMeasure='euclidean')
```

Latent Dirichlet allocation (LDA)

LDA is a type of topic modeling, which is another method of unsupervised classification. The idea is that each example is made up of words/terms, and there are topics which have some of the terms in them. The purpose is to find in which topic the example belongs to.

class pyspark.ml.clustering.**LDA**(*featuresCol='features', maxIter=20, seed=None, checkpointInterval=10, k=10, optimizer='online', learningOffset=1024.0, learningDecay=0.51, subsamplingRate=0.05, optimizeDocConcentration=True, docConcentration=None, topicConcentration=None, topicDistributionCol='topicDistribution', keepLastCheckpoint=True*)

Bisecting k-means

This is a hierarchical clustering algorithm which uses a top-down approach. All the examples start in one cluster and they are split recursively while moving down in the hierarchy.

class pyspark.ml.clustering.**BisectingKMeans**(*featuresCol='features', predictionCol='prediction', maxIter=20, seed=None, k=4, minDivisibleClusterSize=1.0, distanceMeasure='euclidean'*)

Gaussian Mixture Model (GMM)

This is a composite distribution where points are drawn from one of k Gaussian sub-distributions, each with its own probability.

class pyspark.ml.clustering.**GaussianMixture**(*featuresCol='features', predictionCol='prediction', k=2, probabilityCol='probability', tol=0.01, maxIter=100, seed=None*)

4.3.1.6 Collaborative filtering

This technique is used for recommender systems. Spark implements this with the Alternating Least Squares (ALS) to fill missing entries in a user-item association matrix.

4.3.1.7 Model Selection and Tuning

When building a machine learning classifier, there are several choices to be made when it comes to model selection and what parameters to use for the model. Spark's MLlib has the CrossValidator and the TrainValidationSplit methods for this hyperparameter tuning process. To execute this process, inputs must be provided for an Estimator to tune, a set of parameter maps for the parameter combinations to test the performance with and an Evaluator algorithm which will yield a metric to compare and find the best performing parameter combination of the Estimator Model.

The process is as follows:

- The dataset is split into training and test datasets.
- For each parameter combination from the parameter map the Estimator is fitted on the training dataset and then tested on the test dataset. The performance evaluation metric for each parameter combination is calculated and kept.
- After every model for every parameter combination has been evaluated, the metrics are compared and the model with the highest metric is the chosen one.

For the parameter map, the spark method ParamGridBuilder is used, which takes as input vectors of parameter values and creates an iterator which yields every parameter combination.

Cross-Validation works by splitting the dataset into k equal subsets. The training-testing process is completed k times, with the test set being one of the subsets each time and the training set being all the other subsets. With the cross-validation any possible bias that could have existed if there was just one training-test set split is minimized and the performance results are average values of each training-test process.

The TrainValidationSplit method is the simple process of splitting the dataset into a training and a test set and completing the evaluation of each parameter combination on these fixed datasets.

4.4 Machine Learning Classification Architecture

In the case of classifying emails as legitimate or phishing, there are the following characteristics when it comes to the process:

The classification is **supervised**, because the classification classes are predetermined: The email will be classified either as phishing or as not phishing (legitimate. The dataset with the emails will have a column with the label, the value which describes what class the email belongs to, so that the machine learning algorithm can adapt its weights so that it classifies these emails (and new ones) with the minimum errors.

An email is practically a text file. As mentioned before there may be multiple parts in an email, which can all be represented as text. However, the only text that the receiver reads is the text body of the email. In total, there are two basic characteristics in an email:

- The email text body
- Other properties of the email (number of URLs, whether it uses HTML/JavaScript, number of attachments, other header field values etc.)

These two categories will be two different sources of features for the machine learning classifiers. As it will be shown in the implementation chapter, they will be used either separately or in different combinations. The retrieved features from the two sources will be named:

- **Text-based features**: For these features Spark's MLlib module already provides well designed methods, including the TF-IDF algorithm and the Word Embedding.
- **Properties-based features**: For these features there are no already-built methods and tools for their retrieval, because they are too specific in each classification process and in the case of emails, too. For this purpose, simple methods are built to retrieve each useful feature, which will be presented in the implementation chapter.

The data sets from which the emails are retrieved in a previous chapter. They include just emails in one continuous text document. They will be processed one at a time and a Spark DataFrame will be created which will be used by the machine learning algorithms. The steps required to build and train a machine learning model are the following.

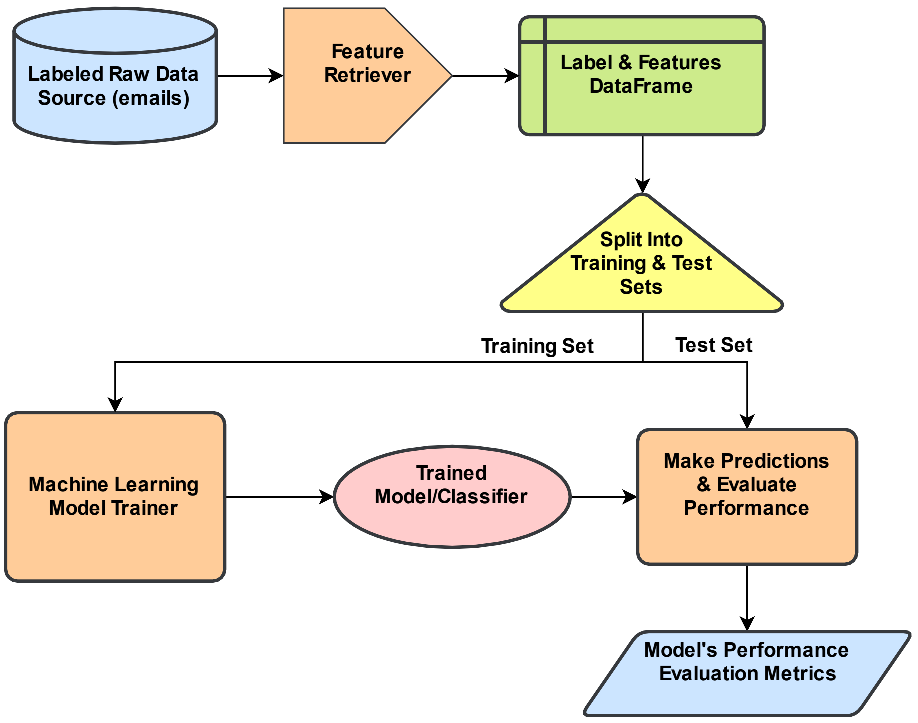

Labeled Raw Data Source (emails)
Feature Retriever
Label & Features DataFrame
Split Into Training & Test Sets
Training Set
Test Set
Machine Learning Model Trainer
Trained Model/Classifier
Make Predictions & Evaluate Performance
Model's Performance Evaluation Metrics

Figure 57 Machine Learning Training and Evaluation Process

The raw data which are labeled (the emails are already classified as phishing or legitimate) are used as input for the Feature Retriever. The Feature Retriever is a function which takes specified inputs (emails in our case) and outputs the corresponding features. The result is a DataFrame with its columns being the retrieved features and one column being the label (the class – zero if the email of legitimate and one if the email is phishing) and the rows are different examples – different emails from the data set. The DataFrame is then split into two subsets: a training set and a test set. The training set is used as input for the Machine Learning Model Trainer. The trainer takes as input a DataFrame which has features and a label column and uses it to train a specified machine learning classifier with specified hyperparameters. The output is a trained Model/Classifier. The Trained Model is then used with the test set as input, predictions for the test set are made and the performance of the classifier is calculated by comparing the predictions with the actual label values of the examples. The last output is the model's performance evaluation metrics, such as accuracy, precision, recall etc.

In the next chapter there will be a thorough presentation of the implementation of the proposed architecture, with flow charts for every step and explanation of every function that was created for that purpose.

5 Implementation

In this chapter the implementation of the proposed architecture will be presented. The analysis of the implementation will be based on the flow chart shown below. The relevant code for every part will be presented and explained. More specifically, there will be a description of the implemented code's parts, including functions, classes and the way it operates. Any information about included libraries and modules will be provided at the parts of code in which they were used.

The code of the implementation as well as the data sets used to train the classifiers can be found here [78].

5.1 Tools used for the implementation

Before going into the details of the implementation, the tools used for it will be presented.

First and foremost, the programming language that was used for the implementation is the Python programming language. More specifically the version of Python that was used is the 3.7.3 version, which is not the most recent version (3.8.1 version, released on 18 December 2019). This version was used at the start of the development of the program and was kept for consistency reasons. There were some concerns about version 2, with the most recent version being 2.7.17, but it has been announced that Python version 2 will become discontinued on January 1, 2020. This means that there will be no more improvements after that day, even if there are important security problems that might appear. That is the reason that Python 2 was not used and should no longer be used in any development.

For the development of the program, the Anaconda Python Distribution was used [79]. It is an open-source distribution of Python and the R programming languages for data science, machine learning, large-scale data processing, predictive analytics and other relevant purposes. It provides with more than 1,500 packages for these purposes already installed and simplifies the package management with the package management system called conda. It can be used on Windows, Linux and MacOS.

Conda helps as a package manager because when it installs a package, instead of just installing any dependent Python package that is required without checking the already installed packages' dependencies, it checks every currently installed package and their version dependencies and then it works out how to install a compatible set of dependencies.

Anaconda provides the Anaconda Navigator, a desktop graphical user interface that allows the launch of applications and management of packages etc. without using a command line interface. The following applications are available in the Anaconda Navigator:

- JupyterLab
- Jupyter Notebook
- QtConsole
- Spyder
- Glueviz
- Orange

- RStudio
- Visual Studio Code

For the development of the thesis implementation, at first the Jupyter Notebook was used. That is because there are extensions using the Jupyter Notebook, Hadoop and the ELK stack that might have helped with the handling of emails, like in the following figure.

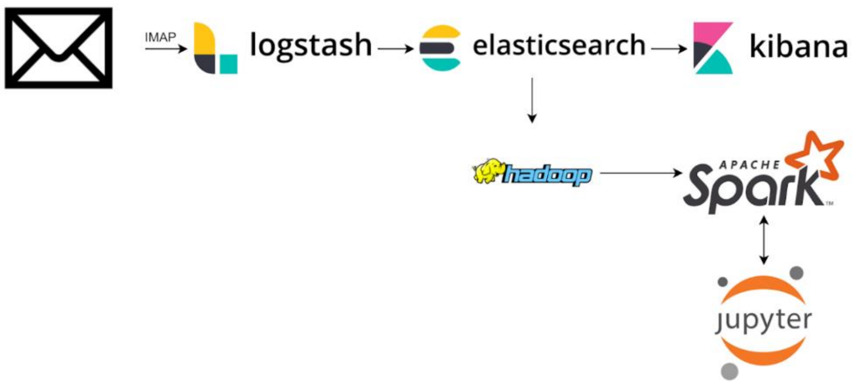


Figure 58 Jupyter Notebooks with Spark and ELK

The process shown in the figure is the following:

1. Emails are inputted into Logstash using the Logstash's functionality of reading emails using the IMAP protocol.
2. Logstash is a data collection engine with real-time pipelining capabilities. It is a data parser, where an input, a filter and an output parameter are set. The input parameter determines where the data is taken from, in our case from an IMAP email server. The filter parameter is practically a function with several methods on how the data is parsed. The output states where the data is sent, in this case to the Elasticsearch application.
3. Elasticsearch is a scalable full-text search and analytics engine. With it one can store, search and analyze large volumes of data in near real time. It is used as the underlying engine that another application uses to execute complex searches. It provides an API which is used by the next application, Kibana.
4. Kibana is an analytics and visualization graphical user interface designed to work with Elasticsearch. It provides with advanced data analysis capabilities and visualizations of the data in charts, tables and maps.
5. Hadoop is used as the communication means between Elasticsearch and Apache Spark.
6. Spark is used through Jupyter Notebooks, which is an application running on browsers and allows the creation and sharing of documents that contain live code, equations, visualization and narrative next. In our case, Jupyter Notebooks would be used to write the code of the python programming which uses Spark to build a machine learning classifier for phishing emails.

However, as mentioned, Jupyter is used for creating a sort of interactive with the user text which also includes pieces of code. While it is possible to be used as a code development application, it is not as effective as a dedicated text editor. That is the reason that Jupyter Notebooks was not used and Spyder was used instead.

Spyder [80] is included in the Anaconda distribution by default. It is an open source integrated development environment for scientific programing in Python. Spyder on its own includes a variety of relevant packages such as NumPy, Matplotlib, Pandas and others. It provides a plethora of features useful for science programming, including:

- An editor with the usual features (syntax highlighting, code completion, type introspection)
- Multiple IPython Consoles: IPython or Interactive Python is a command shell for interactive computing. It is an interactive shell with support for interactive data visualization and use of GUI toolkits, flexible and embeddable interpreters to load into other projects and tools for parallel computing.
- Edit variables in the GUI
- A debugger that allows for step-by-step execution and complete control of the running of the program.
- Static code analysis.

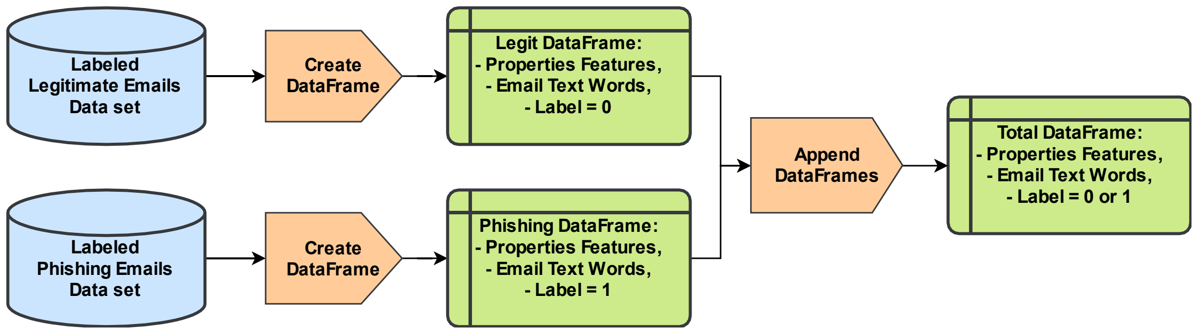
Spyder seemed to be a much more matching to the needs of the project application, and since it was already installed withing Anaconda, it is the application that is used for the implementation of this project. For the Spark tool, its Python API called PySpark.

5.2 Processes of the proposed architectures

The following chapters include the explanation of the different process which were developed for the implementation of the proposed architectures. These are presented at the last sub-chapters of the implementation chapter.

5.2.1 Total DataFrame creation

The following figure is a flowchart of the total DataFrame creation, which includes both legitimate and phishing emails. The data sets found were separate, one with phishing emails and one with legitimate emails, so the process of creating a DataFrame had to be executed twice.



Labeled Legitimate Emails Data set
Create DataFrame
Legit DataFrame: - Properties Features, - Email Text Words, - Label = 0
Labeled Phishing Emails Data set
Create DataFrame
Phishing DataFrame: - Properties Features, - Email Text Words, - Label = 1
Append DataFrames
Total DataFrame: - Properties Features, - Email Text Words, - Label = 0 or 1

Figure 59 Total DataFrame Creation Process

The “Create DataFrame” takes as input an mbox file and whether it consists of phishing or legitimate emails. It retrieves the properties-based features from every email and its text and creates a DataFrame with columns the properties-based features, the email text words as a vector and a column called “label” which has a value of 0 of the email is legitimate and 1 if the email is phishing. Once the “Create DataFrame” process is completed for both the data sets, the two resulting DataFrames include only legitimate and only phishing emails respectively. They are appended together into one total DataFrame which includes both kinds of emails.

5.2.2 Creating a consistence file format of the email data sets

As mentioned before, the email data sets that were found being used in the literature were stored in different file formats. More specifically, the phishing emails were stored using the mbox file format and the legitimate emails were stored with the maildir file format. In order to use the same algorithms for DataFrame creation for both data sets, an algorithm is built to convert the maildir format to mbox. The maildir file format stores emails in the following structure.

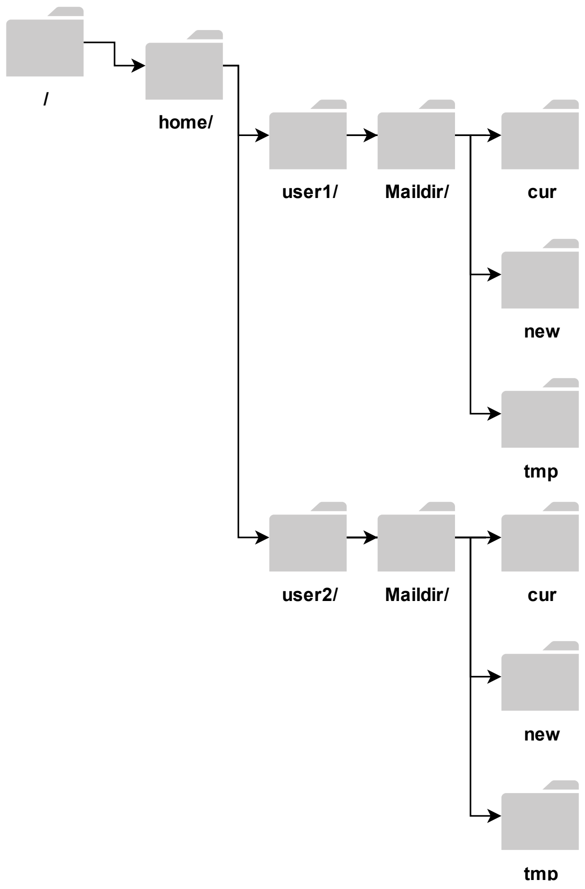


Figure 60 Maildir file format

The algorithm of the maildir to mbox file format conversion is shown below.

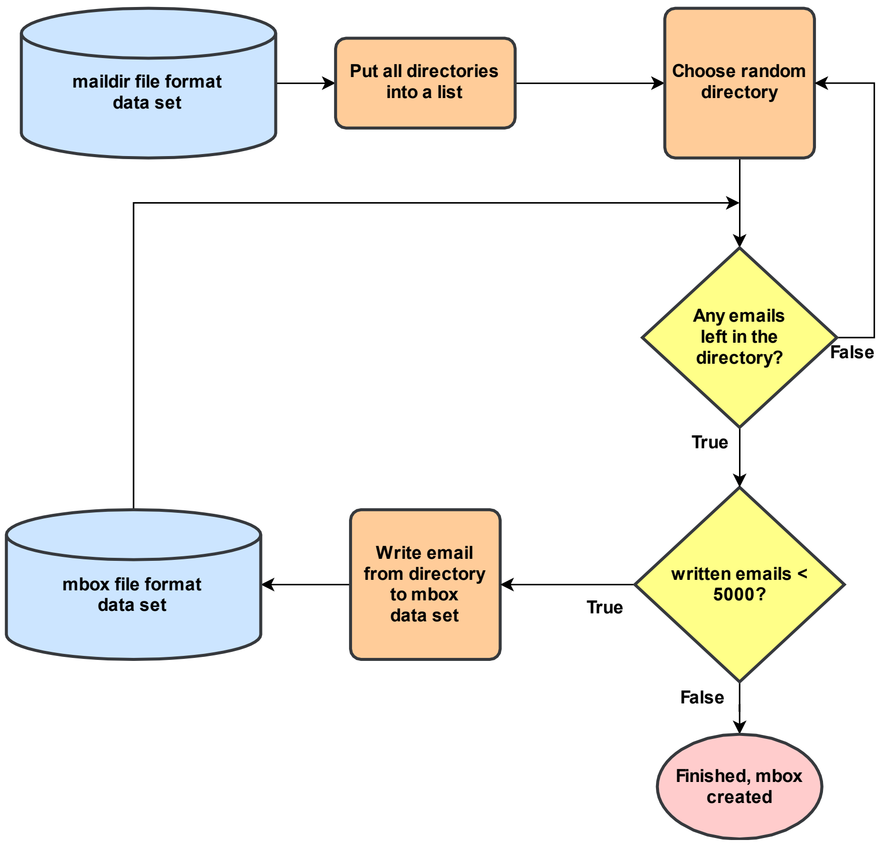

maildir file format data set
Put all directories into a list
Choose random directory
Any emails left in the directory?
False
True
written emails < 5000?
True
Write email from directory to mbox data set
mbox file format data set
False
Finished, mbox created

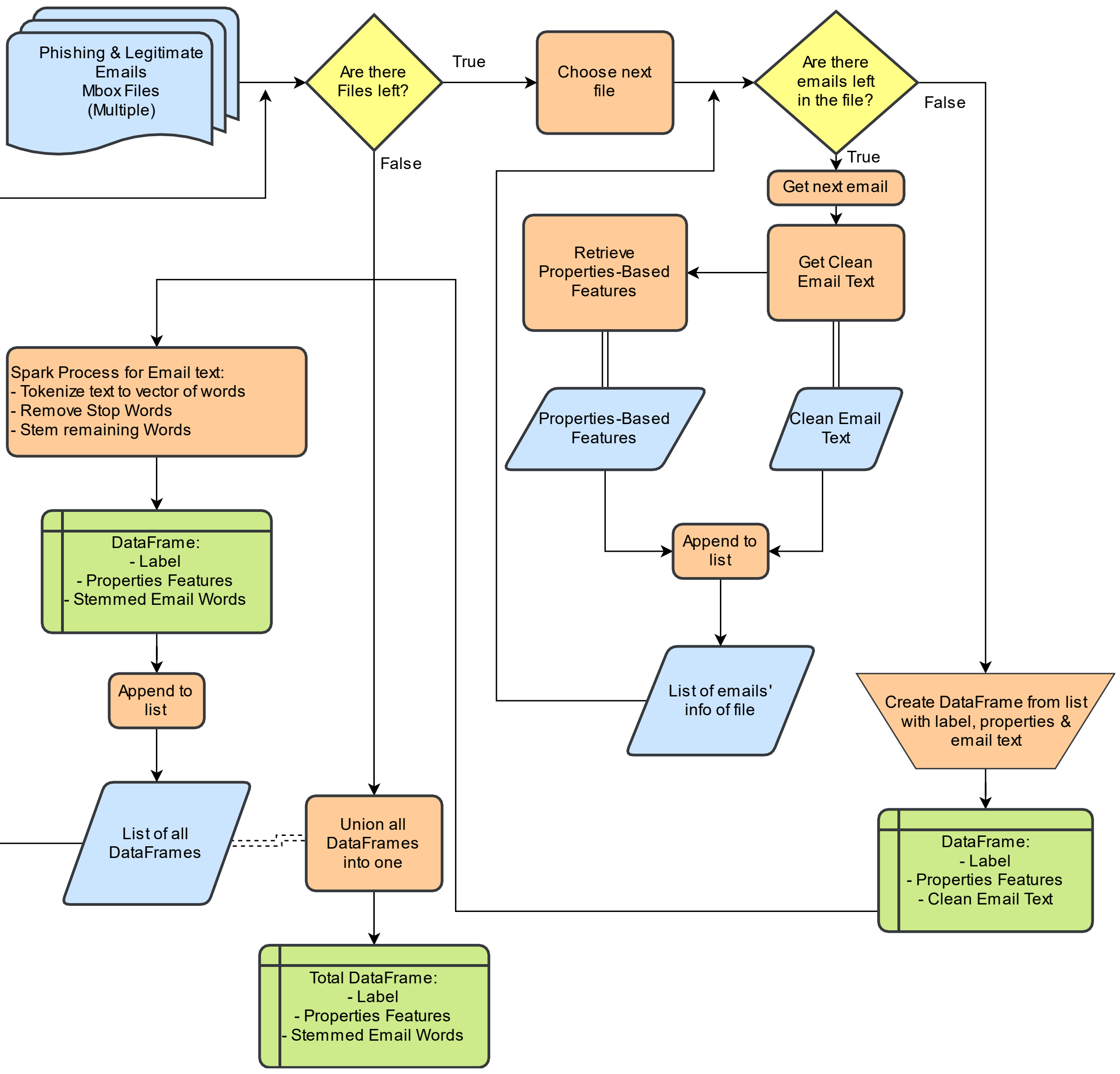
Figure 61 Maildir to Mbox file format conversion

The first step is to put all the directories into a list, from which the program can iterate directories to write to the mbox file. Because the directories are stored in alphabetical order, a random one is chosen to minimize the chances of any sort of bias. The program checks if the directory has any emails, and if yes then it checks if 5,000 emails have already been written to the mbox file. This limitation is put there in order to handle the size of the mbox file. Note here that this process was repeated ten times in order to create ten mbox files with each one having 5,000 emails, in order to have more than enough emails to train the machine learning classifier. The reason that not every email of the maildir file was not used is because the file had more than 500,000 emails which is too many emails, would not provide any additional useful information for the classifier and would require a long processing time. Back to the process itself, if the mbox file does not have 5,000 emails written in it, then an email from the chosen directory is written to the mbox file. This process of choosing random directories from the maildir file and writing emails from them to the mbox file is repeated until there are 5,000 emails written to the mbox file.

After completing that process, the result is that now there are mbox files for both legitimate and phishing emails, so the handling of them can be done using the same algorithm, which is described below.

5.2.3 "Create DataFrame" process explained

Now that there are only mbox files for the legitimate and phishing data sets, the next step is to use their contents to create a useable DataFrame for training a machine learning classifier with Spark. Note here that the data sets that are used as input are multiple mbox files, both in the case of legitimate emails and phishing emails. The previously shown Total DataFrame creation process was a higher-level presentation for better understand, and the actual process is slightly different. The process to create the Total DataFrame is the following.



Phishing & Legitimate Emails Mbox Files (Multiple)
Are there Files left?
True
False
Choose next file
Are there emails left in the file?
False
True
Get next email
Get Clean Email Text
Retrieve Properties-Based Features
Properties-Based Features
Clean Email Text
Append to list
List of emails' info of file
Create DataFrame from list with label, properties & email text
DataFrame: - Label - Properties Features - Clean Email Text
Spark Process for Email text: - Tokenize text to vector of words - Remove Stop Words - Stem remaining Words
DataFrame: - Label - Properties Features - Stemmed Email Words
Append to list
List of all DataFrames
Union all DataFrames into one
Total DataFrame: - Label - Properties Features - Stemmed Email Words

Figure 62 Total DataFrame Creation Process Detailed

There are multiple mbox files in different folders: one folder for phishing emails and one for legitimate emails. In the flowchart they are presented as one because it is the same process for both with the only difference being during the creation of the DataFrames, where the label is equal to one and zero respectively. The process is like follows:

1. Select an mbox file from folder.
2. Select an email from the mbox file

3. Get the Clean text from the email. The clean text is the text that is meant to be read by the receiver of the email. The process of getting the clean text will be presented in a different flowchart.
4. Get the properties-based features from the email.
5. Append the information retrieved from the email to a list which includes this information for all the emails of the currently selected mbox file.
6. If there are emails left in the mbox file, then go to step 2
7. Create a DataFrame using the list which has features and clean text from every email of the mbox file.
8. Use Spark’s functionality to process the email text of the emails and produce stemmed words arrays.
9. Append the DataFrame to a list which includes DataFrames of all the mbox files of the folders which include phishing and legitimate emails.
10. If there are mbox files left in the folder, then go to step 1.
11. Union the DataFrames from the list with all the DataFrames and produce one DataFrame with every email.

The properties-based features initially retrieved are the following:

1. **Number of URLs**: This is the number of URLs in the email, both included in any HTML part that might be in the email and in plaint text parts.
2. **Email encoding**: This is the encoding of the email itself. There is information about email encoding in a previous chapter.
3. **Number of email parts**: This is a number that counts the total number of the email. It accesses the email as a tree (in case there are more multipart parts) and reaches the leaves of the tree and counts them.
4. **Has HTML**: This is a boolean feature which is true if there is at least one part of the email which is text/html. HTML type emails means there are more capabilities for the writer and can include more malicious parts.
5. **Attachments**: This feature is the number of attachments in the email, which could potentially be malicious. It does not count the inlines, which are images etc. shown directly in the email.
6. **Bad Words**: This feature counts from the clean text of the email the number of words occurrences. Bad words are words which were considered to appear more frequently in phishing emails, thus contributing as a feature towards classification. Bad Words are the following: 'link', 'click', 'confirm', 'user', 'customer', 'client', 'suspend', 'restrict', 'verify' and 'protect'.
7. **IP URLs**: This is a counter of the URLs in the email which have direct IPs instead of domain names. This is because not having a domain name for the web server means that there is not enough effort put into the website and probably it is a temporary phishing website.
8. **Diffhref**: This feature refers to the emails’ parts which are html type. It compares the URLs with the hyperlink text which is displayed to the reader and if it is not the same then it increases the counter by one. The purpose of this feature is that phishing emails usually include hyperlink texts which are of legitimate websites but actually direct to a different, phishing website.
9. **Forms**: This feature counts the number of HTML-based forms in the email. This feature is because there are attempts to retrieve sensitive information through html forms in the emails by phishing attacks.

10. **Scripts**: This feature counts the occurrences of the ‘<script>’ substring in the email, which represents the presence of JavaScript code. JavaScript, much more than HTML, enhances the capabilities of the email writer and provides with more ways of including malicious parts in the email.
11. **Size**: This feature was not used because it was found that it did not have any correlation with the type of email (phishing or legitimate). It counts the total number of characters that the email has, when the email is written as a text file. This includes the attachments as binary files etc.

The “Stemmed” column has the processed clean text of the email, which has been tokenized, had its stop words removed and the rest of the words stemmed. The result is a vector of informative words which have had their influxes removed. The last step is to have consistency of words between their occurrences in the case of word-frequency-based algorithms. For example, if there was not the stemming step, the words “run” and “running” would be considered as different words, but with stemming they are now both “run” and they will be counted as the same word.

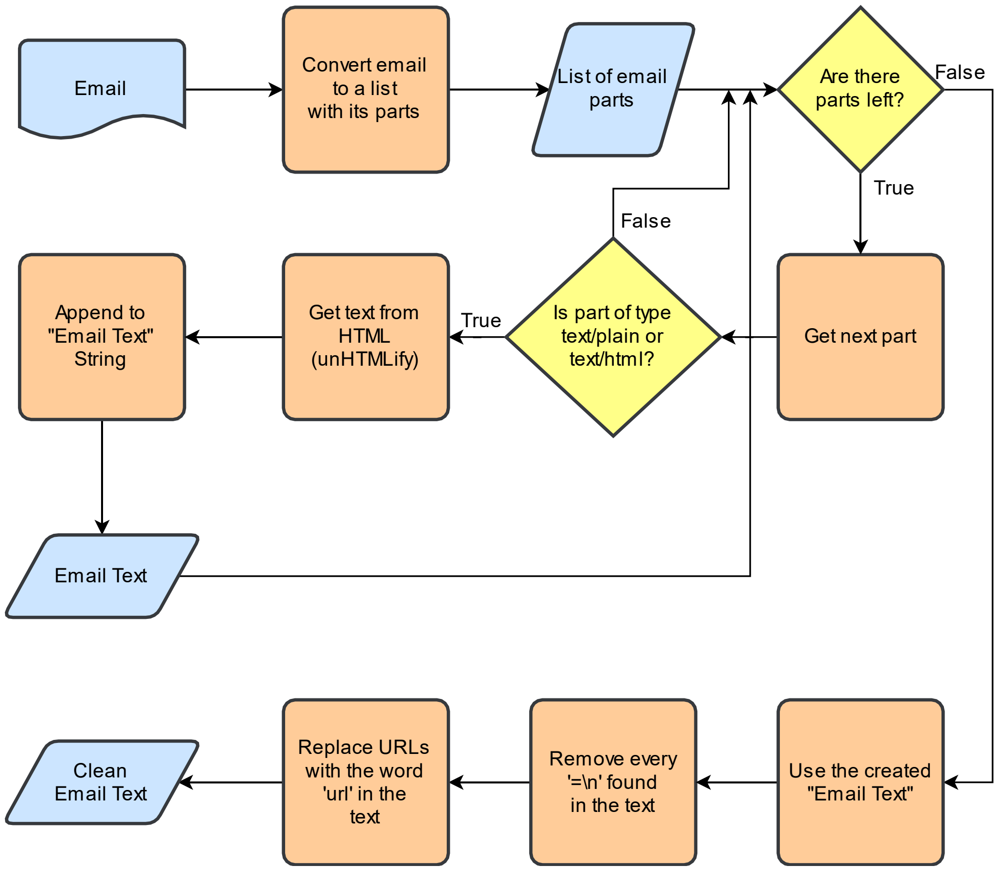
The DataFrames created looks like the following table. Note that the table is symmetric here. Normally the columns would represent features and the rows would represent examples/emails, but the table is symmetric here for presentation clarity, because otherwise it would be hard to read the 12 columns of features. This example DataFrame table has two examples (emails):

Table 8 DataFrame Format (two examples)

Label	1	0
Number of URLs	2	0
Email encoding	7bit	7bit
Number of emails parts	1	0
hasHTML	1	0
Attachments	0	0
Bad words	6	0
IP URLs	0	0
Diffhref	1	0
Forms	0	0
Scripts	0	0
Stemmed	[dear, valu, custom, record, indic, haven, submit, correct, inform, updat, account, pleas, updat, account, fill, tab, form, respond, email, repli, receiv, like, updat, ebay, account, pleas, use, link, url, thank, patienc, matter, regard, custom, support, trust, safeti, depart, ebay, inc]	[pay, well, pay, perform, bonus, base, merit, entitl, soundbit, prc, email, get, worri, mike, open, discuss, phillip]

5.2.4 Get Clean Text Process

The sub-process for getting the clean text from an email is shown in the figure below. By clean text it is meant that with emails usually the text that is to be shown to the receiver is written in different encodings or formats. For example, the text could be written within HTML parts, which would make the text-based machine learning classifiers take into account the HTML elements as words, which would make the classification less effective. This is because the elements as words would not provide any relevant information towards whether the email is phishing or legitimate.



Email
Convert email to a list with its parts
List of email parts
Are there parts left?
False
True
Get next part
Is part of type text/plain or text/html?
False
True
Get text from HTML (unHTMLify)
Append to "Email Text" String
Email Text
Use the created "Email Text"
Remove every '=\n' found in the text
Replace URLs with the word 'url' in the text
Clean Email Text

Figure 63 Get Clean Text from email Process

The steps of the process with more details are the following:

1. **Convert email to a list with its parts**: As mentioned in the chapter of the structure of emails, the email has a tree-like structure. It can be a multipart type email and will have more sub-parts with content, like the figure below. This sub-process will take all the leaf nodes from the email part tree and make a list with them.

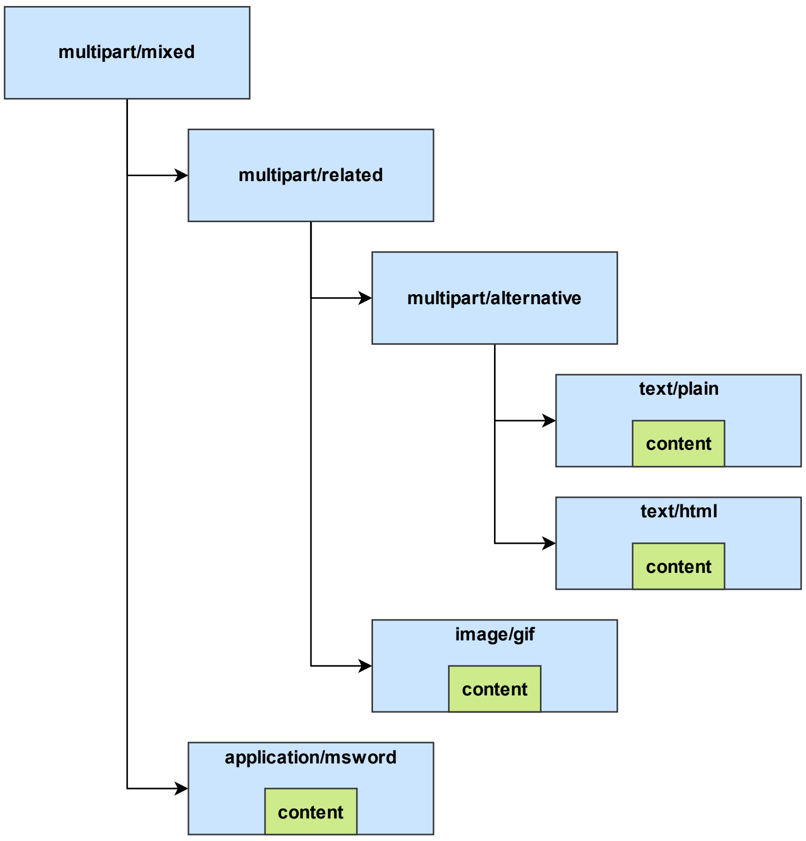


Figure 64 Mulitpart email structure

2. **Get the next part** from the list of the email parts.
3. If the part is not of type text/plain or text/html and there are more parts left in the list, go to step 2.
4. Get text from HTML (un-HTML-ify): Here a created function called “unhtmlify” is used. What it practically does is retrieve every textual information from an HTML file. This includes every part of text that was meant to be read by the receiver of the email. The reason that not only the text/html parts are getting “unhtmlified” but the text/plain are, too, is because it has been seen that there were parts that were “text/plain” but also included HTML code in them.
   This function uses the Beautiful Soup Python library [81] (version 4.4.0). This is a library for pulling data out of HTML and XML files.
5. Append to “Email Text” string. This string includes any text part that was meant to be read by the receiver. Note here that this text is not yet cleaned for a program to read from, because it has a few unusual symbols which can be interpreted by an email reader application but not as plain text.
6. If there are more parts left in the list, go to step 2.
7. At this step we now have all the textual information from the email that was meant to be read by the receiver, but some cleaning is required.
8. It has been noticed that a few emails had the character phrase “=\n” (equal sign and new line) after every certain number of characters. The character phrase would go in-between words and separate them, making the words unreadable by a computer program. This has been noticed to be happening when emails are sent with the PHPMailer, a code library to send emails via PHP code from a web server. PHPMailer is usually used on websites built

with WordPress, but it can be used on its own. The first step to clean up the email text is to remove every occurrence of the character phrase “=\n”.

9. Replace URLs with the word “url” in the text: Any URL that was in the text of the email does not provide any relevant information when it comes to what the sender wanted to say to the reader. Thus, every URL from the text is replaced with the word ‘url’. The reason that the URLs are not just removed but replaced with the “url” word is:
    a. To indirectly be a properties-based features, because text-based algorithms will use the word “url” and if it occurs frequently on phishing emails it will be used as a word with higher weight.
    b. To help the more sophisticated algorithms (word embedding, Recurrent Neural Networks), which take into account phrases of words, make more sense of what was meant to be said with the text.
10. The result is a clean text, ready to be further processed and used by text-based algorithms and classification.

5.2.5 Retrieve properties-based features Process

The sub-process that retrieves the properties-based features is presented at this point. The properties-based features functions are implemented as classes, which have the following structure.

```
class feature:
   def get_feat(self, msg):
      ##get feature
   def get_name(self):
      return 'feature name'
```

Every properties-based feature class has the same structure. For example, the “encoding” feature which returns the content transfer encoding that was used to send the email is the following class:

```
class encoding:
    def get_feat(self, msg):
        enc = msg["Content-Transfer-Encoding"]
        if(not enc):
            enc = 'none'
        return enc
    def get_name(self):
        return 'encoding'
```

The reasoning for building the functions like this is to simplify the process of getting the feature and the corresponding feature name. This is useful when retrieving the features from every email, writing them in a list and building a DataFrame using the list and the feature names as column names. The process for retrieving the properties-based features is the following:

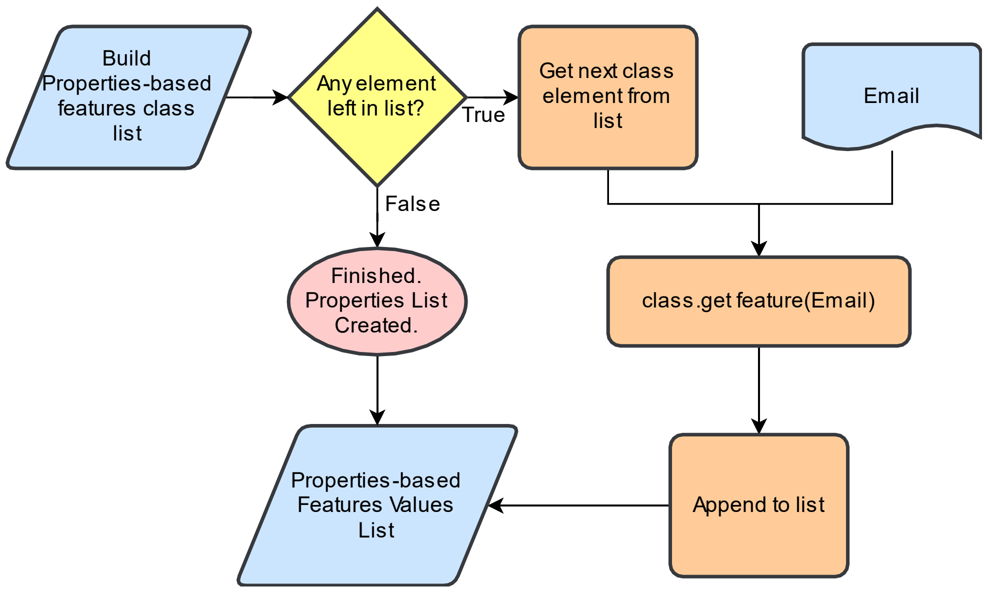

Build Properties-based features class list
Any element left in list?
True
False
Get next class element from list
Email
Finished. Properties List Created.
class.get feature(Email)
Properties-based Features Values List
Append to list

Figure 65 Properties-Based Features Retrieval Process

The steps with more details are the following:

1. Build the Properties-based feature class list: This list includes the classes of all the properties-based features. The classes have the structure shown before.
2. Get the next class from the list.
3. Call the function of the feature retrieval of the class, with input the email which is to have its features retrieved.
4. Append the output to the Properties-based features values list. This list includes the outputs of the features' functions. Going back to the "Create DataFrame" process, this list will be appended to the output of the clean text and the label value and the DataFrame will be created.
5. If there are elements in the feature class list (not the value list) then go to step 2.
6. After being done, there will be an output list which includes the values of all the properties-based features for the inputted email.

At this point every part of the process of creating the total DataFrame has been explained. After having a DataFrame with all the properties-based features, the stemmed words of the clean email text and the label value (zero if the email is legitimate and one if the email is phishing), the next step is to use that DataFrame to train and test different classifiers.

5.2.6 Proposed hybrid "assembled" architecture

Now that there is an available DataFrame which include both legitimate and phishing emails, the process of training and testing different classifiers will be presented. The process of finding the best combination of features, classifier and their hyperparameters combination is shown in the following figure. The proposed "assembled" architecture is tested here, where the properties-based features are combined with the text-based features into one consistent feature vector which is used as the input for the training process of the machine learning classifiers.

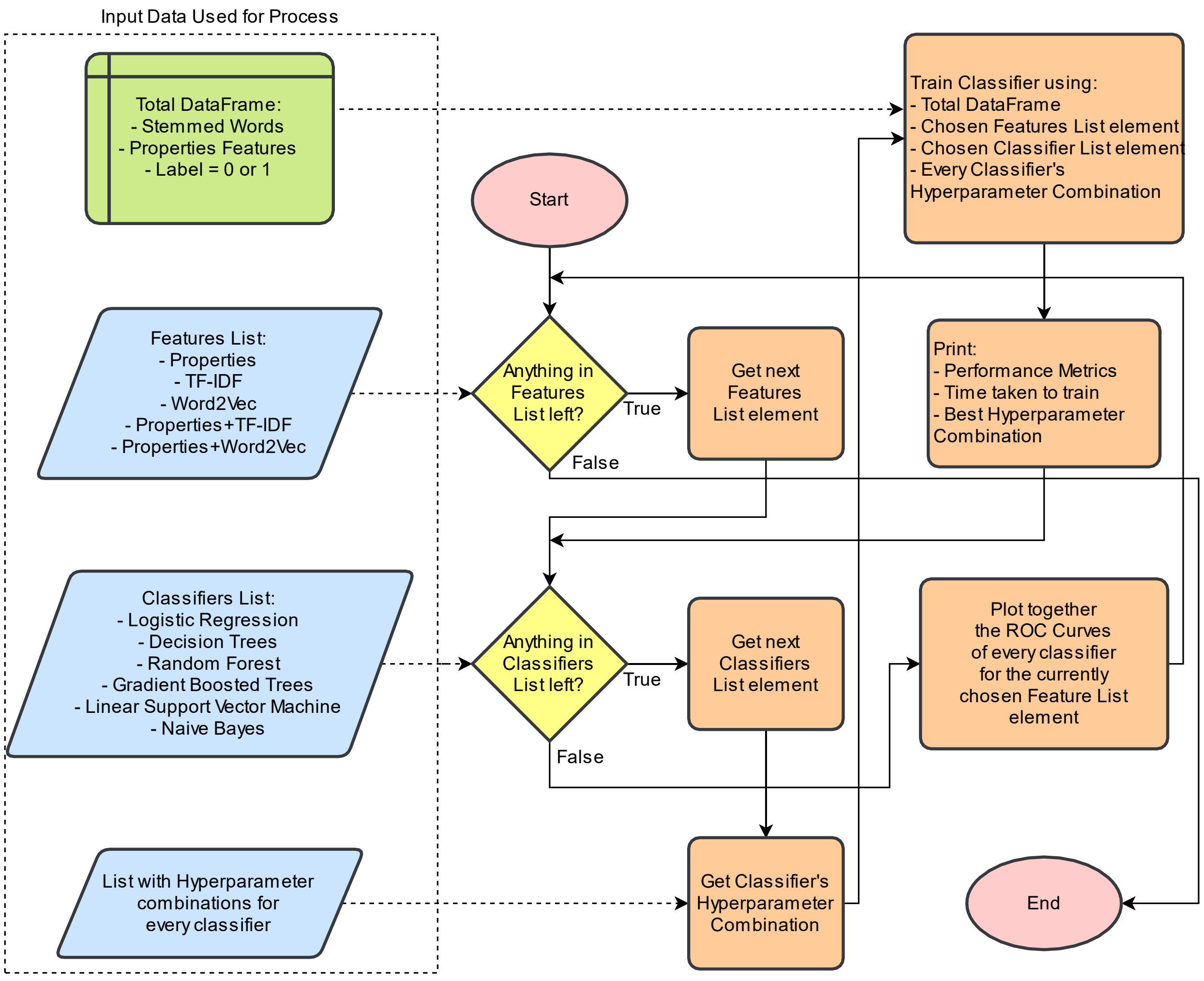

Input Data Used for Process
Total DataFrame:
- Stemmed Words
- Properties Features
- Label = 0 or 1
Features List:
- Properties
- TF-IDF
- Word2Vec
- Properties+TF-IDF
- Properties+Word2Vec
Classifiers List:
- Logistic Regression
- Decision Trees
- Random Forest
- Gradient Boosted Trees
- Linear Support Vector Machine
- Naive Bayes
List with Hyperparameter combinations for every classifier
Start
Anything in Features List left?
True
False
Get next Features List element
Anything in Classifiers List left?
True
False
Get next Classifiers List element
Get Classifier's Hyperparameter Combination
Train Classifier using:
- Total DataFrame
- Chosen Features List element
- Chosen Classifier List element
- Every Classifier's Hyperparameter Combination
Print:
- Performance Metrics
- Time taken to train
- Best Hyperparameter Combination
Plot together the ROC Curves of every classifier for the currently chosen Feature List element
End

Figure 66 Process to find the best features/classifier/hyperparameters combination

The steps of the process, in more detail, as the following:

1. First of all, there is the **Total DataFrame**, which was created from a previous process. It includes emails transformed into properties-based features, the stemmed words of the email text (the text which is to be read by the receiver of the email) and the label value (zero if the email is legitimate and one if the email is phishing).
2. There are **lists** which are to be used to find the best performing combination. These lists are the following:
    a. **A Features List**: This list includes what features the classifier will use to train itself and later make classifications with. Note here that although the properties-based features are already retrieved from each email and placed at the Total DataFrame, the text-based features are not retrieved yet. There are two reasons for that. The first one is that there are two major methods for retrieving text-based features (Term Frequency – Inverse Document Frequency or TF-IDF and Word Embedding with its implementation function called Word2Vec) which yield different features, so

    at each test the method needs to be determined first. The second reason for not retrieving the text-based features beforehand is that Spark offers Pipeline-based methods for their retrieval. Because the training and testing phases in Spark are completed using Pipeline-based methods, too, it is more efficient to have one long Pipeline than two shorter ones. The Pipeline stages will be presented in a later figure.
    b. **A Classifiers List**: This list includes the classifiers which are provided by the Spark's MLlib library.
    c. **A list with the hyperparameter combinations of the classifiers**: This list is used together with the classifiers list. It includes sets of values for the hyperparameters of every classifier. When training a classifier in this process, the training actually happens for every hyperparameter value combination. In the end, the hyperparameter value combination with the best performance is chosen.
3. Now that the data used for the process is ready, the process can start. The first step is to get the next list element from the ***Features list***.
4. Get the next list element from the ***Classifiers list***.
5. Get the **hyperparameter values combinations** for the just chosen classifier from the list.
6. **Train the classifier** using:
    a. The created Total DataFrame
    b. The chosen feature list element which represents the features that are to be used as input for the classifier.
    c. The chosen classifier
    d. The classifier's corresponding hyperparameter value combinations
7. After the training is done, **print the performance results**:
    a. Different performance metrics (accuracy, precision, f1measure etc.)
    b. The time taken for the training to complete and for the performance metrics to be calculated. Note here that the performance metrics calculations took a time comparable to the training time.
    c. The hyperparameter value combination with the best performance.
8. If there is any element left in the ***Classifiers List,*** then go to **Step 4.**
9. **Plot together the ROC curves** of the trained classifiers for the features which are currently chosen as input for the training and classification. This is one figure for each feature input type and includes the ROC curves of every classifier for that input type. It is useful to get a more graphical representation of the performance of every classifier and a different comparison means.
10. If there is any element left in the ***Features List***, then go to **Step 3**.

After the process is done, the **final results** are the following:

- **Performance metrics for every features/classifier/hyperparameter combination**. This includes:
    - Area under the ROC curve
    - The confusion matrix of the predictions
    - Accuracy
    - Precision
    - Recall
    - F1Measure
    - False Positive Rate
    - False Negative Rate

- **ROC Curves** for every classifier plotted together for each feature type input.

5.2.6.1 Pipeline Stages for the training of the classifiers

As mentioned previously, the training process of the classifiers involves the Pipeline functionality of Spark to streamline the process and make it more efficient. The stages of the Pipeline were different for each case of feature inputs and classifier. The stages for each case are presented in the following. Note here that detailed descriptions of the functionality of every stage used and other useful stages which are provided by Spark are presented in the Approach chapter.

Properties-Based Features Stages:

1. String Indexer: This takes the categorical (string) type features and encodes them to label indices. Each different string of the categorical feature takes a different numerical value in the range [0, number of different labels).
2. One Hot Encoder Estimator: This transformer takes as input categorical feature, represented as a label index to a binary vector which has a value of zero at every index except a value of one at the index which represents the specific categorical feature value of every case.
3. Vector Assembler: This transformer takes as input all the selected feature columns and combines them into a single vector column. The output column from this transformer is called "features" and will be used as input for the different classifiers.
4. Classifier: This is any classifier from the aforementioned Classifiers List.

The output is a DataFrame which includes the input *Total DataFrame* and additional columns of raw prediction, probability, and prediction.

TF-IDF Stages:

1. HashingTF: This is a Transformer which takes sets of terms and converts them to fixed-length feature vectors.
2. IDF: This is an Estimator which fits on a dataset and produces an IDFModel, which takes feature vectors from the HashingTF output and scaled each feature with higher weights to the features which appear more rarely in the data set. The output of this model is a column of features which are to be used as input by the classifiers.
3. Classifier

Word Embedding Stages:

1. Word2Vec: This is an Estimator which takes sequences of words and trains a Word2VecModel. This model maps each word to a unique fixed-size vector. It transforms each document into a vector using the average of all words in the document. The vector is used as the features column for the classification process.
2. Classifier

Hybrid – Properties + TF-IDF Stages:

This includes the stages of the properties-based stages and the TF-IDF stages. Their outputs are assembled together into a hybrid feature set.

1. String Indexer
2. One Hot Encoder Estimator
3. Vector Assembler
4. HashingTF
5. IDF
6. Total Vector Assembler: This assembler takes the outputs of the Steps 3 and 5 and assembles them together into one consistent hybrid feature set, so that the classifier can then take into account both the properties of the email as well as its text for the classification process. The output is one feature column which combines both properties-based and text-based features.
7. Classifier

Hybrid – Properties + Word Embedding Stages:

1. String Indexer
2. One Hot Encoder Estimator
3. Vector Assembler
4. Word2Vec
5. Total Vector Assembler: Same as the previous method, but now the text-based features are provided by the word embedding algorithm which produces a more sophisticated feature set of the text.
6. Classifier

The results of the proposed training process should give a better perspective as to which combination is performing better and if implementing a hybrid-based model, which involves using both properties-based and text-based features, improves the performance. In the next chapter (Numerical Results) there are the numerical results of the experiment.

5.2.7 Proposed hybrid “stacked” architecture.

Another implementation model which was tested was one of the stacked architecture. As mentioned in a previous chapter, a stacked model is a type of ensemble classifier, which combines multiple simple classifiers, takes their prediction outputs and uses them as inputs/features for a total classifier which will make the final classification. The proposed model is slightly different, where there is a classifier at first using text-based features and is trained. A new “total” classifier will then use the first classifier’s output as a feature in addition to having the default properties-based features. The architecture is shown in the following figure.

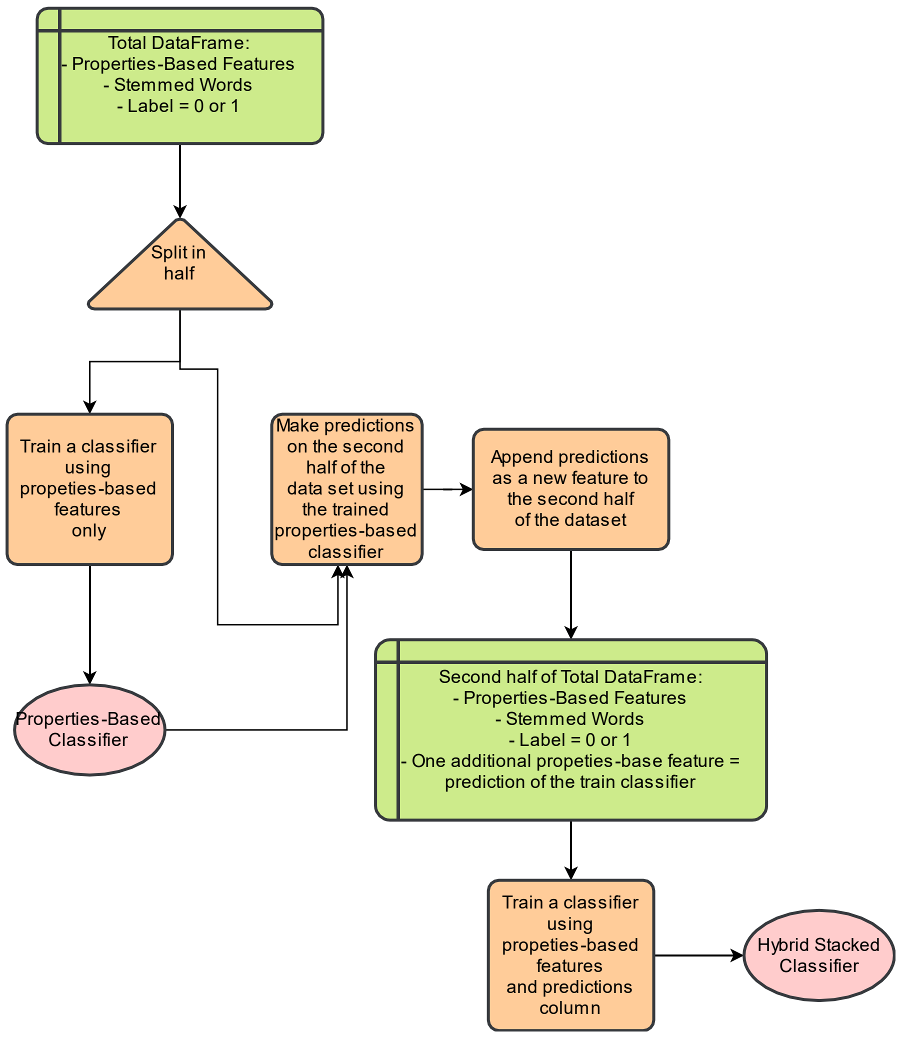

Total DataFrame:
- Properties-Based Features
- Stemmed Words
- Label = 0 or 1
Split in half
Train a classifier using propeties-based features only
Make predictions on the second half of the data set using the trained properties-based classifier
Append predictions as a new feature to the second half of the dataset
Properties-Based Classifier
Second half of Total DataFrame:
- Properties-Based Features
- Stemmed Words
- Label = 0 or 1
- One additional propeties-base feature = prediction of the train classifier
Train a classifier using propeties-based features and predictions column
Hybrid Stacked Classifier

Figure 67 Hybrid Stacked Model

Having thoroughly described the proposed architecture, the next step is to test them and compare their performance in order to determine whether the proposed architectures improve the performance when compared to simpler classifiers and which combination of features/classifier/hyperparameters produces the best results. The performance results of the implementation are presented in the next chapter.

6 Results

In this chapter the results of the implementation described before will be presented. More specifically there are execution times of each different algorithm executed, performance results for different features-classifier combination as well as performance results with different hyperparameters set for every feature retriever and classifier. The structure of the results is as follows. Because there are too many combinations, they could not be presented in one table, so there are multiple tables, one for each different used feature type.

The performance metrics are the following (they are calculated in percentages %):

- Area under the ROC Curve
- Accuracy
- Precision
- Recall
- F1Measure
- False Positive Rate (FPR)
- False Negative Rate (FNR)
- Confusion Matrix (this is not in percentages but rather the actual number of correct and wrong predictions).
- Time required to train the classifier and calculate the performance metrics.

6.1 Properties-Based Features

The performance results of every classifier and their best performing hyperparameter combination using the properties-based features is presented here. In the end there are the ROC curves of every classifier.

Table 9 Properties-Based Features Performance (values are in % except confusion matrix)

Algorithm	AUC	Confusion Matrix	Accuracy	Precision	Recall	F1 Measure	FPR	FNR	Time (m)
Logistic Regression	95.4	40 148 35 371	90.3	91.4	88.5	90.0	8.0	11.5	3.0
Decision Trees	97.2	444 12 13 412	97.2	96.9	97.2	97.1	2.8	2.8	3.91
Random Forest	99.5	458 10 6 451	98.3	98.7	97.8	98.3	1.3	2.2	3.59
Gradient Boosted Trees	99.6	452 9 6 417	98.3	98.6	97.9	98.2	1.3	2.1	3.26
Linear Support Vector Machine	96.0	381 40 28 408	92.1	93.6	91.1	92.3	6.8	8.9	13.89
Naïve Bayes	5.0	398 168 22 164	77.7	92.3	61.1	73.5	5.2	38.9	2.61

The parameter grid for each algorithm and the best performing hyperparameter combination are presented below.

Table 10 Properties + Logistic Regression best parameters

Parameter	Explanation	Values Tested	Best Performing
regParam	regularization parameter (>= 0)	0.1, 0.01	0.01
elasticNetParam	the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty	0.0, 0.5, 1.0	0.0

Table 11 Properties + Decision Trees best parameters

Parameter	Explanation	Values Tested	Best Performing
maxDepth	Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes.	2, 5, 10	2
minInfoGain	Minimum information gain for a split to be considered at a tree node.	0.0, 0.1	0.1
maxBins	Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.	6, 32	6

Table 12 Properties + Random Forest best parameters

Parameter	Explanation	Values Tested	Best Performing
numTrees	Number of trees to train (>= 1)	5, 20	20
maxDepth	Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node	2, 5	5

Table 13 Properties + Gradient Boosted Trees best parameters

Parameter	Explanation	Values Tested	Best Performing
maxDepth	Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node	2, 5	5
maxIter	maximum number of iterations (>= 0)	5, 20	20
stepSize	Step size (a.k.a. learning rate) in interval (0, 1] for shrinking the contribution of each estimator.	0.01, 0.1	0.1

Table 14 Properties + Linear Support Vector Machines best parameters

Parameter	Explanation	Values Tested	Best Performing
threshold	threshold in binary classification prediction applied to rawPrediction	0.0, 0.05	0.0

Table 15 Properties + Naive Bayes best parameters

Parameter	Explanation	Values Tested	Best Performing
smoothing	The smoothing parameter	0.2, 0.5, 1.0	0.2

The following figure includes the ROC Curves of every classifier for the classification using properties-based only features.

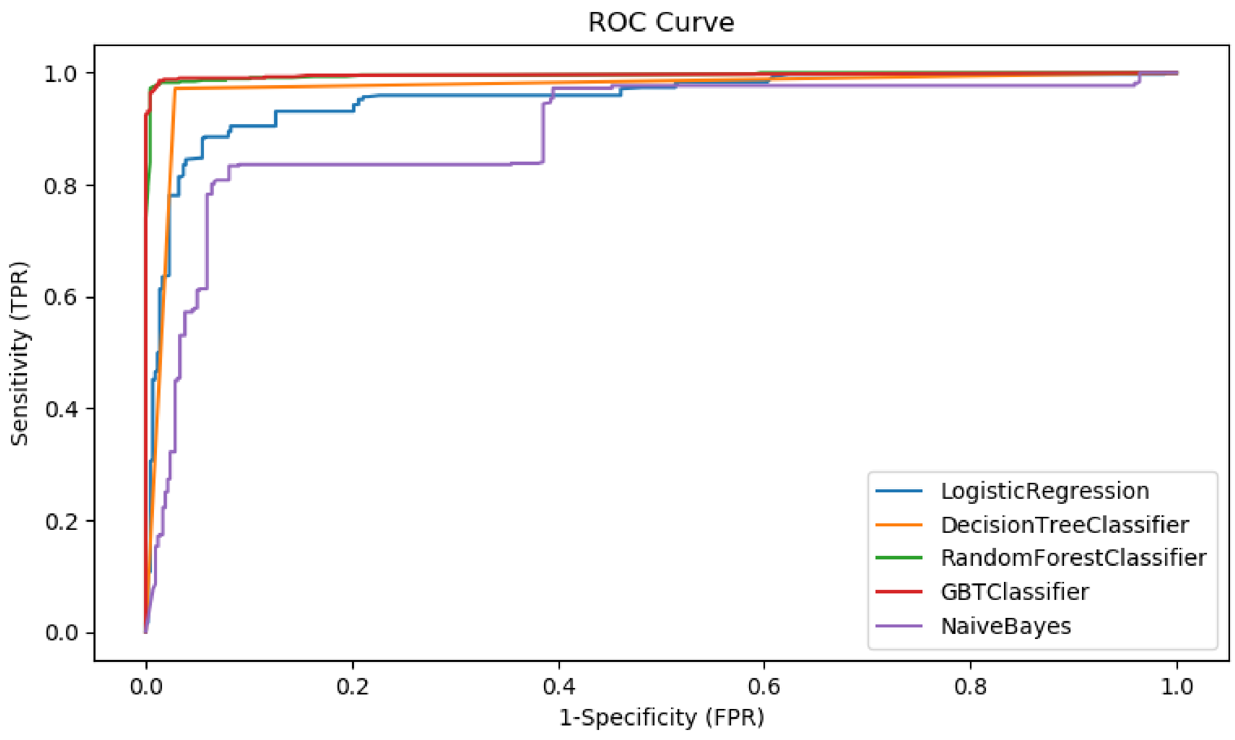


Figure 68 ROC Curves using Properties-based features

From the ROC curves it can be seen that the Naïve Bayes classifier has the worst performance, which can be confirmed by the performance metrics, which are significantly worse than all the other classifiers. Note here that the Support Vector Machines classifier was not included in the ROC curves because the ROC curves require a *probability* value to be produced by the classifier which shows the probability that an example will be in a certain class. However, the Support Vector machines do not predict like that, as it is shown in the theoretical background chapter.

The best performing classifiers are the Gradient Boosted Trees and the Random Forest classifier.

As it is shown, the performance metrics are already significantly high, meaning that the classifiers are already performing exceptionally well. This is a sign of probable overfitting, which is to be blamed on the datasets found in the literature. No other data sets have been found in the literature that are more "difficult" to classify, in order to build classifiers with better generalization. However, even if there is a case of overfitting, there can be a comparison among every classifier combination in order to find the best performing combination and whether creating a hybrid model where there are both properties-based and text-based features would increase the performance.

6.2 TF-IDF (Text-Based) Features

The performance results of every classifier/hyperparameter combination for the features retrieved using the Term Frequency – Inverse Document Frequency algorithm is shown in the following table.

Table 16 TF-IDF Features Performance (values are in % except confusion matrix)

Algorithm	AUC	Confusion Matrix	Accuracy	Precision	Recall	F1 Measure	FPR	FNR	Time (m)
Logistic Regression	99.1	405 23 20 400	94.9	95.2	94.6	94.9	4.7	5.4	2.79
Decision Trees	94.0	388 23 63 403	90.2	86.5	94.6	90.4	14.0	5.4	3.84
Random Forest	98.1	413 47 32 394	91.1	92.5	89.3	90.9	7.2	10.7	3.84
Gradient Boosted Trees	99.1	459 54 11 438	93.2	97.6	89.0	93.1	2.3	11.0	3.92
Linear Support Vector Machine	99.5	436 36 11 399	94.7	97.3	91.7	94.4	2.5	8.3	12.16
Naïve Bayes	62.1	390 83 41 372	86.0	90.0	81.8	85.7	9.5	18.2	2.84

The parameter grid for each algorithm and the best performing hyperparameter combination are presented below.

Table 17 TF-IDF + Logistic Regression best parameters

Parameter	Explanation	Values Tested	Best Performing
regParam	regularization parameter (>= 0)	0.1, 0.01	0.01
elasticNetParam	the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty	0.0, 0.5, 1.0	0.0

Table 18 TF-IDF + Decision Trees best parameters

Parameter	Explanation	Values Tested	Best Performing
maxDepth	Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes.	2, 5, 10	10
minInfoGain	Minimum information gain for a split to be considered at a tree node.	0.0, 0.1	0.0
maxBins	Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.	6, 32	6

Table 19 TF-IDF + Random Forest best parameters

Parameter	Explanation	Values Tested	Best Performing
numTrees	Number of trees to train (>= 1)	5, 20	20
maxDepth	Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node	2, 5	5

Table 20 TF-IDF + Gradient Boosted Trees best parameters

Parameter	Explanation	Values Tested	Best Performing
maxDepth	Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node	2, 5	5
maxIter	maximum number of iterations (>= 0)	5, 20	20
stepSize	Step size (a.k.a. learning rate) in interval (0, 1] for shrinking the contribution of each estimator.	0.01, 0.1	0.1

Table 21 TF-IDF + Linear Support Vector Machines best parameters

Parameter	Explanation	Values Tested	Best Performing
threshold	threshold in binary classification prediction applied to rawPrediction	0.0, 0.05	0.05

Table 22 TF-IDF + Naive Bayes best parameters

Parameter	Explanation	Values Tested	Best Performing
smoothing	The smoothing parameter	0.2, 0.5, 1.0	0.2

The following figure includes the ROC Curves of every classifier for the classification using properties-based only features.

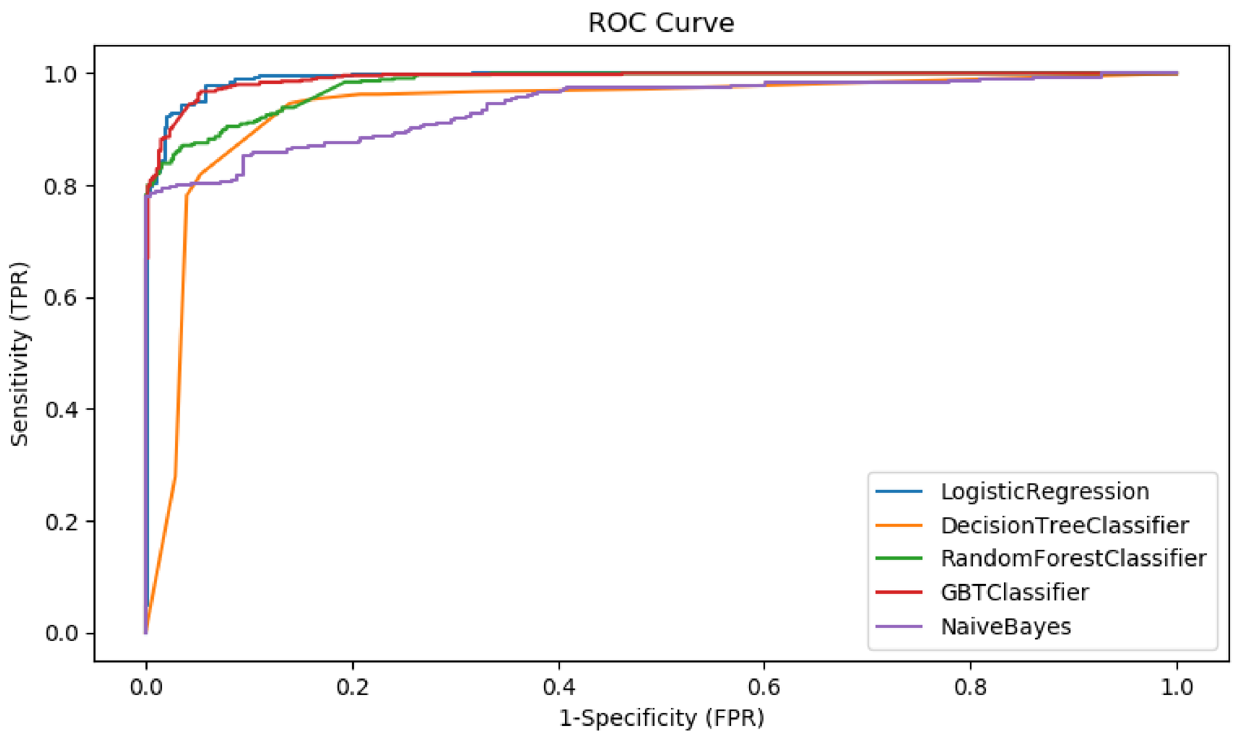


Figure 69 ROC Curves using TF-IDF Features

From the ROC curves figure it can be seen that the worst performing classifier is the Naïve Bayes one. The best performing classifiers are three: Gradient Boosted Trees, Logistic Regression and Linear Support Vectors, all with similar performance. Note here again that Support Vector Machines do not produce probabilities when classifying examples, and thus cannot have a ROC curve unless artificially creating one.

6.3 Word Embedding (Text-Based) Features

The performance results of every classifier/hyperparameter combination for the features retrieved using the Word Embedding (Word2Vec) algorithm are shown in the following table.

Table 23 Word2Vec Features Performance (values are in % except confusion matrix)

Algorithm	AUC	Confusion Matrix	Accuracy	Precision	Recall	F1 Measure	FPR	FNR	Time (m)
Logistic Regression	99.3	449 9 7 459	98.3	98.5	98.1	98.2	1.5	1.9	3.66
Decision Trees	98.2	402 13 15 397	96.6	96.4	96.8	96.9	3.6	3.2	4.81
Random Forest	99.7	405 10 10 410	97.6	97.6	97.6	97.6	2.4	2.4	4.33
Gradient Boosted Trees	99.4	448 15 16 441	96.6	96.5	96.7	96.6	3.4	3.3	4.62
Linear Support Vector Machine	99.7	442 5 9 408	98.4	97.8	98.8	98.3	2.0	1.2	12.64
Naïve Bayes	66.9	322 38 126 387	81.2	75.4	91.1	82.5	28.1	8.9	3.64

The parameter grid for each algorithm and the best performing hyperparameter combination are presented below.

Table 24 Word2Vec + Logistic Regression best parameters

Parameter	Explanation	Values Tested	Best Performing
regParam	regularization parameter (>= 0)	0.1, 0.01	0.01
elasticNetParam	the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty	0.0, 0.5, 1.0	0.0

Table 25 Word2Vec + Decision Trees best parameters

Parameter	Explanation	Values Tested	Best Performing
maxDepth	Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes.	2, 5, 10	10
minInfoGain	Minimum information gain for a split to be considered at a tree node.	0.0, 0.1	0.0
maxBins	Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.	6, 32	6

Table 26 Word2Vec + Random Forest best parameters

Parameter	Explanation	Values Tested	Best Performing
numTrees	Number of trees to train (>= 1)	5, 20	20
maxDepth	Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node	2, 5	5

Table 27 Word2Vec + Gradient Boosted Trees best parameters

Parameter	Explanation	Values Tested	Best Performing
maxDepth	Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node	2, 5	5
maxIter	maximum number of iterations (>= 0)	5, 20	20
stepSize	Step size (a.k.a. learning rate) in interval (0, 1] for shrinking the contribution of each estimator.	0.01, 0.1	0.1

Table 28 Word2Vec + Linear Support Vector Machines best parameters

Parameter	Explanation	Values Tested	Best Performing
threshold	threshold in binary classification prediction applied to rawPrediction	0.0, 0.05	0.0

Table 29 Word2Vec + Naive Bayes best parameters

Parameter	Explanation	Values Tested	Best Performing
smoothing	The smoothing parameter	0.2, 0.5, 1.0	0.2

The following figure includes the ROC Curves of every classifier for the classification using properties-based only features.

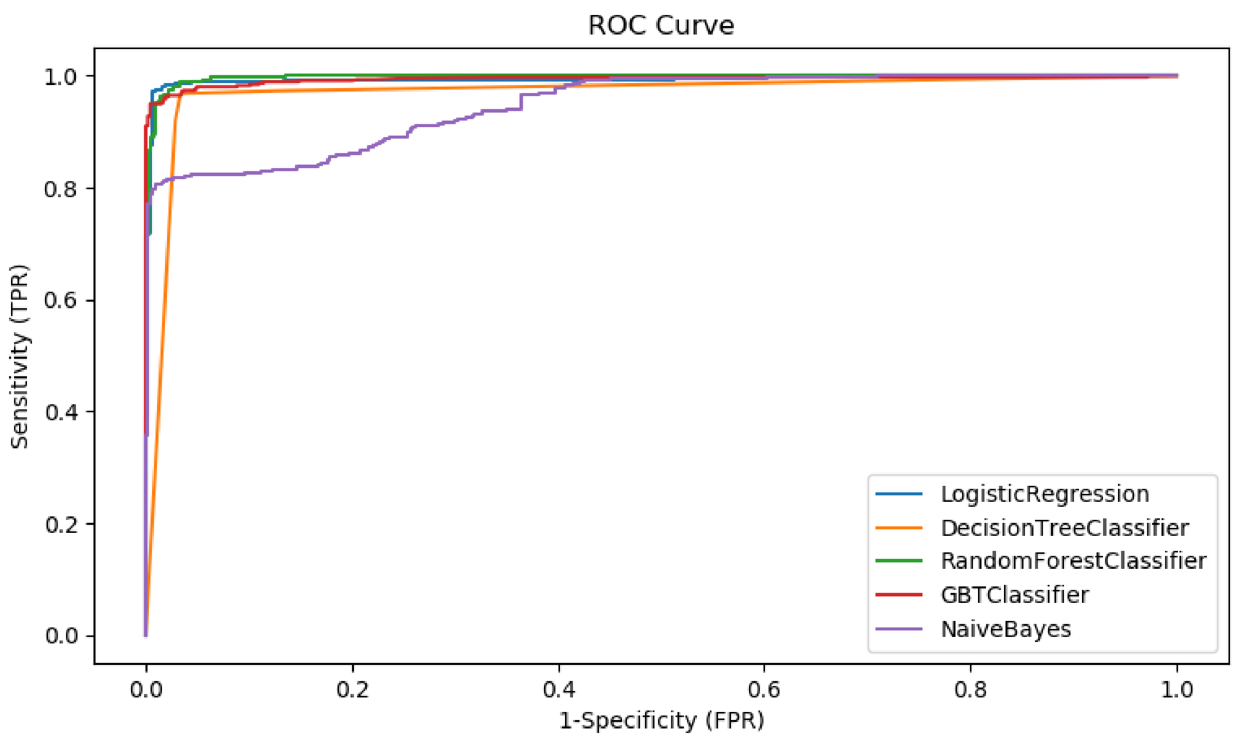


Figure 70 ROC Curves using Word Embedding features

It becomes apparent in this case that word embedding provides with more information when it comes to text classification, as it has already been mentioned in its theoretical description. Word Embedding is a more sophisticated version than TF-IDF, because, in addition to term frequency information, it also provides information of the words' meanings. And it can be seen from the results, where almost every classifier performed equally well. Naïve Bayes is the worst performing one due to its simplistic classification process.

6.4 Proposed Hybrid “assembled” Architecture: Properties-based and TF-IDF (Text-based) hybrid features

The performance results for every classifier/hyperparameter combination for the hybrid features retrieved from the properties of the email and the Term Frequency – Inverse Document Frequency algorithm are shown in the following table.

Table 30 Properties-based + TF-IDF Features Performance (values are in % except confusion matrix)

Algorithm	**AUC**	**Confusion Matrix**	**Accuracy**	**Precision**	**Recall**	**F1 Measure**	**FPR**	**FNR**	**Time (m)**
Logistic Regression	99.9	455 7 2 414	99.0	99.5	98.3	98.9	0.4	1.7	3.2
Decision Trees	94.0	460 7 1 435	99.1	99.7	98.4	99.1	0.2	1.6	4.24
Random Forest	99.9	391 4 2 474	99.3	99.6	99.2	99.4	0.5	0.8	4.27
Gradient Boosted Trees	99.7	471 14 1 428	98.3	99.8	96.8	98.3	0.2	3.2	4.1
Linear Support Vector Machine	99.9	437 5 2 422	99.2	99.5	98.8	99.2	0.2	1.2	12.32
Naïve Bayes	62.7	431 52 12 404	92.9	97.1	88.6	92.7	2.7	11.4	3.18

The parameter grid for each algorithm and the best performing hyperparameter combination are presented below.

Table 31 Properties-based + TF-IDF + Logistic Regression best parameters

Parameter	**Explanation**	**Values Tested**	**Best Performing**
regParam	regularization parameter (>= 0)	0.1, 0.01	0.01
elasticNetParam	the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty	0.0, 0.5, 1.0	0.5

Table 32 Properties-based + TF-IDF + Decision Trees best parameters

Parameter	Explanation	Values Tested	Best Performing
maxDepth	Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes.	2, 5, 10	10
minInfoGain	Minimum information gain for a split to be considered at a tree node.	0.0, 0.1	0.0
maxBins	Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.	6, 32	6

Table 33 Properties-based + TF-IDF + Random Forest best parameters

Parameter	Explanation	Values Tested	Best Performing
numTrees	Number of trees to train (>= 1)	5, 20	20
maxDepth	Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node	2, 5	5

Table 34 Properties-based + TF-IDF + Gradient Boosted Trees best parameters

Parameter	Explanation	Values Tested	Best Performing
maxDepth	Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node	2, 5	2
maxIter	maximum number of iterations (>= 0)	5, 20	20
stepSize	Step size (a.k.a. learning rate) in interval (0, 1] for shrinking the contribution of each estimator.	0.01, 0.1	0.1

Table 35 Properties-based + TF-IDF + Linear Support Vector Machines best parameters

Parameter	Explanation	Values Tested	Best Performing
threshold	threshold in binary classification prediction applied to rawPrediction	0.0, 0.05	0.05

Table 36 Properties-based + TF-IDF + Naive Bayes best parameters

Parameter	Explanation	Values Tested	Best Performing
smoothing	The smoothing parameter	0.2, 0.5, 1.0	0.2

The following figure includes the ROC Curves of every classifier for the classification using properties-based only features.

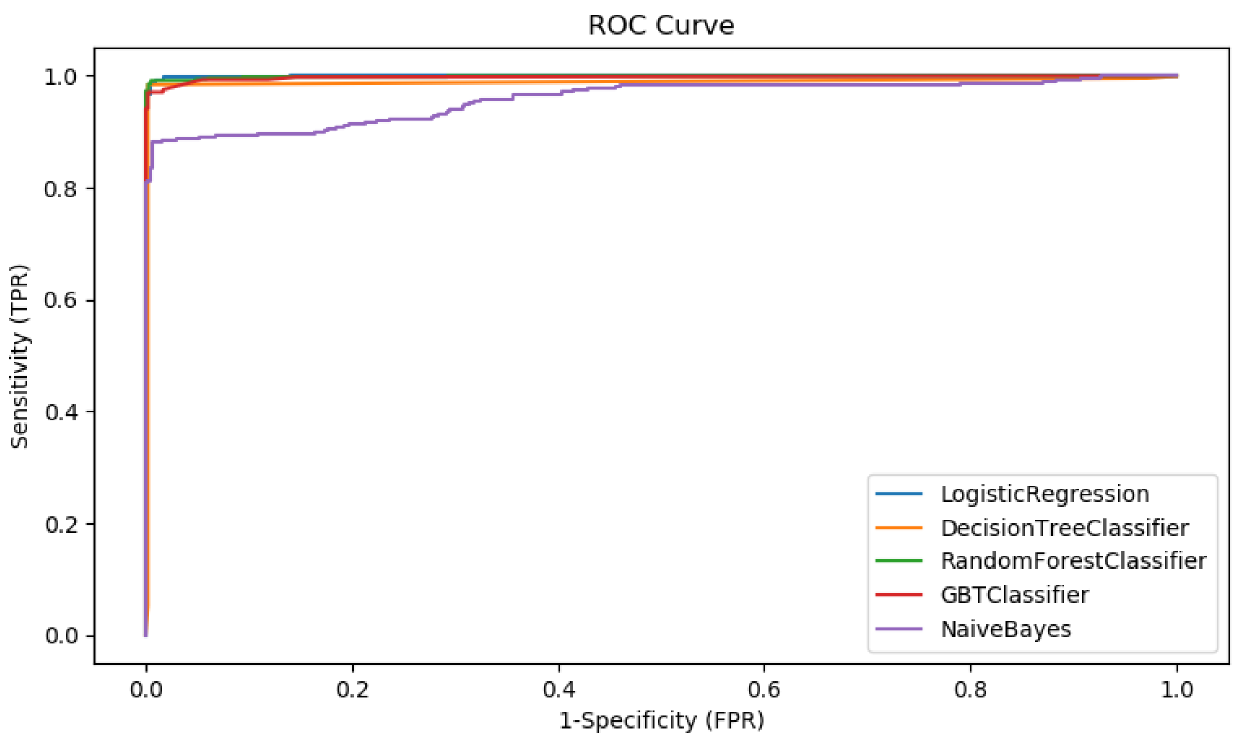


Figure 71 ROC Curves using Properties-based and TF-IDF features

It is shown here that building classifiers which use both properties-based features and TF-IDF (text-based) features produce better performing classifiers than the properties-based only and the TF-IDF only classifiers. The Naïve Bayes classifier has the worst performance, but the other classifiers, except Decision Trees, have equally good performance.

6.5 Proposed Hybrid "assembled" Architecture: Properties-Based and Word Embedding (Text-Based) Features

The performance metrics of every classifier/hyperparameter combination using the features retrieved from the properties of the email and from the Word Embedding algorithm are shown in the following table.

Table 37 Properties-based + Word2Vec Features Performance (values are in % except confusion matrix)

Algorithm	AUC	Confusion Matrix	Accuracy	Precision	Recall	F1 Measure	FPR	FNR	Time (m)
Logistic Regression	99.9	477 2 2 445	99.6	99.6	99.6	99.6	0.4	0.4	3.96
Decision Trees	97.2	425 5 7 424	98.6	98.4	98.8	98.6	1.6	1.2	5.15
Random Forest	99.9	433 3 2 456	99.4	99.6	99.3	99.5	0.4	0.6	4.95
Gradient Boosted Trees	99.8	434 6 2 457	99.1	99.6	98.7	99.1	0.4	1.3	5.2
Linear Support Vector Machine	99.9	439 3 1 400	99.5	99.8	99.3	99.5	0.2	0.7	13.08
Naïve Bayes	30.9	413 13 13 452	97.1	97.2	97.2	97.2	3.1	2.7	3.96

The parameter grid for each algorithm and the best performing hyperparameter combination are presented below.

Table 38 Properties-based + Word2Vec + Logistic Regression best parameters

Parameter	Explanation	Values Tested	Best Performing
regParam	regularization parameter (>= 0)	0.1, 0.01	0.01
elasticNetParam	the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty	0.0, 0.5, 1.0	0.0

Table 39 Properties-based + Word2Vec + Decision Trees best parameters

Parameter	Explanation	Values Tested	Best Performing
maxDepth	Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes.	2, 5, 10	10
minInfoGain	Minimum information gain for a split to be considered at a tree node.	0.0, 0.1	0.0
maxBins	Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.	6, 32	32

Table 40 Properties-based + Word2Vec + Random Forest best parameters

Parameter	Explanation	Values Tested	Best Performing
numTrees	Number of trees to train (>= 1)	5, 20	20
maxDepth	Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node	2, 5	5

Table 41 Properties-based + Word2Vec + Gradient Boosted Trees best parameters

Parameter	Explanation	Values Tested	Best Performing
maxDepth	Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node	2, 5	2
maxIter	maximum number of iterations (>= 0)	5, 20	20
stepSize	Step size (a.k.a. learning rate) in interval (0, 1] for shrinking the contribution of each estimator.	0.01, 0.1	0.1

Table 42 Properties-based + Word2Vec + Linear Support Vector Machines best parameters

Parameter	Explanation	Values Tested	Best Performing
threshold	threshold in binary classification prediction applied to rawPrediction	0.0, 0.05	0.0

Table 43 Properties-based + Word2Vec + Naive Bayes best parameters

Parameter	Explanation	Values Tested	Best Performing
smoothing	The smoothing parameter	0.2, 0.5, 1.0	0.2

The following figure includes the ROC Curves of every classifier for the classification using properties-based only features.

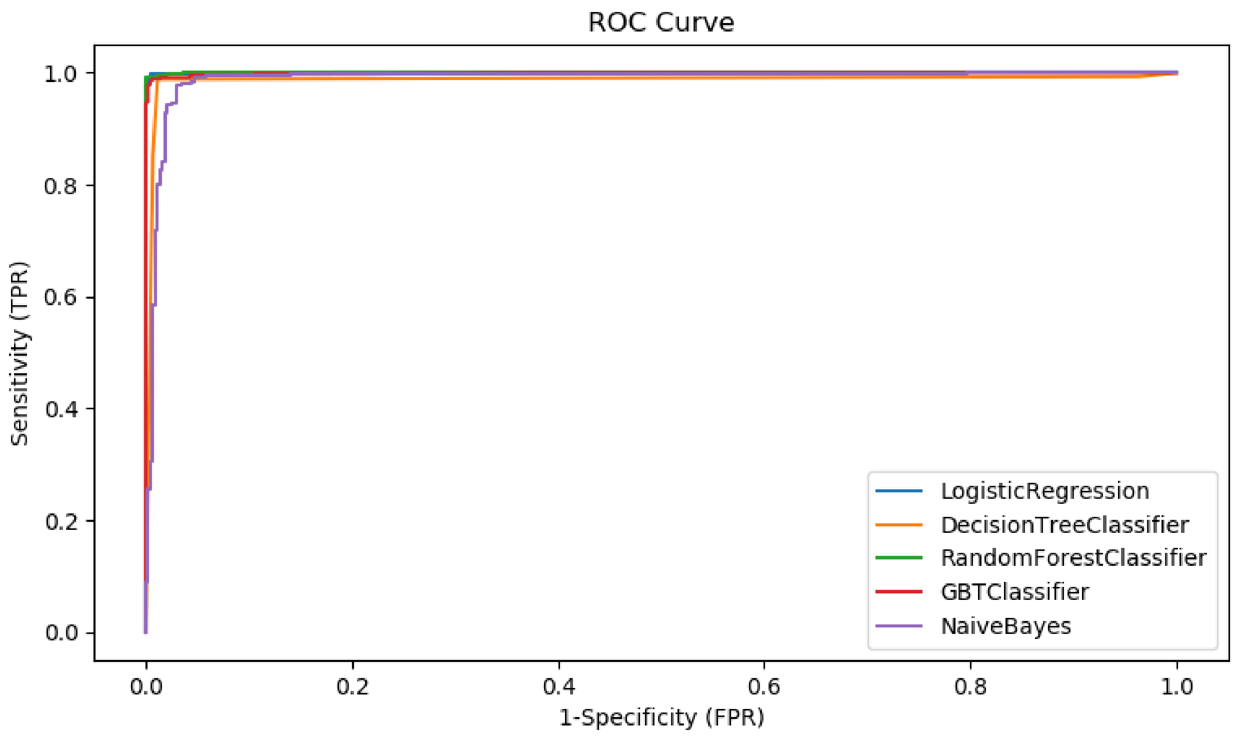


Figure 72 ROC Curves using Properties-based and Word2Vec features

From this figure it is clear that this Hybrid assembled architecture (Word Embedding features plus Properties-based features) produces the best performing classifiers and should be used as the best solution of the proposed architecture. Even the Naïve Bayes classifier, which did not perform well in the previous cases, performs well enough. The other classifiers perform equally well. Note here that when considering the Linear Support Vector Machines, the double processing time which was required should be taken into account, too. Taking the accuracy for every classifier from each feature input case, the resulting chart is the following.

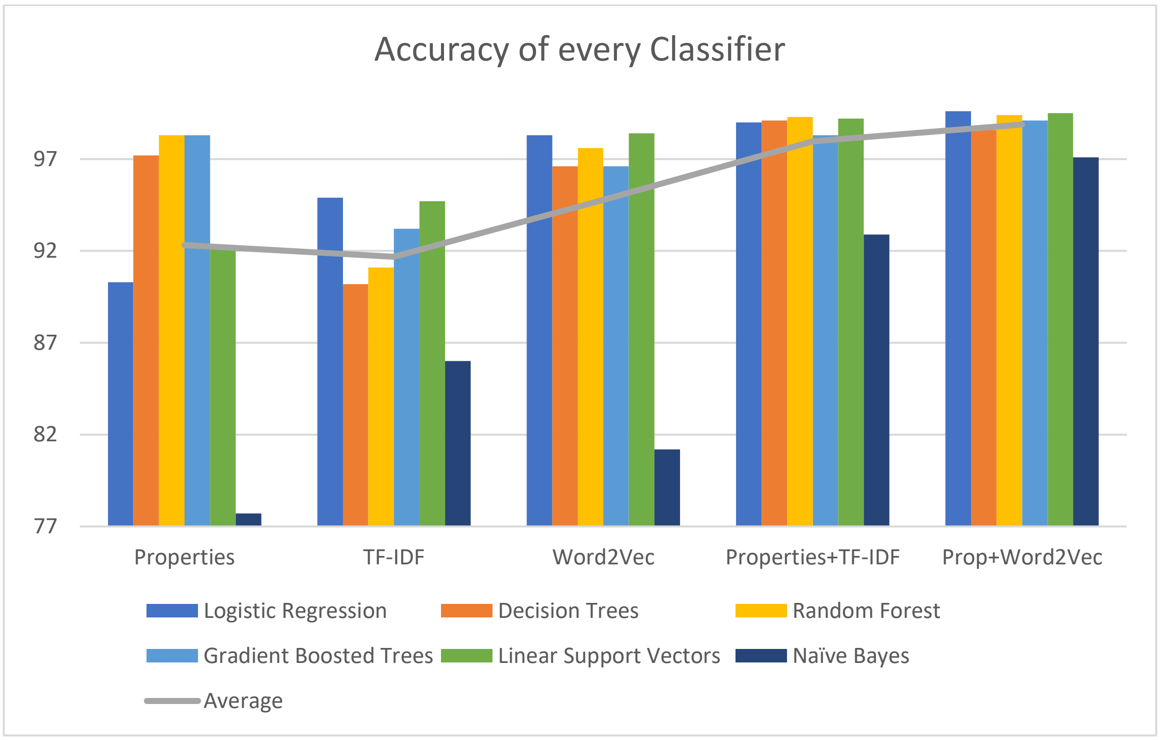


Figure 73 Chart: Accuracy of every Classifier

It can be seen from the chart that the best performing feature input is the hybrid Properties with Word Embedding features.

6.6 Hybrid Stacked Model

Here the performance results of the aforementioned hybrid stacked model are presented. It is tested whether there is a performance improvement, so the classifiers which were tested are:

A. Logistic Regression using **word embedding** features
B. Logistic Regression using **properties-based** features
C. Logistic Regression using hybrid **stacked** features: default **properties-based** features and the predictions of an already trained logistic regression classifier using **word embedding** features.
D. Logistic Regression using hybrid **stacked** features: default **properties-based** features and the predictions of an already train logistic regression classifier using again the default **properties-based** features.

Note here that, although the model A and B have already been tested, they are retested here for consistency reason and because Spark operates differently depending on specific processes. For example, because the created DataFrame in these tests is larger (about 10,000 examples in contrast to the previous tests where the size were about 4,500 examples), because it will be split in two halves, the splitting is done randomly so that process alone requires additional execution time which will change the time of the algorithms.

Table 44 Performance metrics of hybrid stacked models

Model	AUC	Confusion Matrix	Accuracy	Precision	Recall	F1 Measure	FPR	FNR	Time (m)
A	99.7	502 22 14 495	96.5	97.2	95.7	96.5	2.7	4.3	7.96
B	95.3	471 83 36 431	88.3	92.3	83.9	87.9	7.1	16.1	6.17
C	99.4	468 28 20 509	95.3	96.2	94.8	95.5	4.1	5.2	15.29
D	94.8	433 87 37 432	87.5	92.1	83.2	87.4	7.9	16.8	14.44

Table 45 Best performing hyperparameters for hybrid stacked model

Model	regParam values	Best regParam	elasticNetParam values	Best elasticNetParam
A	0.1, 0.01	0.01	0.0, 0.5, 1.0	0.0
B	0.1, 0.01	0.01	0.0, 0.5, 1.0	0.0
C	0.1, 0.01	0.01	0.0, 0.5, 1.0	0.0
D	0.1, 0.01	0.01	0.0, 0.5, 1.0	0.0

The following figures include the ROC Curves of models B, D and A, C correspondingly.

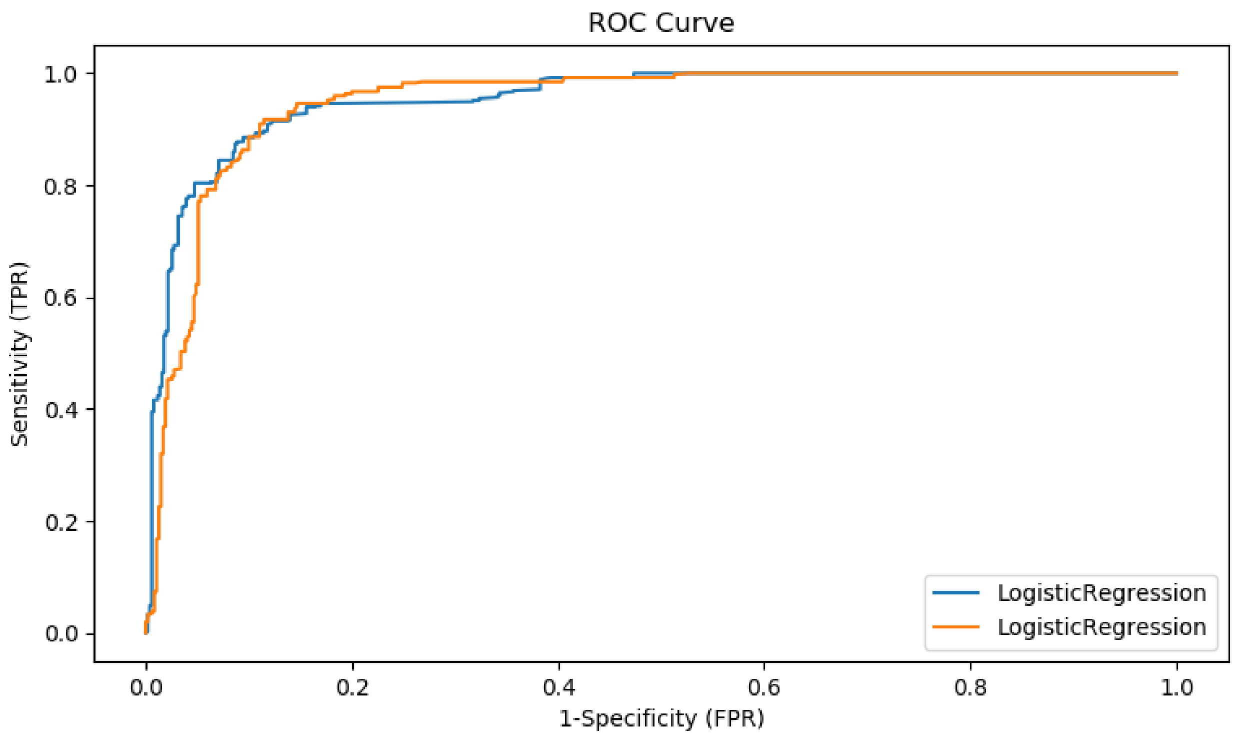


Figure 74 ROC Curves of models B and D

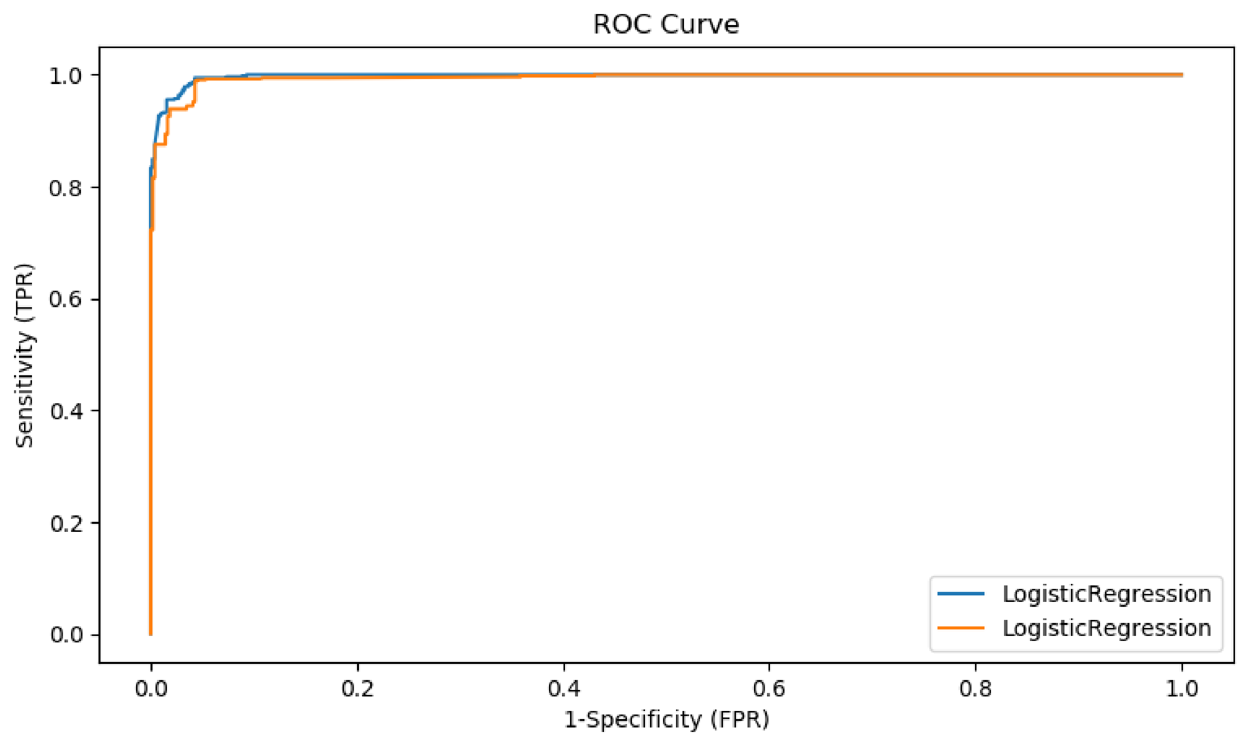


Figure 75 ROC Curves of models A and C

In both cases, (hybrid model of properties-based plus word embedding-based prediction used as feature and a hybrid model of properties-based plus another properties-based prediction used as feature), the performance actually decreases slightly. This can be seen when comparing the cases A -> C and B -> D and the ROC curves figures, where the blue line is the base classifier and the yellow line is the hybrid stacked classifier.

In conclusion, the proposed hybrid *assembled* architecture does increase the performance when compared to the simple classifiers which use only the text or only the properties of the email as features. As mentioned before, this architecture combines the properties-based and the text-based features into one consistent feature vector and uses that to train the classifiers. The proposed hybrid *stacked* model does not provide a significant performance increase, but rather has a reduced performance when compared to a simple classifier which uses only one type of features.

7 Future Work

In this chapter a few ideas for future work are presented, which should provide improvements to the detection of phishing emails.

7.1 Data Sets

First and foremost, as it became apparent in the chapter with the numerical results, there was some overfitting happening. That was because of the data sets, which were "easy" for the created machine learning models to classify. That was mostly the case for the legitimate emails.

Therefore, it would be optimal to build more data sets which include more legitimate and phishing emails, and as many different cases as possible in order to make the data sets more "difficult" for a classifier to predict and, as a natural result, make the classifiers more generalized and able to actually learn the pattern that is to be detected. This will result in slightly worse performance metrics, but the trained classifier will be able to perform in new emails and data sets with similar performance, while the previously created classifiers will most likely see a slight drop in performance when classifying new emails.

7.2 Online Training Classifier

There was a proposed architecture of an online training classifier in [82]. A similar model architecture is proposed as a future work idea. The architecture is the following figure.

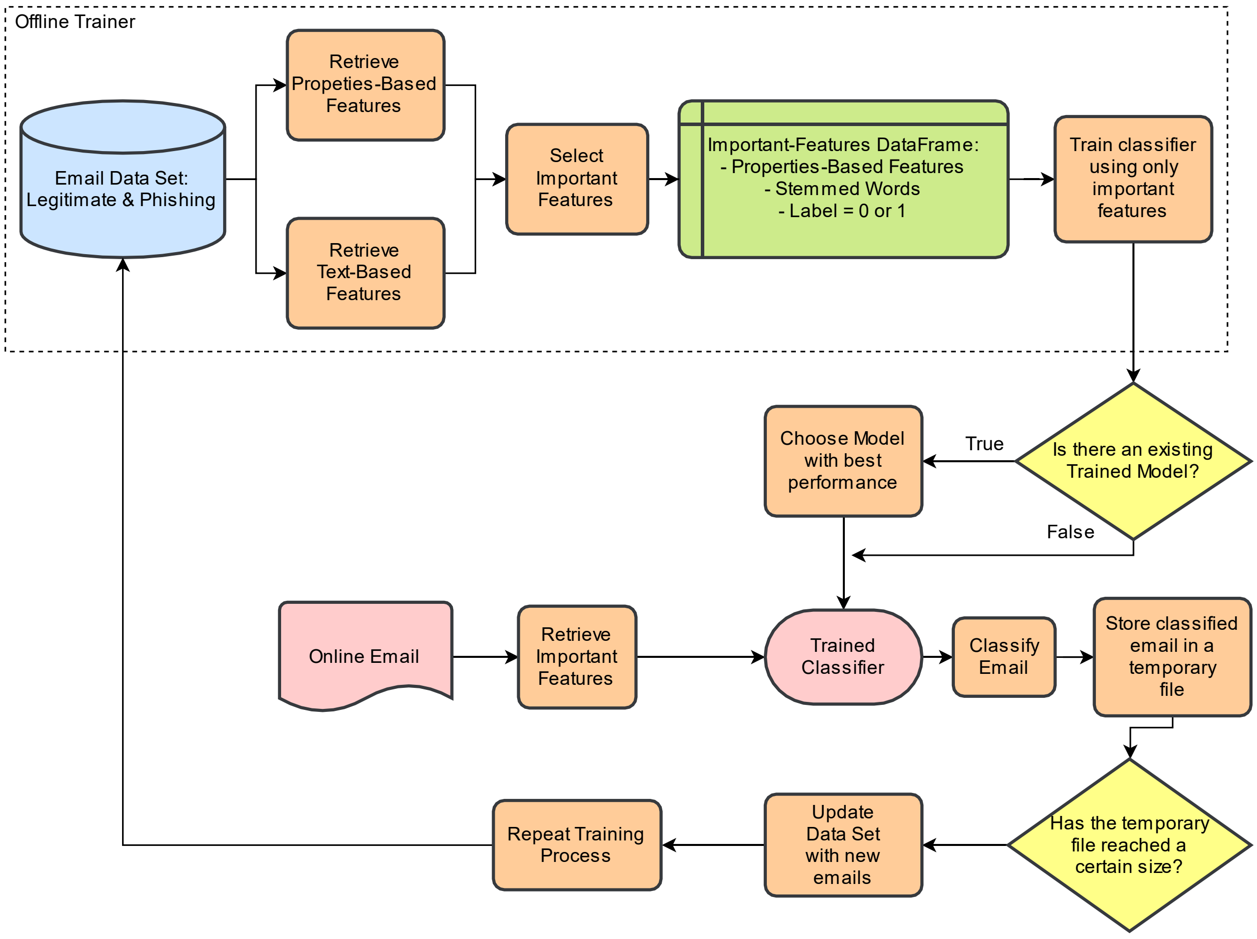

Offline Trainer
Email Data Set: Legitimate & Phishing
Retrieve Propeties-Based Features
Retrieve Text-Based Features
Select Important Features
Important-Features DataFrame: - Properties-Based Features - Stemmed Words - Label = 0 or 1
Train classifier using only important features
Is there an existing Trained Model?
True
False
Choose Model with best performance
Online Email
Retrieve Important Features
Trained Classifier
Classify Email
Store classified email in a temporary file
Has the temporary file reached a certain size?
Update Data Set with new emails
Repeat Training Process

Figure 76 Online Training Classifier

The process is the following:

1. The offline trainer process is already explained:
   a. The properties-based and the text-based features are retrieved from the emails.
   b. The important features are selected, and a DataFrame is created.
   c. A classifier is trained using the important features
2. If there is a Trained Model already, then choose the model with the best performance (highest performance metrics). If not, use the just trained classifier.
3. The additional online process is the following: Online emails are coming from a source.
4. Their important features (which were determined by the offline training process) are retrieved.
5. Each online email is classified and stored in a temporary file.
6. If the temporary file has reached a certain size, for example 10% of the original data set, then update the original data set by appending the temporary file to it.
7. Repeat the training process by going to step 1.

8 Conclusion

In this work there was an attempt to use machine learning techniques with the purpose of classifying emails as phishing or legitimate with a low error rate.

In order to properly develop the machine learning classifier, firstly a theoretical background was provided, including theoretical information and relevant work about machine learning classification techniques. Afterwards the approach of tackling the problem was provided, with machine learning tools and the components of the email as a file being explored. Lastly, the implementation algorithm and the numerical results of the performance metrics were presented.

Even though, as mentioned in the chapter of relevant work, there have already been research works for detecting phishing emails using machine learning methodologies, in this work it was attempted to build a hybrid machine learning model.

With the hybrid machine learning model, it was attempted to use two sources of inputs as features. These two are:

- The **properties** of the email, which include characteristics of the email, such as the content transfer encoding of the email, the number of URLs or the number of attachments in the email.
- The **text** of the email: This includes the text of the email that is to be read by the receiver of the email. In order to transform the text into features or, in other words, retrieve features from the email text, established algorithms were used: TF-IDF and Word Embedding.

Two different architectures were proposed and tested whether they improve the performance of phishing email classification. These architectures are:

- The hybrid **assembled** classifier: In this case the features which were retrieved from the properties and the text were assembled into one consistent feature vector and used as input to train classifiers. After testing it was proved that this classifier does improve the performance of classification when compared to the simpler classifiers which use only one type of feature as input for training.
- The hybrid **stacked** classifier: This case includes two classifiers. The first one is trained using text-based features only as inputs. The second one uses as inputs to train properties-based features plus an additional feature which is the prediction of the already trained text-based classifier for the to-be-classified email. This hybrid classifier was proved to not improved the performance but actually slightly reduce the classification performance.

In conclusion, this work includes the information needed for building machine learning models, relevant work about machine learning classification of phishing emails and a proposed architecture for machine learning classification of phishing emails which was proved to improve the performance of classification.

9 References

[1] "Financial Cryptography," [Online]. Available: https://financialcryptography.com/mt/archives/000609.html. [Accessed 2019].

[2] "APWG Phishing Activity Trends Reports," [Online]. Available: http://www.antiphishing.org/trendsreports/. [Accessed 2019].

[3] "Phishing Techniques," [Online]. Available: https://www.phishing.org/phishing-techniques. [Accessed 2019].

[4] C. a. O. J. a. K. E. Drake, "Anatomy of a Phishing Email," 2004.

[5] TinyURL. [Online]. Available: https://tinyurl.com/. [Accessed 2019].

[6] I. C. Q. P. David Ellis VP, "securitymetrics," [Online]. Available: https://www.securitymetrics.com/blog/top-10-types-phishing-emails. [Accessed 2019].

[7] O. Abdul, "An updated perspective on phishing countermeasures and their effectiveness," 2015.

[8] "Vade Secure: Predictive Email Defense," [Online]. Available: https://www.vadesecure.com/en/phishing-awareness-training-8-things-employees-understand/.

[9] "Symtrex, User Education - Phishing," [Online]. Available: https://symtrex.com/security-solutions/user-education/. [Accessed 2019].

[10] "Google Safe Browsing," [Online]. Available: https://safebrowsing.google.com/. [Accessed 2019].

[11] "PhishTank: Out of the Net, into the Tank.," [Online]. Available: https://www.phishtank.com/.

[12] "Googles Developers Introduction to Machine Learning," [Online]. Available: https://developers.google.com/machine-learning/crash-course/ml-intro.

[13] L. a. M. M. O. a. N. S. a. K. S. a. K. J. Juárez-Orozco, "The machine learning horizon in cardiac hybrid imaging," *European Journal of Hybrid Imaging,* vol. 2, 2018.

[14] C. Catal, "Performance evaluation metrics for software fault prediction studies," *Acta Polytechnica Hungarica,* vol. 9, no. 4, pp. 193-206, 2012.

[15] "Google Developers, ROC and AUC," [Online]. Available: https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc. [Accessed 2019].

[16] F. a. K. J. a. L. C.-G. a. C. K. Akram, "Segmentation of Regions of Interest Using Active Contours with SPF Function," *Computational and Mathematical Methods in Medicine,* vol. 2015, 2015.

[17] L. a. A. I. a. N. J. Halgas, "Catching the Phish: Detecting Phishing Attacks using Recurrent Neural Networks (RNNs)," August 2019.

[18] L. a. M. M. O. a. N. S. a. K. S. a. K. J. Juárez-Orozco, "The machine learning horizon in cardiac hybrid imaging," *European Journal of Hybrid Imaging,* vol. 2, 2018.

[19] A. G. Hetal Bhavsar, "A Comparative Study of Training Algorithms for Supervised Machine Learning," *International Journal of Soft Computing and Engineering,* vol. 2, no. 4, 2012.

[20] T. SRIVASTAVA, "analytics vidhya," 26 March 2018. [Online]. Available: https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/. [Accessed 2019].

[21] S. J. P. P.SIBI, "ANALYSIS OF DIFFERENT ACTIVATION FUNCTIONS USING BACK PROPAGATION NEURAL NETWORKS," *Journal of Theoretical and Applied Information Technology,* vol. 47, no. 3, pp. 1264-1268, 2013.

[22] "Interactive Matchematics," 15 April 2018. [Online]. Available: https://www.intmath.com/laplace-transformation/1a-unit-step-functions-definition.php. [Accessed 2019].

[23] T. a. L. D. a. A.-R. S. a. K. a. A. D. Leibovich-Raveh, "A new method for calculating individual subitizing ranges," 2018.

[24] M. West, "bouvet," 20 August 2019. [Online]. Available: https://www.bouvet.no/bouvet-deler/explaining-recurrent-neural-networks.

[25] A. a. A. A. Akinyelu, "Classification of Phishing Email Using Random Forest Machine Learning Technique," *Journal of Applied Mathematics,* 2014.

[26] H. Singh, "Towards Data Science," 3 November 2018. [Online]. Available: https://towardsdatascience.com/understanding-gradient-boosting-machines-9be756fe76ab. [Accessed 2019].

[27] B. a. R. K. H. Adhi Tama, "An extensive empirical evaluation of classifier ensembles for intrusion detection task," *Computer Systems Science and Engineering,* vol. 32, pp. 149-158, 2017.

[28] F. a. G. A. a. G.-V. F. a. G. T. M. a. T. J. Divina, "Stacking Ensemble Learning for Short-Term Electricity Consumption Forecasting," *Energies,* vol. 11, p. 949, 2018.

[29] S. A. S. S. G. S. S. H. R. K. S. RISHIKESH B H, "Analysis and detection of e-mail phishing using PySpark," *International Research Journal of Engineering and Technology (IRJET),* vol. 5, no. 4, pp. 984-986, 2018.

[30] "Virus Total," [Online]. Available: https://www.virustotal.com/. [Accessed 2019].

[31] S. a. N. D. a. W. X. a. N. S. Abu-Nimeh, "A comparison of machine learning techniques for phishing detection," *ACM International Conference Proceeding Series,* vol. 269, pp. 60-69, 2007.

[32] N. A. U. V. R. S. K. Hiransha M, "Deep Learning Based Phishing E-mail Detection".

[33] R. a. Q. U. Hussain, "An Approach to Detect Spam Emails by Using Majority Voting," 2014.

[34] E. R. G. F. J. S. Mark Hopkins, "UCI Machine Learning Repository, Spambase Data Set," [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Spambase. [Accessed 2019].

[35] "rapidminer, Lightning Fast Data Science Platform for Teams," [Online]. Available: https://rapidminer.com/. [Accessed 2019].

[36] N. a. C. B. a. B. W. Moradpoor, "Employing Machine Learning Techniques for Detection and Classification of Phishing Emails," in *Computing Conference*, London, 2017.

[37] "Spam Assassin spam email public corpus," [Online]. Available: https://spamassassin.apache.org/old/publiccorpus/. [Accessed 2019].

[38] J. Navario, "monkey," [Online]. Available: https://monkey.org/~jose/phishing/.

[39] "Mathworks Matlab," [Online]. Available: https://www.mathworks.com/. [Accessed 2019].

[40] F. a. K. F. Thabtah, "Phishing Detection: A Case Analysis on Classifiers with Rules Using Machine Learning," *Journal of Information & Knowledge Management,* vol. 16, 2017.

[41] C. L. a. C. K. a. M. N. a. I. D. Tan, "Identifying the Most Effective Feature Category in Machine Learning-based Phishing Website Detection," pp. 1-6, 2016.

[42] M. a. K. S. Zareapoor, "Feature Extraction or Feature Selection for Text Classification: A Case Study on Phishing Email Detection," *International Journal of Information Engineering and Electronic Business,* vol. 7, pp. 60-65, 2015.

[43] G. a. T. J. Park, "Using Syntactic Features for Phishing Detection," 2015.

[44] A. a. V. R. Das, "Automated email Generation for Targeted Attacks using Natural Language," 2019.

[45] R. a. S. N. a. H. N. Verma, "Detecting Phishing Emails the Natural Language Way," pp. 824-841, 2012.

[46] A. a. C. J. H. a. P. G. a. R. F. a. S. S. Bergholz, "Improved Phishing Detection using Model-Based Features," 2008.

[47] D. a. G. B. B. a. A. S. a. M. A. a. A. E. Almomani, "A Survey of Phishing Email Filtering Techniques," *IEEE Communications Surveys &amp Tutorials,* vol. 15, pp. 2070-2090, 2013.

[48] A. a. C. J. H. a. P. G. a. R. F. a. S. S. Bergholz, "Improved Phishing Detection using Model-Based Features," in *Conference on Email and Anti-Spam (CEAS)*, California, 2008.

[49] L. a. O. B. a. W. P. a. B. S. Ma, "Detecting Phishing Emails Using Hybrid Features," pp. 493-497, 2009.

[50] e. a. M. del Castillo, "An Integrated Approach to Filtering Phishing Emails," *Computer Aided Systems Theory EUROCAST 2007,* vol. 4739, pp. 321-328, 2007.

[51] R. I. a. J. Abawajy, "A Multi-tier Phishing Detection and Filtering Approach," *Journal of Network and Computer Applications,2012,* 2012.

[52] e. a. A. Almomani, "Evolving Fuzzy Neural Network for Phishing Emails Detection," *Journal of Computer Science,* vol. 8, no. 7, pp. 1099-1107, 2012.

[53] N. Kasabov, "Evolving fuzzy neural networks-algorithms, applications and biological motivation," *Methodologies for the conception, design and application of soft computing, World Scientific,* pp. 271-274, 1998.

[54] e. a. A. ALmomani, "An enhanced online phishing e-mail detection framework based on evolving connectionist system," *International Journal of Innovative Computing, Information and Control (IJICIC),* vol. 9, no. 3, 2013.

[55] e. a. A. ALmomani, "Phishing Dynamic Evolving Neural Fuzzy Framework for Online Detection Zero-day Phishing Email," *Indian Journal of Science and technology,* vol. 6, no. 1, 2013.

[56] E. P. Resnick, "RFC 5322 - Internet Message Format," October 2008. [Online]. Available: https://tools.ietf.org/html/rfc5322. [Accessed 2019].

[57] T. Makin, "Multipart MIME Email Guide," [Online]. Available: https://gist.github.com/tylermakin/d820f65eb3c9dd98d58721c7fb1939a8. [Accessed 2019].

[58] G. K. a. J. Palme, "RFC 4021 - Registration of Mail and MIME Header Fields," March 2005. [Online]. Available: https://tools.ietf.org/html/rfc4021. [Accessed 2019].

[59] "Fileformat - Email File Formats," [Online]. Available: https://wiki.fileformat.com/email. [Accessed 2019].

[60] "Spam Assassin Public Mail Corpus," [Online]. Available: https://spamassassin.apache.org/old/publiccorpus/. [Accessed 2019].

[61] "Apache SpamAssassin," [Online]. Available: https://spamassassin.apache.org/. [Accessed 2019].

[62] "Jose Navario phishing corpus," [Online]. Available: https://monkey.org/~jose/phishing/. [Accessed 2019].

[63] W. W. Cohen, "Enron Email Dataset," 8 May 2015. [Online]. Available: https://www.cs.cmu.edu/~enron/. [Accessed 2019].

[64] "Enron Corporation - Company Profile," [Online]. Available: https://www.referenceforbusiness.com/history2/57/Enron-Corporation.html. [Accessed 2019].

[65] "github diego ocampo machine learning phishing," [Online]. Available: https://github.com/diegoocampoh/MachineLearningPhishing. [Accessed 2019].

[66] "UCI Machine Learning Repository - Spambase Data Set," [Online]. Available: https://archive.ics.uci.edu/ml/datasets/spambase.

[67] M. A. H. a. I. H. W. Eibe Frank, The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques", Fourth Edition, Morgan Kaufmann, 2016.

[68] "Github python 3 weka wrapper," [Online]. Available: https://github.com/fracpete/python-weka-wrapper3. [Accessed 2019].

[69] K. Hornik, "RWeka R Documentation," [Online]. Available: https://www.rdocumentation.org/packages/RWeka/versions/0.4-41. [Accessed 2019].

[70] M. Hall, "Mark Hall on Data Mining & Weka," 4 March 2015. [Online]. Available: https://markahall.blogspot.com/2015/03/weka-and-spark.html. [Accessed 2019].

[71] "TensorFlow," [Online]. Available: https://www.tensorflow.org/. [Accessed 2019].

[72] "Apache Spark," [Online]. Available: https://spark.apache.org/. [Accessed 2019].

[73] "Cluster Mode Overview Apache Spark," [Online]. Available: https://spark.apache.org/docs/latest/cluster-overview.html. [Accessed 2019].

[74] R. S. o. Mesos. [Online]. Available: https://spark.apache.org/docs/latest/running-on-mesos.html.

[75] "Running Spark on YARN," [Online]. Available: Running Spark on YARN.

[76] "Apache Hadoop," [Online]. Available: https://hadoop.apache.org/docs/stable/.

[77] "Running Spark on Kubernetes," [Online]. Available: https://spark.apache.org/docs/latest/running-on-kubernetes.html. [Accessed 2019].

[78] K. Koutroumpouchos, "github: Spark Hybrid ML Phishing Emails," 2019. [Online]. Available: https://github.com/KostasKoutrou/Spark-Hybrid-ML-Phishing-Emails.

[79] "Anaconda," [Online]. Available: https://www.anaconda.com/. [Accessed 2019].

[80] "Spyder," [Online]. Available: https://www.spyder-ide.org/. [Accessed 2019].

[81] "Beautiful Soup 4.4.0 Documentation," [Online]. Available: https://www.crummy.com/software/BeautifulSoup/bs4/doc/. [Accessed 2019].

[82] S. Smadi, Detection of online phishing email using dynamic evolving neural network based on reinforcement learning. Doctoral thesis, Northumbria University., 2017.

[83] G. Ziccardi, "Opening Remarks: Hacking and Digital Dissidence," 2013, pp. 1-25.

[84] L. a. A. I. a. N. J. Halgas, "Catching the Phish: Detecting Phishing Attacks using Recurrent Neural Networks (RNNs)," 2019.

[85] B. A. Pashel, "Teaching Students to Hack: Ethical Implications in Teaching Students to Hack at the University Level," *University, Kennesaw State.*

[86] Thomas, Georg, Charles Sturt University, School of Computing and Mathematics; Burmeister, Oliver, Charles Sturt University, School of Computing and Mathematics; Low, Gregory, SQL Down Under, "Issues of Implied Trust in Ethical Hacking," *Orbit,* 2018.

[87] Jamil, Danish, Department of Computer Engineering, Sir Syed University of Engineering & Technology; Numan Ali Khan, Muhammad,, *Is Ethical Hacking Ethical?,* 2011.

[88] Himma, Kenneth Einar, Seattle Pacific University, Internet Security: Hacking, Counterhacking, and Society, Massachusetts: Jones and Bartlett publishers, 2007.

[89] Smith, Bryan; Yurcik, William; Doss, David, "Ethical Hacking: The Security Justification Redux," Illinois State University, 2002, pp. 374-380.

[90] A. Farsole, Ajinkya; G. Kashikar, Amruta; Zunzunwala, Apurva, "Ethical Hacking," *International Journal of Computer Applications,* pp. 14-20, 2010.