

École polytechnique de Louvain

On the Transferability of Adversarial Examples in Machine Learning: the Case of Email Spam Detection

Author: **Simon JACQUET**

Supervisors: **Michel VERLEYSEN, François-Xavier STANDAERT**

Reader: **Olivier BRONCHAIN**

Academic year 2020–2021

Master [120] in Data Sciences Engineering

Abstract

Nowadays, machine learning techniques are used widely. As they are more and more employed, the question of safety comes into play: are machine learning techniques robust to attacks from malicious users? It has been shown that they are not. In particular, a number of articles have shown that even state-of-the-art machine learning models are weak to adversarial examples attacks. So-called adversarial examples are inputs modified as to be misclassified by a model. In image classification, these adversarial examples have also been shown to transfer across models: adversarial examples crafted by attacking a model A often fools another model B. This phenomenon called transferability has been less studied in the case of text classification. In this thesis, we study the transferability of adversarial examples for email spam detection. For this purpose, we choose six models: (i) multinomial naive Bayes classifier, (ii) logistic regression, (iii) linear support vector machine, (iv) random forest, (v) k-nearest neighbors and (vi) convolution neural networks. We also chose to focus on two attacks: (i) text fooler, a black-box attack, and (ii) hot-flip, a white-box attack. As our main result, we evidence that, for our two chosen attacks, adversarial examples do not transfer when one takes into account natural constraints on the transformations that an email can undergo.

Keywords: Machine Learning, Spam Detection, Adversarial Examples, Transferability, CNN, SVM, KNN, Multinomial Naive Bayes, Random Forest

Acknowledgments

I would first like to thank my supervisors Michel Verleysen and François-Xavier Standaert. Without their feedback, this thesis would not have been possible. I also thank Olivier Bronchain who accepted to be a reader for this thesis.

I wish to express my gratitude to my family for their support during my thesis. Their advice helped me tremendously and they proved exceptional in their moral support all throughout my thesis.

Contents

1	Introduction	1
1.1	Background	1
1.2	Scope	2
1.3	Outline	3
2	Machine learning background	5
2.1	Machine Learning	6
2.1.1	Learning	6
2.1.2	Supervised learning	7
2.1.3	Unsupervised learning	9
2.1.4	Reinforcement learning	9
2.2	Adversarial Machine Learning	11
2.2.1	Dimensions of attacks	11
2.2.2	Main types of attacks	15
2.3	Learning algorithms	19
2.3.1	Multinomial naive Bayes classifier	20
2.3.2	Logistic regression	21

2.3.3	Random forest	22
2.3.4	K-nearest neighbors	23
2.3.5	Linear support vector machine	24
2.3.6	Convolution neural network	26
2.4	Conclusion	30
3	Text classification background	32
3.1	Preprocessing	32
3.1.1	Noise removal	33
3.1.2	Normalization	33
3.1.3	Tokenization	34
3.1.4	Feature representation	34
3.2	Learning algorithms	36
3.2.1	Bag of word approach	38
3.2.2	Word embedding approach	38
3.2.3	Convolution neural network for text classification	38
3.3	Conclusion	40
4	Adversarial examples background	41
4.1	Text attack	41
4.1.1	Goal function	42
4.1.2	Constraints	42
4.1.3	Transformation	44
4.1.4	Search method	44

4.2	Computing gradient	45
4.2.1	Multinomial Bayes classifier [2.3.1]	45
4.2.2	Linear support vector machine [2.3.5]	47
4.2.3	Logistic regression [2.3.2]	47
4.2.4	Convolution neural network [2.3.6]	48
4.3	Attacks	51
4.3.1	Text fooler	51
4.3.2	Hot-flip	52
4.4	Conclusion	52
5	Spam detection	54
5.1	Spams	55
5.2	Spam detection in practice	55
5.2.1	Data collection	56
5.2.2	Data preprocessing	57
5.2.3	Data exploration and visualisation	58
5.2.4	Model building	59
5.2.5	Model evaluation	64
5.3	Conclusion	66
6	Adversarial examples	67
6.1	Methodology	68
6.1.1	Constraints imposed	68
6.1.2	Implementing the attacks	70

6.1.3	Experiment details	71
6.2	Results	72
6.2.1	Building adversarial examples	72
6.2.2	Composition of adversarial examples	73
6.2.3	Transferability of adversarial examples	73
6.3	Discussion	75
6.4	Conclusion	77
7	Conclusion	79
A	Preprocessing	81
B	Cross-validation results	86

Chapter 1

Introduction

1.1 Background

Machine learning is an old field of study. Arthur Samuel, pioneer in the field of artificial intelligence, already worked on the subject in the 50s [20]. As early as 1958, Frank Rosenblatt released an article introducing the perceptron, key element in nowadays' neural networks [28]. Following this article, Frank Rosenblatt declared that in the future, neural nets would be capable of viewing images, beating humans at chess or even reproduce. In 1969 however, Marvin Minsky and Seymour Papert published a book proving that perceptrons could only be trained to solve linear separable problems, and consequently could not model the XOR function, but more importantly raised doubts on multi-layered nets of perceptrons by stating that there were no reasons “why the virtues of perceptrons would carry over the many-layered version” [23]. That has severely broke the hype around machine learning. On an historical note, one has to confess that, at that time, the technique of backpropagation was not known.

In recent years, machine learning has gained a renewed interest. This interest certainly builds upon scientific discoveries deep neural networks [16] but is mostly explained by two factors. On the one hand, data is easily available everywhere. An american media, the International Data Group predicted the global data volume to grow exponentially from 4.4 zettabytes to 44 zettabytes between 2013 and 2020. IDC forecasted 163 zettabytes of data by 2025 [27]. On the other hand, current computers are very powerful, much more than before. Moore's law predicts that the computational power of computers doubles every 1.5 to 2 years. Hence, calculations

requiring a day's worth of computation today wouldn't have been possible twenty years ago.

As a result, machine learning techniques are nowadays used widely. One can find them in domains such as online advertising, algorithmic trading, health, fraud detection, computer vision, machine translation and natural language understanding. With the ever-growing deployment of machine learning, the question of its safety comes to mind: are machine learning techniques robust to attacks from malicious users? This is of particular interest for applications related to the domain of cyber-security which include anti-viruses, firewalls, malwares and spams. In fact, in those applications, machine learning techniques help to protect against malicious behaviors. They are a defense against an attack and the security of machine learning is of paramount importance.

1.2 Scope

From a security point of view, applications are typically depicted around two kinds of persons: attackers and defenders. Defenders deploy machine learning models which are then the target of attacks. Hence by design, security contains an adversarial dimension. The main idea behind so-called adversarial machine learning is in fact to produce faulty results by manipulating trained learning models.

Adversarial machine learning attacks can take many forms. Just in email spam detection, there are at least three vectors of attack. The first one is to craft a spam that bypasses the detection of machine learning models. This form of attack is called evasion attack. In the second type of attack, attackers poison the database on which the model is learned, skewing the resulting classifier. This is a form of data poisoning. In the third category of attack, attackers query the spam classifier and use the result to build their own machine learning model. We call this model stealing.

This thesis focuses on evasion attacks for email spam detection. We study how small modifications of a spam can result in it bypassing the detection of machine learning models. These modified spams are referred to as adversarial examples, from which originate the title of this thesis. More generally, adversarial examples are inputs modified by an adversary, another name for attacker, to be misclassified by a model.

Adversarial examples are known to fool even state of the art models in the image

classification setting [29]. They have also been found to transfer across models – adversarial examples fooling one model is likely to fool the other one, even for models of different structures [26]. For text based classification, deep neural networks have been shown to be weak against adversarial examples [17]. Some interest has been given to their transferability [32, 35]. However, to the author’s best knowledge, there has been no interest in studying the transferability of adversarial examples from deep neural networks that uses word embedding to models with other types of input representation such as bag of words or tf-idf.

In this context, this thesis raises the following question: in the case of text based classification problems, such as email spams, are adversarial examples transferable from neural networks structures with word embedding to more classical machine learning models with bag of words input representation?

1.3 Outline

In order to answer this question, we proceed by identifying two auxiliary sub-questions. First, can we build machine learning models to detect email spams from our dataset? As explained subsequently in this thesis, the answer is given by building convolution neural networks to represent the word embedding models. With the bag of words representation, we choose five architectures: (i) multinomial naive Bayes classifier, (ii) logistic regression, (iii) linear support vector machine, (iv) random forest and (v) k-nearest neighbors. We chose those five architectures because they were very different from one another. As we will see, except for logistic regression, they turned out to perform well above random guess.

As a second sub-question, we ask ourselves whether we can craft adversarial examples to fool our models. To craft adversarial examples, we consider two methods from the literature: (i) the text fooler attack [10] and (ii) the hot flip attack [6]. We select those two methods because they come from well known papers and are very different. In fact, the first is a black-box attack while the second is a white-box attack. As shown later in the thesis, we observe that the text fooler attack manage to fool well our models, with the exception of our k-nearest neighbors model. We could, however, attribute this result to the choice of the hyperparameter for the model. In contrast our hot flip attack does not fool very well our models. We attribute this result to the fact that gradients used by the model do not take into account particular structures and, as such, perform poorly.

Having answered the two sub-questions, we will see that we are in a position to

answer our main question: do our adversarial examples transfer across models? We observe that our adversarial examples do transfer but not to the extent of the literature [32, 35]. As we shall see, we justify this result by two factors: (i) on the one hand, those articles use the n-gram representation of texts while we use word tokenization and (ii) on the other hand, we make use of particular structures.

In order to present our work according to the above lines, the rest of the document is organized as follows.

The first three chapters provide the reader with the background material necessary to understand this thesis. Chapter 2 provides the basic knowledge on machine learning and establishes the context in which are inscribed adversarial examples. Chapter 3 delves into text classification since spam detection is but a special case of text classification. Chapter 4 gives fundamental knowledge on adversarial examples in the case of text classification.

In the next two chapters, we construct experiments aimed at answering the question of transferability. This is done in two chapters.

Chapter 5 focuses on building machine learning models to classify email spams, using both the Enron email archive and Apache spam email dataset [7]. It answers the first sub-question. Chapter 6 concentrates on attacking our now up and running machine learning models. It answers the second sub-question as well as the main questions.

Finally chapter 7 presents our conclusion and suggests future work.

Chapter 2

Machine learning background

This chapter aims at providing the reader with the background necessary to understand this thesis. Our main focus is on adversarial examples, which constitute a form of adversarial machine learning, which itself is part of machine learning.

In section 2.1, we introduce machine learning. We begin by explaining how the learning in machine learning works. We follow that by describing each of the three categories of machine learning: (i) supervised learning, (ii) unsupervised learning and (iii) reinforcement learning.

In section 2.2, we interest ourselves to the adversarial part of machine learning. We first propose a classification method for the different adversarial machine learning techniques using three dimensions of attacks: (i) attack timing, (ii) attacker information and (iii) attack goals. We then describe the main types of attacks: (i) data poisoning, (ii) evasion attacks and (iii) model stealing.

Section 2.3 presents several major learning algorithms: (i) multinomial naive Bayes classifier, (ii) logistic regression, (iii) random forest, (iv) k-nearest neighbors, (v) linear support vector machine and (vi) convolution neural network. We give the fundamental mathematics behind each of these structures. This serves to show the diversity of structures in the algorithms. This also allows us to introduce the reader to the machine learning techniques we shall use subsequently in the thesis.

We finish in section 2.4 by a brief conclusion.

2.1 Machine Learning

Machine Learning is the study of computer algorithms that improve automatically through experience [24]. Depending on the task, machine learning can be further categorized as supervised, unsupervised or reinforcement learning.

2.1.1 Learning

Machine learning generally follows the same steps as the pipeline shown in Figure 2.1.

- At first, we have raw data. In the case of spam emails, raw data consists of a collection of emails.
- Raw data is then processed. For each instance i of the raw data, we want to extract a series of numerical features put together under a single feature vector x_i . For instance, for each email instance i , given keywords $w_1, \dots, w_m$, we count the number of times n_k each keyword w_k appears, thereby constructing the vector $(n_1, \dots, n_m)$.
- That processed data is then fed to a learning algorithm which outputs a model.

A model can be determined by two aspects, namely its parameters and its hyperparameters.

Hyperparameters are parameters used to control the learning process. We distinguish two types of hyperparameters. The first one, model hyperparameters, relate to the structure of models. They restrain the family of models over which the learning algorithm can pick the best one. As a consequence, the choice of model hyperparameters affect directly the performance of the model. Examples of hyperparameters are the number of layers in a neural network, the value of k in K-nearest neighbor, or even the choice between support vector machines and random forest. The second one, algorithm hyperparameters, are choices made for the learning algorithm. Those choices have no repercussion on the performance of the model. However, they do determine the speed and quality of the learning process. To cite a few, learning rate and mini-batch size are algorithm hyperparameters.

Contrarily to hyperparameters, parameters are learned from data through training. They can be the weights or the values of nodes in the model.

2.1.2 Supervised learning

In supervised machine learning, data given to the learning algorithm is composed of two parts, inputs and their corresponding outputs. The objective is to learn the function mapping inputs to outputs. The model produced is a system that, for any given input, tries to predict its output.

In the case of email spams, the inputs are the emails and the outputs are the labels characterizing the emails as spams or hams. The model produced tries to guess whether a new email is a spam or a ham.

Depending on the structure of the output, supervised learning is further subdivided in classification and regression:

- Classification. The output is a label, a class per say. During the learning phase, the learning algorithm tries to draw boundaries delimiting each class.
- Regression. The output is a real number. During the learning phase, the learning algorithm tries to draw a function that best fits the cloud of points.

Figure 2.3 shows simple examples that allows to better differentiate classification from regression. In the classification example, we separate data points belonging to three different classes by drawing some boundaries. The boundaries are not perfect as there are one square and one star that are misclassified. Those two data points are located outside the boundaries of their classes. In the regression example, we are given data points in blue. The input of data points is given by the x axis and the output, or the value of the data points, is given by the y axis. We draw in red a function that gives an output for any input.

It is worth noting that in supervised learning, while model hyperparameters cannot be directly learned from data, one can compare models with different parameters to choose the model with the best hyperparameters. This is because, in supervised learning, models predict outputs from some input and, given a metric, one can measure the quality of the predicted outputs with respect to the true output.

A first way to do so would be to compare the models on the data from training. This method, however, is flawed as models already have seen those data. For

instance, a model repeating the output for any known input and choosing randomly a value for any other input would score perfectly on the training data. It would however be useless for any other input. Predicting accurately training data but poorly on other data is called overfitting.

To prevent overfitting, there exists a method called cross-validation. Cross-validation functions by round, one round consisting of:

1. Splitting data into a training set and a validation set, both sets being complementary;
2. Training all models on the training set;
3. Measuring the performance of those models on the validation set.

To reduce variability, most cross-validation techniques rely on multiple such rounds. The results are then combined, e.g. by averaging, to give for each model an overall score.

There exist many types of cross-validation techniques. Among those, the most popular are the following:

- Leave-one-out cross-validation. In this cross-validation technique, there are n rounds. At each round i , the training set is composed of all data except the instance i . The validation set is composed of the sole instance i .
- K-fold cross-validation. In this technique, the data is split in k sets. There are k rounds. At each round i , set i becomes the validation set. The training set is composed of the union of all other sets.
- Holdout method. In the holdout method, there is a single round. The size of the training and validation set is arbitrary, though, in practice, the size of the validation set is usually smaller than the size of the training set. The advantage of this technique is its speed. Its downside is that results vary much more than other cross-validation techniques as averaging on multiple rounds is essential to reducing variability.
- Repeated random sub-sampling validation. This technique is also called Monte Carlo cross-validation. This technique has an arbitrary number of rounds k . In each round, the data is randomly split into the training and validation sets, whose sizes are constant throughout all rounds. Contrarily to

K-fold cross-validation, this method has the advantage that the number of rounds is arbitrary and independent from the size of the validation set. Its disadvantage, however, is that, as each round is independent from each other, some data may never appear in the validation set.

2.1.3 Unsupervised learning

In unsupervised machine learning, as shown in Figure 2.4, we are not given any expected outputs or labels. The objective, thus, cannot be to predict some target label. The objective actually varies from one problem to another. Common unsupervised learning problems are:

- Clustering. The objective is to form groups and give a group to each feature vector. Groups are composed in a way that members of a same group are closer than members of other groups.
- Anomaly detection. The objective is to identify outliers amongst the data, e.g. feature vectors that differ significantly from the rest of the data. This approach can be used to identify bank fraud, medical problems, equipment defects.
- Matrix factorization. The objective is to decompose a matrix in the product of two other matrices of lower dimension. This approach is particularly useful for recommendation systems.

2.1.4 Reinforcement learning

In reinforcement learning, the system interacts with a dynamic environment and learns from this interaction. Generally, one considers a Markov decision process, composed of:

- the state space S ,
- the action space A ,
- probabilities of the form $P_a(s, s') = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$, which denotes the probability of reaching state s' when leaving state s through action a .

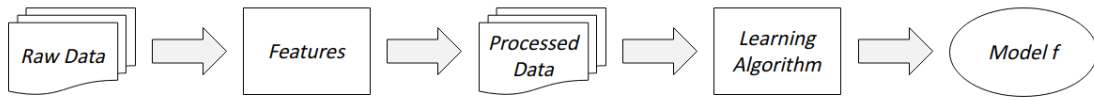


Figure 2.1: Steps in machine learning. From raw data are extracted features to give processed data which is then fed to a learning algorithm to produce a model.

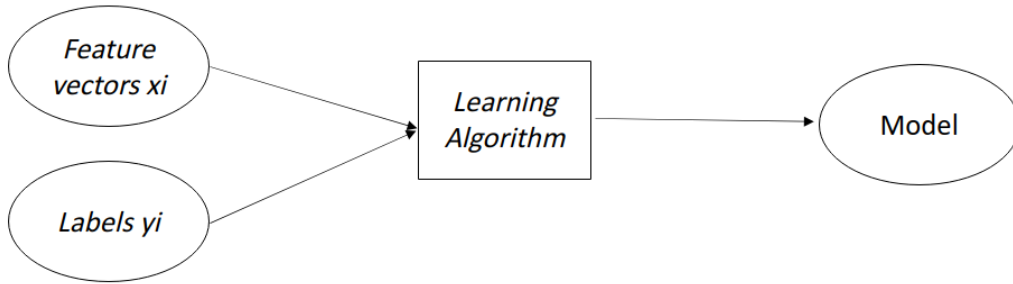


Figure 2.2: Supervised learning. The learning algorithm is given both feature vectors and their label to produce a model.

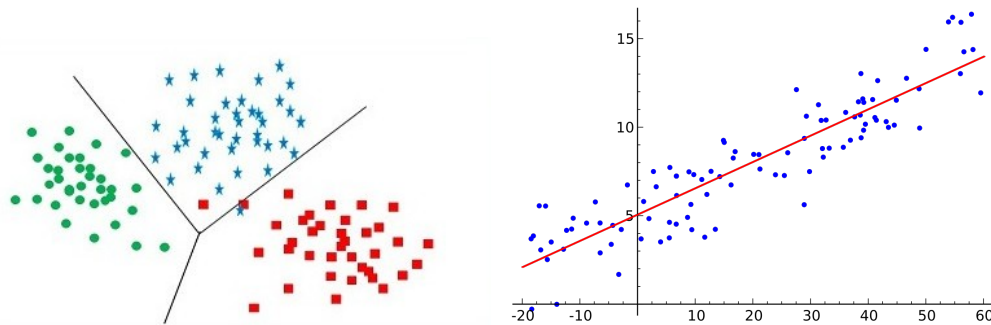


Figure 2.3: In classification (left), one draws boundaries between each class. In regression (right), one learns a function that fits the data.

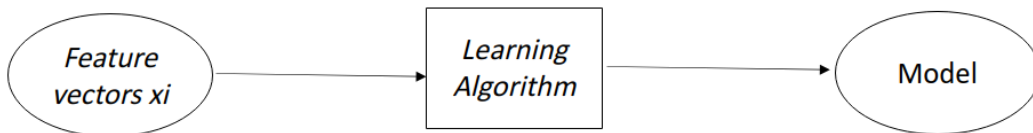


Figure 2.4: Unsupervised learning. The learning algorithm is given only feature vectors to produce a model.

Attack timing	Decision time (e.g. evasion attack) vs. training time (poisoning)
Attacker information	White-box vs. black-box attacks vs. grey-box attacks
Attack goals	Targeted attacks vs. reliability attacks

Table 2.1: *The three dimensions of attacks on machine learning (from [31])*

- rewards of the form $R_a(s, s')$, which denotes the reward given by reaching state s' when leaving state s through action a .

The objective is to learn the best policy, for instance, for each state of the state space, the action maximizing the expected reward.

2.2 Adversarial Machine Learning

Adversarial machine learning studies techniques used by malicious persons, called adversaries, to attack machine learning models or machine learning algorithms. Such attacks can be classified in various ways. We first model this variety of attacks along three dimensions. We then describe the three main categories of adversarial machine learning.

2.2.1 Dimensions of attacks

Kuleshov et al. propose a way to distinguish between different forms of attacks against machine learning in order to capture the most important features of attacks [14]. Accordingly, attacks are classified along three dimensions: attack timing, attacker information and attack goals. Table 2.1 gives a brief summary of these dimensions.

2.2.1.1 Attack timing

The attack timing differentiates attacks based on when they occur. Figure 2.7 shows where each of them happens.

There are attacks on algorithms, which occur at training time. They are usually called poisoning attacks. As these attacks modify a part of the training data (either raw or processed), they have a direct impact on the model created.

There are attacks on models, which occur at decision time. Such attacks are done after the model is learned. The attackers have access to the model, or at least decisions made by the model. In both cases, their attacks do not have the effect of changing the model. Most known decision-time attacks are evasion attacks.

2.2.1.2 Attacker information

The methods of attacks depend on the amount of information the adversary has access to. For training-time attacks, it is the amount of knowledge the attacker has on the learning algorithm. For decision-time attacks, it is the amount of knowledge they have on the learned model. Commonly, we divide attacks into white-box and black-box attacks.

In the case of white-box attacks, the attacker knows perfectly well the learning algorithm or model. In the case of poisoning attacks, the attacker has full knowledge of the hyperparameters of the learning algorithm. In the case of decision-time attacks, the attacker knows all model parameters.

It may seem strange at first glance for an attacker to have access to so much information. However such white-box attacks can happen. Let's consider the case of email spam detection. We can find in the literature many structures proposed for the model, including features and hyperparameters. We can find online public datasets for the training. If we suppose that the victims have trained their models accordingly to the literature, with a public dataset (or one close enough), the attacker has all the tools he needs to make a white-box attack.

Moreover, the open design principle in security states that the security of a mechanism should not depend on the secrecy of its design or implementation [2]. This principle opposes directly security by obscurity. In the case of adversarial machine learning, the open design principle translates in robustness of machine learning against white box attacks.

In the case of black-box attacks, the attacker has few to no information on the learning algorithm or model. For decision-time attacks, we can even subdivide black-box attacks, depending on the amount of information the attacker has access to. At the low end, the attacker would have no information. A luckier attacker would have access to some training data, different from the one used for the actual model, but no other knowledge. A more knowledgeable attacker could even know the model class and features, but not the training data. An even more informed one could have all of the precedent plus training data drawn from the same distribution

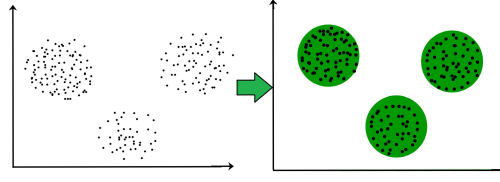


Figure 2.5: *Example of clustering. The learning algorithm is given data points that he separates into clusters.*

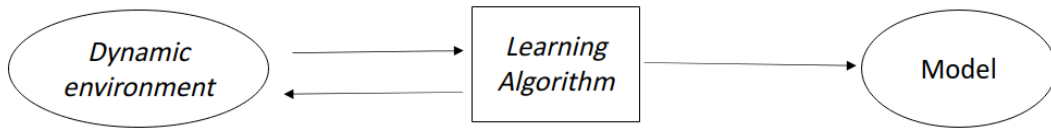


Figure 2.6: *Reinforcement learning. The learning algorithm communicates with a dynamic environment to learn and produce a model.*

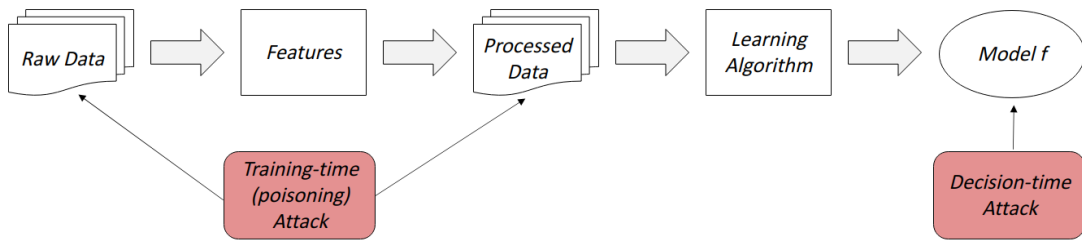


Figure 2.7: *Attack timing (from [14]). Training-time attacks happen on raw data or processed data. Decision-time attacks only occur on the finalized model.*

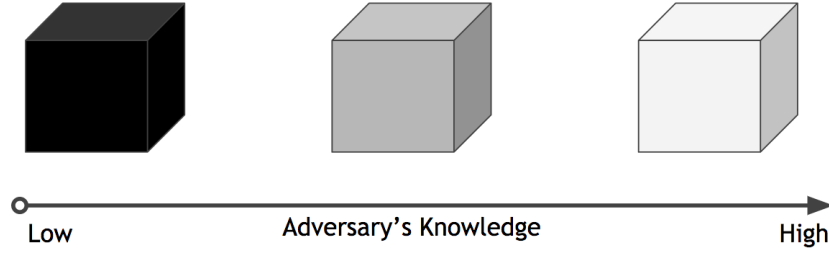


Figure 2.8: *Types of attacks depending on the adversary’s knowledge, from no knowledge (black-box), to some information (gray-box), to total knowledge (white-box) (from this [source](#)).*

as the training data used for the actual model. In contrast to the white-box attack, there are different kinds of black-box attacks. [Biggio and Roli](#) introduced the term gray-box attacks to denote cases where the attacker has an incomplete knowledge. For poisoning attacks, the principles apply. An attacker could have, or not, knowledge of the algorithm used, the hyperparameters or even features.

2.2.1.3 Attack goals

The purpose of attacks is another important dividing factor. There are two main categories of goals: the targeted attacks and the reliability attacks.

Targeted attacks have for objective to cause a mistake on specific instances of a specific nature. For example, let’s say we have a specific instance x of true label y and target label t with $t \neq y$. The objective in this case is to cause the decision function $D(x)$ of the learned model to predict, for the instance x , the label t instead of the true label y .

Reliability attacks have for objective to maximize the number of mistakes made by the learned model when making decisions. In supervised learning, this translates to maximizing the prediction error. Reliability attacks are also commonly called untargeted attacks.

The difference between targeted and reliability attacks can be clearly seen with multi-class classifier. Where targeted attacks have specific instances and a clear target label, reliability attacks consider all instances and do not have a specific target label. For a particular instance x of true label y , any label $t \neq y$ would be a potential target label in the case of reliability attacks.

2.2.2 Main types of attacks

There are many ways for an adversary to mount machine learning attacks. There are however only three main strategies that are commonly used in adversarial machine learning. Those strategies are data poisoning, evasion attacks and model stealing.

2.2.2.1 Data poisoning

In contrast to decision-time attacks, data poisoning occurs during training. Data poisoning attacks modify either the raw data or the processed data. Learning algorithms then use the poisoned data to train models. With this type of attack, the attacker aims to change the behavior of models.

Depending on the attacker's capabilities, the attack can be further divided into four attacks subcategories, three of which are illustrated in Figure 2.9:

- Label modification attacks: this type of attack exists only in the case of supervised learning. The attacker is limited to changing labels in the dataset. While such an attack can modify the label of an instance in this dataset, it is usually limited by a total modification cost. When restricted to binary classifiers, this attack is known as the label flipping attack.
- Poison insertion attacks: in this case, the attacker can insert his own feature vectors. Concerning supervised learning, whether the labels that go with those features are up to the attacker or not is variable. As for the previous attack, the number of inserted data is limited by an upper bound.
- Data modification attacks: in this case, the attacker can modify the features vectors and/or labels of the training dataset. The number of data modified is also limited.
- Boiling frog attacks: in this context, the defender is supposedly updating his model continuously. Exploiting this, the attacker can inject poisoned data in small doses. Each dose has a small effect and, over time, as the poisoned data accumulates, the model becomes heavily corrupted.

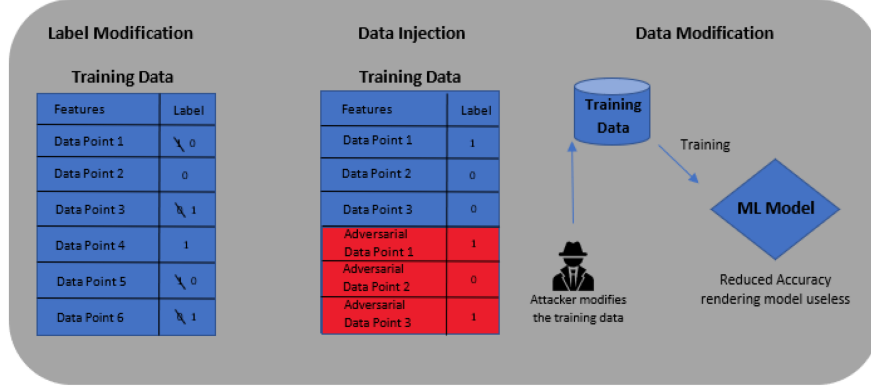


Figure 2.9: *Different types of data poisoning (from [5]).*

2.2.2.2 Evasion attacks

Evasion attacks are a type of decision-time attack, i.e. attacks made on a learned model when its training has been done. In this attack, the adversary composes an adversarial example that bypass the detection of the model. An adversarial example is defined as an input maliciously crafted with the purpose of being misclassified by a model. Techniques to build adversarial examples vary greatly from model to model.

In image classification, adversarial examples are most studied with state of the art neural network structures. By adding an adversarial perturbation to an image, we obtain adversarial examples. Figure 2.10 shows a famous example of adversarial examples. For the human eye, the original image and the produced adversarial examples are indiscernible. On the other hand, the machine learning model classifies the adversarial example as gibbon with great confidence.

To produce adversarial examples in image classification, many approaches have been proposed in the literature.

- L-BFGS approach [29]: Let f be the classifier function, $[0, 1]^m = \mathcal{X}$ the set of all possible inputs, $x \in \mathcal{X}$ the input and l the target label. One can find an adversarial example by resolving the following equation:

$$\begin{aligned}
 \min_r \quad & ||r||_2 \\
 \text{s.t.} \quad & f(x + r) = l \\
 & x + r \in \mathcal{X}
 \end{aligned} \tag{2.1}$$

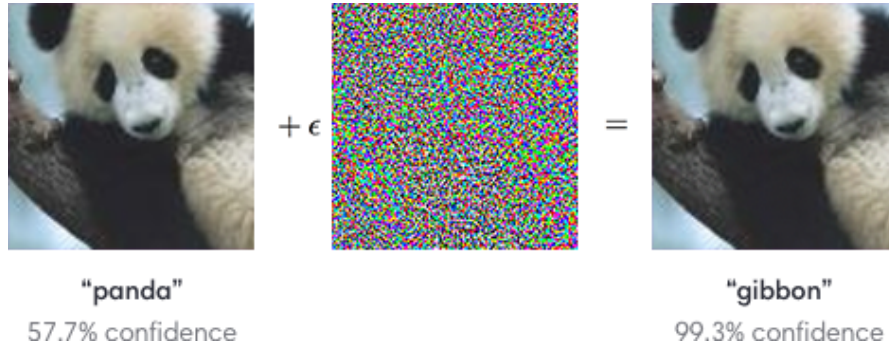


Figure 2.10: Construction of an adversarial example based on an image of a panda and an adversarial perturbation (from [8]).

We denote $x + r$ by $D(x, l)$. The sum $x + r$ is the closest image to x that f classifies as l . When $l = f(x)$, then $D(x + r, l)$ is simply x . The problem becomes non-trivial for $l \neq f(x)$. Generally, solving $D(x, l)$ is hard and a solution is to approximate the solution using a box-constrained L-BFGS. L-BFGS performs a line search to find

$$\begin{aligned} \min_{\epsilon < 0} \quad & c|r| + D(x + r, l) \\ \text{s.t.} \quad & x + r \in \mathcal{X} \end{aligned} \tag{2.2}$$

In the case of convex functions, the solution found is exact. Neural networks are non-convex in general, however, leading us to obtain an approximation. This method finds reliably adversarial examples.

- Fast Gradient Sign Method (FGSM) [8]: This method uses the gradients of the neural network to produce adversarial examples. Let θ be the parameters of a model, x the image input to the model, y the true label of image x and $J(\theta, x, y)$ the cost function. The method computes the adversarial example x^A as:

$$x^A = x + \epsilon \times \text{sgn}(\nabla_x J(\theta, x, y)) \tag{2.3}$$

This method of producing adversarial examples has the advantage of producing adversarial examples fast. One can also control by how much the original image is perturbed by tuning the parameter ϵ .

Adversarial examples are quite different in text-based classification. In a text, a word is either present or not. In an image, it makes sense to add a little bit of red in a pixel. In a text, it is absurd to add a little bit of the word 'cat' in a text. An adversary can modify a text based on three operations.

- Addition: By adding a word, a text can change his meaning. In 2005, it has been found that good word attacks were strong against statistical spam filters [18]. Good word attacks consists in adding words mostly present in non-spam texts to fool statistical spam filters.
- Deletion: Deleting words strongly connoted to the text class can lower the confidence with which machine learning models predict a class.
- Substitution: This operation consists in replacing a word in the text by another one. It can be seen as an addition followed by a deletion. However, it consists of a single operation.

2.2.2.3 Model stealing

Model stealing attacks happen at decision-time. In this type of attack, the attacker has access to a victim model only as a black-box. The attacker either wants to steal this model or obtain its training data it [13, 33].

A common setting for model stealing attacks are APIs. Some companies construct APIs to give public access to one of their machine learning models. Users are allowed to make queries to the model and are given back the model's prediction in response. Translation tools offered by Google and Amazon on their cloud platforms are one such case. The user provides the API a sentence in a language as query and obtain the sentence in another language as the model's prediction.

An attacker can take advantage of the APIs and tries to reverse-engineer the model. His goal is to create his own copy of the victim model with a performance as close as possible to the original one. If the attack is successful, the attacker has truly stolen the victim model and with it has stolen intellectual property. Stealing a model can pose other problems when the model or its training data are sensitive.

With this new model, the attacker gains many benefits. He has no need to further pay the company for access to its APIs as he can query his own model locally. He can also use this model to extract private information on the training data of the victim model. Moreover he can use this new model as a base to mount other types of attacks, e.g. train adversarial examples.

In finance, the attacker can steal a model for proprietary stock trading and benefits from it financially. In spam detection, the attacker can steal the classifier of some service and builds his own spam impervious to spam detection.

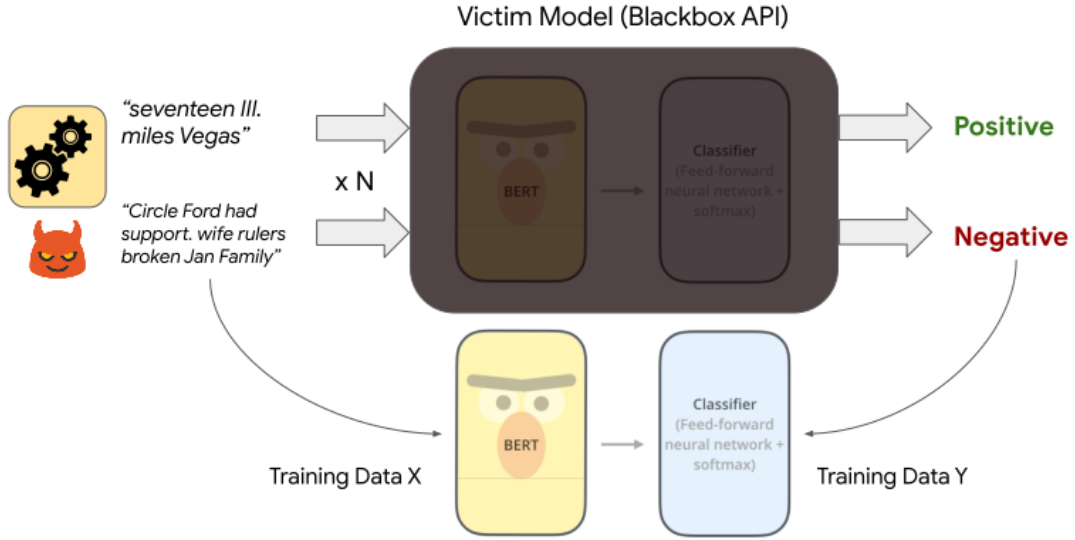


Figure 2.11: *Stealing a model (from [13]). The attacker queries the victim model random inputs and uses its feedback to build his own dataset.*

In their article, [Krishna and Papernot](#) gave a simple representation (see Figure 2.11) for attacks to steal the victim model.

- The attacker generates some input. [Krishna and Papernot](#) give two ways to do so. The first one is to build sentences of random words found in the WikiText-103 dataset. The second one is to take random sentences or part of sentences from the same dataset. In spam filtering, such queries can be sentences of random words.
- The victim model produces an output. In spam filtering, that would be spam or ham.
- The attacker aggregates all input/output pairs to form his own dataset.

2.3 Learning algorithms

In this section, we describe the learning algorithms used in this thesis. They are designed for supervised machine learning 2.2. Their role is to take preprocessed data and use it to build models.

Later, we defend our choice of learning algorithms. Convolution neural networks were chosen to represent word embedding models. Concerning the bag of word models, we chose five models for their diversity in structure: (i) multinomial naive Bayes classifier, (ii) logistic regression, (iii) linear support vector machine, (iv) random forest and (v) k-nearest neighbors.

2.3.1 Multinomial naive Bayes classifier

The multinomial naive bayes classifier is a simple and highly scalable classification algorithm. It is the multinomial version from the naive Bayes classifier. While naive Bayes classifier only considers feature vectors with binary values, multinomial naive Bayes classifier considers feature vectors with non-negative integers.

The algorithm aims to estimate the probability $P(y|x_1, \dots, x_n)$ of feature x belonging to class y . The most probable class is then selected as prediction. Using Bayes' theorem, we have that

$$P(y|x_1, \dots, x_n) = \frac{P(y)P(x_1, \dots, x_n|y)}{P(x_1, \dots, x_n)}$$

The naive conditional independence assumption specific to naive Bayes classifier is that $P(x_i|y, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = P(x_i|y)$. The search for the best class then becomes:

$$\begin{aligned} \text{Best } y &= \underset{y}{\operatorname{argmax}} P(y|x_1, \dots, x_n) \\ &= \underset{y}{\operatorname{argmax}} \frac{P(y)P(x_1, \dots, x_n|y)}{P(x_1, \dots, x_n)} \\ &= \underset{y}{\operatorname{argmax}} P(y)P(x_1, \dots, x_n|y) \\ &= \underset{y}{\operatorname{argmax}} P(y) \prod_{i=1}^n P(x_i|y) \end{aligned}$$

While we do not have access to the truth value of $P(y)$ and $P(x_i|y)$, we can compute estimates of those values $\overline{P(y)}$ and $\overline{P(x_i|y)}$, as follows. Let $\mathcal{T}$ be the training data. One has

$$\overline{P(y)} = \#\{(x^t, y^t) \in \mathcal{T} \mid y_t = y\}$$

Consider N_{yi} the number of times feature i appears in samples of class y in the

training set $\mathcal{T}$. It is characterized by

$$N_{yi} = \sum_{\substack{(x^t, y^t) \in \mathcal{T} \\ y^t = y}} x_i^t$$

Let $N_y = \sum_{i=1}^n N_{yi}$ be the number of total count for all features in samples of class y in the training set $\mathcal{T}$. Let $\alpha > 0$ be the smoothing parameter preventing zero probabilities. One has

$$\overline{P(x_i|y)} = \frac{N_{yi} + \alpha}{N_y + n\alpha}$$

Given some x , the prediction then becomes:

$$\begin{aligned} \text{Best } y &= \underset{y}{\operatorname{argmax}} \overline{P(y)} \prod_{i=1}^n \overline{P(x_i|y)}^{x_i} \\ &= \underset{y}{\operatorname{argmax}} \log(\overline{P(y)}) + \sum_{i=1}^n x_i \log(\overline{P(x_i|y)}) \end{aligned}$$

Learning a model on training set $\mathcal{T}$ is done in $O(n * |\mathcal{T}|)$ and predicting the class for a single sample is done in $O(n)$. From this, one can see why the multinomial naive Bayes classifier is highly scalable.

2.3.2 Logistic regression

In this section we restrain ourselves to logistic regression for two classes. We will not discuss multi-class logistic regression as it is not relevant to the case of spam detection.

In logistic regression, we consider that the probability of feature vector x belonging to class y is given by a logistic function with parameter θ . We consider here that class $y \in \{+1, -1\}$. As

$$P(y|x, \theta) = \frac{1}{1 + \exp^{-y(\theta^T x)}}$$

we obtain θ by maximizing the log-likelihood $\log(L(\theta; \mathcal{T}))$ and regularizing:

$$\begin{aligned}
L(\theta; \mathcal{T}) &= \prod_{(x^t, y^t) \in \mathcal{T}} P(y^t | x^t, \theta) \\
&= \prod_{(x^t, y^t) \in \mathcal{T}} \frac{1}{1 + \exp^{-y^t(\theta^T x^t)}} \\
\log(L(\theta; \mathcal{T})) &= - \sum_{(x^t, y^t) \in \mathcal{T}} \log(1 + \exp^{-y^t(\theta^T x^t)})
\end{aligned}$$

The regularization term $R(\theta)$ can take different forms:

- l_1 regularization: $R(\theta) = \|\theta\|_1$
- l_2 regularization: $R(\theta) = \frac{1}{2} \theta^T \theta$
- Elastic-Net regularization: $R(\theta) = \rho \|\theta\|_1 + \frac{1-\rho}{2} \theta^T \theta$ where $0 \leq \rho \leq 1$ controls the strength of l_1 regularization with respect to l_2 regularization.

Adding regularization to the mix, finding θ is resolving the following:

$$\min_{\theta, c} R(\theta) + C \sum_{(x^t, y^t) \in \mathcal{T}} \log(1 + \exp^{-y^t(\theta^T x^t + c)})$$

2.3.3 Random forest

Random forest is a type of ensemble method. It works by building many decision trees with bagging and by combining the answers given by the different trees to make a decision. Assume these trees are identified by indices $b = 1, \dots, B$. The procedure is in two steps:

1. Sample with replacement $(x^t, y^t) \in \mathcal{T}$ to compose a new training set $\mathcal{T}_b$.
2. Train decision tree $tree_b$ on $\mathcal{T}_b$.

For a new feature vector x , the value of y is predicted with a majority vote of all decision trees.

Now, how are decision trees learned? Figure 2.12 gives a schematic representation of a decision trees. For a new feature vector x , we begin at the root node of the

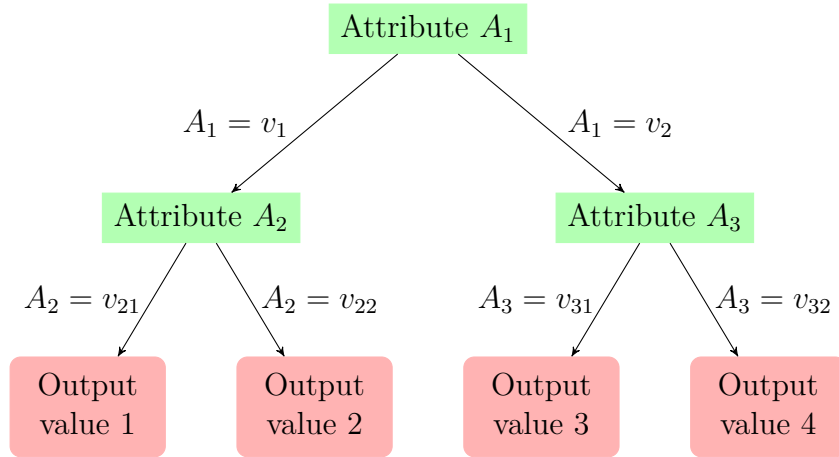


Figure 2.12: *Schema of a decision tree. A decision tree is composed of internal nodes (in green) testing attributes, branches (arrows) corresponding to attribute values and leaf nodes (orange) assigning classes.*

tree, i.e. the highest node in the tree structure. This node is an internal node and, as such, tests x on one of its attribute named A_1 . Depending on the value of attribute A_1 in x , one takes different paths. In our case, let's assume that we take the leftmost path. We arrive at a second node that tests the attribute A_2 of x . Depending on the attribute value, one ends up in one of two possible leaf nodes. Those nodes then give the class y corresponding with feature vector x .

Building the tree consists in finding the right attribute to be tested for each of the various nodes. For each node, this attribute is selected as to maximize the drop of impurity.

Let k be the current node. Let A be the attribute to test. Let S be the subset of samples from the training set $\mathcal{T}$ to take the path to node n . Let p_i be the proportion of samples from S with attribute value a_i for attribute A . The impurity can be calculated with the Gini index, $Gini = 1 - \sum p_i (p_i)^2$, or with the entropy, $E = -\sum p_i p_i \log_2(p_i)$. Results using both methods are similar.

2.3.4 K-nearest neighbors

K-nearest neighbors (KNN) is a non parametric algorithm. There are no parameters to learn and training is done instantly. Figure 2.13 illustrates how a K-nearest neighbors classifier works.

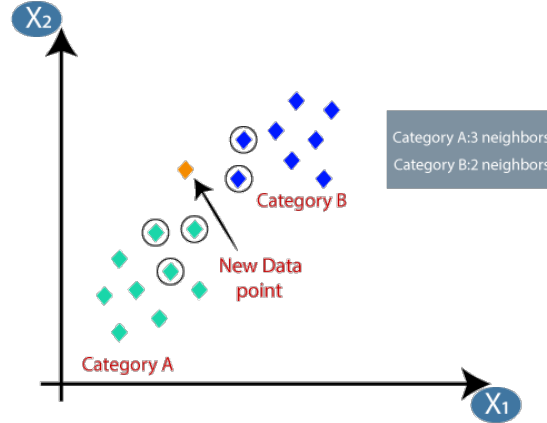


Figure 2.13: *Example of KNN classification. To classify the new data point (orange), the method looks for the 5 nearest neighbors. Through a majority vote, the new data point is predicted to belong to category A. (from this [source](#))*

- Let $\mathcal{T}$ denote the training data and x a feature vector whose label y we try to predict.
- Compute the distances $d_i = d(x_i, x)$ for all $(x_i, y_i) \in \mathcal{T}$.
- According to these distances, select $\mathcal{T}_k$, the set of k nearest training data points to x .
- The predicted class $\hat{y} = \underset{y}{\operatorname{argmax}} \sum_{(x_i, y_i) \in \mathcal{T}_k} w_i \times \delta_{y_i y}$ where the Kronecker delta $\delta_{y_i y} = 1$ if $y_i = y$ and 0 otherwise.

In the simplest versions of KNN, w_i is constant. The prediction then becomes a majority vote. In weighted KNN, w_i depends on d_i . The idea behind this weight is that the nearest points have a bigger say in the vote. A common way to do so is to set $w_i = 1/d_i$.

2.3.5 Linear support vector machine

In this section, we will discuss support linear vector machines for classification. Its application on regression will not be discussed. For more information on the subject of regression SVMs, the reader is invited to read the article “Support vector regression machines” [4].

Support vector machines (SVMs) are divided in two categories. On one hand there are linear SVMs which separates data points by a hyperplane. These type of SVMs are used for linearly separable data. In particular, data with high dimension features are more susceptible to be linearly separable. On the other hand, non-linear SVMs are used on non-linearly separable data. By using the kernel trick, non-linear SVMs project data into higher order dimensions (certain kernels project in an infinite number of dimensions) [9]. In this new space, non-linear SVMs separate data points by a hyperplane, in the same way linear SVMs separate data points in the original space.

Linear SVMs may be explained as follows. Let $\mathcal{T} = \{(x_i, y_i) | i = 1, \dots, n\}$ be the training set. Let i be a sample of the training data, $x_i \in \mathbb{R}^p$ be the feature vector of i and $y_i \in \{+1, -1\}$ be the class of i . According to the y_i 's, one may thus partition the x_i 's into two classes: those which correspond to $y_i = 1$ and those that correspond to $y_i = -1$. The goal of a linear SVM is to precisely find the hyperplane that separates the two classes with the most margin. The parameters of the hyperplane are $w \in \mathbb{R}^p$ and $b \in \mathbb{R}$, with the equation of the hyperplane being:

$$w^T x - b = 0$$

Once the hyperplane is found, the predicted class $\hat{y}$ for some x is given as:

$$\hat{y} = \text{sign}(w^T x - b)$$

Now, we make the assumption that the data is linearly separable. That means that there exist some w and b such that, for any i :

$$\begin{cases} w^T x_i > b, & \text{if } y_i = +1 \\ w^T x_i < b, & \text{if } y_i = -1 \end{cases}$$

In particular, it is also true that, for any i :

$$\begin{cases} w^T x_i - b \geq 1, & \text{if } y_i = +1 \\ w^T x_i - b \leq -1, & \text{if } y_i = -1 \end{cases}$$

which amounts to:

$$y_i(w^T x_i - b) \geq 1$$

The two planes $p^+ \equiv w^T x - b = 1$ and $p^- \equiv w^T x - b = -1$ separate data points. The margin M is defined as the distance between those two planes. Let $x^- \in p^-$. The closest point $x^+ \in p^+$ can be found by the relation:

$$x^+ = x^- + Mw$$

Given that $x^+ \in p^+$ and $x^- \in p^-$, one has:

$$\begin{aligned} w^T x^+ - b &= 1 \\ w^T(x^- + Mw) - b &= 1 \\ -1 + Mw^T w &= 1 \\ M &= \frac{2}{w^T w} \end{aligned}$$

The objective is to find the w that maximizes the margin M while correctly classifying the data points. Maximizing M is the same as minimizing $w^T w$, that is $\|w\|_2$. We need to solve:

$$\begin{aligned} \min_w \quad & \|w\|_2 \\ \text{s.t.} \quad & y_i(w^T x_i - b) \geq 1 \\ & \forall (x_i, y_i) \in \mathcal{T} \end{aligned}$$

This formulation doesn't allow for any data point to be misclassified. This is the formulation for hard margin. With soft margin, we allow more freedom to data. This is useful for data which is not linearly separable. In this version, we introduce the hinge loss function:

$$h(x_i, y_i) = \max(0, 1 - y_i(w^T x_i - b))$$

The hinge function $h(x_i, y_i)$ is non-zero only when the data point (x_i, y_i) violates the constraint $y_i(w^T x_i - b) \geq 1$. The hinge function penalizes violation of constraints. Using this function instead of constraints, the problem becomes:

$$\|w\|_2 + C \frac{1}{n} \sum_{i=1}^n \max(0, 1 - y_i(w^T x_i - b))$$

with C is a hyperparameter that regulates the cost of violations.

2.3.6 Convolution neural network

As their name indicates, convolution neural networks (CNN) are a type of neural networks composed of three parts: convolution layers, pooling layers and fully connected layers. We begin this section by presenting neural networks in general. We join this description by the method to train neural networks. We then describe

the convolution layers. We finish by presenting the pooling layers. It is worth noting that fully connected layers present no difference with neural networks. For this reason, there is no section on fully connected layers.

We illustrate an architecture for convolution neural networks in section 2.3.6. The reason behind this decision is that CNNs for text classification and image classification are very different in structure. In particular, CNNs for image classification have multiple convolution layers most of the time contrarily to CNNs for text classification. For more information on CNNs for image classification, the reader is invited to read the survey on CNNs for medical image analysis [34].

2.3.6.1 Neural network

Neural networks come in many shapes. In this subsection we will describe the multilayer perceptron (MLP). Other neural networks deviate from it in some ways. It is, however, a good model for comprehending neural networks in general. The following notations are inspired by the course “Machine learning : regression, deep networks and dimensionality reduction” [30].

As shown in figure 2.14, the multilayer perceptron is composed of neurons (circles) and connections between these neurons (arrows). Neurons take multiple inputs and produce a single value. Neurons are organized in layers in such a way that inputs for neurons of the l -th layer are the outputs of the neurons of the $l - 1$ -th layer. The inputs for the first layer neurons correspond to the input to the model. The outputs of the last layer of neurons correspond to the outputs of the model. Connections link neurons of succeeding layers.

Let $z_i^{(l)}$ denote the output of the i -th neuron of the l -th layer. Let L be the number of layers. Let $M_{(l)}$ be the number of neurons in layer l . Let $f^{(l)}$ be the activation function of layer l . Let $w_{ij}^{(l)}$ be the weight on the connection from the j -th neuron of layer $l - 1$ and the i -th neuron of layer l . This output $z_i^{(l)}$ is calculated as:

$$z_i^{(l)} = f^{(l)}\left(\sum_{j=1}^{M_{(l-1)}} w_{ij}^{(l)} z_j^{(l-1)}\right)$$

From this, in a two layer MLP, the output y of the model is:

$$y = f^{(2)}(w^{(2)} f^{(1)}(w^{(1)} x))$$

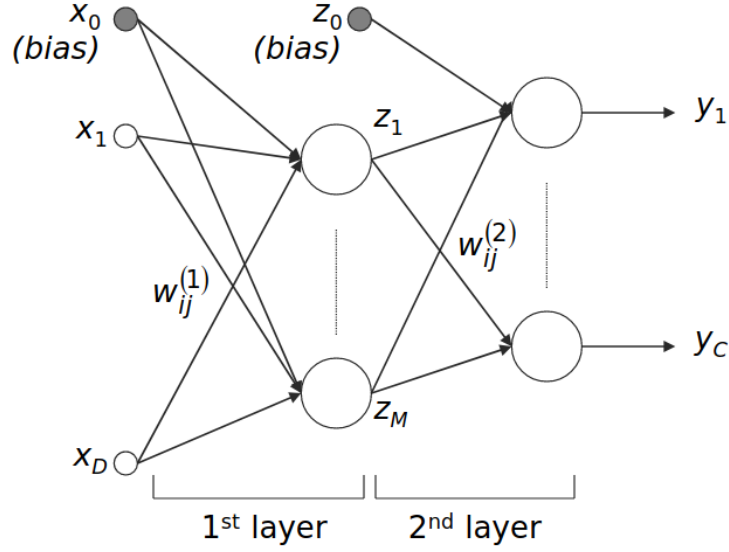


Figure 2.14: Sketch of a 2 layer MLP [30]

The activation functions $f^{(l)}$ are the only elements that induce non-linearity in the model. The most popular choice for the activation function are:

- The rectified linear unit (ReLU): $f(x) = \max(0, x)$.
- The logistic or sigmoid function: $f(x) = \frac{1}{1 + e^{-x}}$.
- The hyperbolic tangent: $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$.

These functions are differentiable everywhere with the exception of ReLU for $x = 0$. In practice, for ReLU, when $x = 0$, we use a subderivative. That being said, we can derivate any of those activation functions. This is crucial as it allows us to derivate the loss function through backpropagation and train the model weights. We will now describe the mathematics behind backpropagation.

Let us introduce the term $a_i^{(l)} = \sum_{j=1}^{M_{(l-1)}} w_{ij}^{(l)} z_j^{(l-1)}$. We now compute the gradient of the loss function E with respect to the weights w . For any weight $w_{ij}^{(l)}$, we have that:

$$\frac{\partial E}{\partial w_{ij}^{(l)}} = \frac{\partial E}{\partial a_i^{(l)}} \frac{\partial a_i^{(l)}}{\partial w_{ij}^{(l)}} = \frac{\partial E}{\partial a_i^{(l)}} z_j^{(l-1)}$$

We now introduce the term $\delta_i^{(l)} = \frac{\partial E}{\partial a_i^{(l)}}$. To compute this term, we distinguish two cases:

- The output layer ($l = L$):

$$\frac{\partial E}{\partial a_i^{(l)}} = \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial a_i^{(l)}} = \frac{\partial E}{\partial y_i} (f^{(l)}(a_i^{(l)}))'$$

$\frac{\partial E}{\partial y_i}$ is the derivative of the error criterion and is known. The expression $(f^{(l)}(a_i^{(l)}))'$ can be computed as all activation functions possess at least sub-derivatives.

- Hidden layers ($k \neq L$):

$$\begin{aligned} \delta_i^{(l)} &= \frac{\partial E}{\partial a_i^{(l)}} = \sum_{k=1}^{M_{(l+1)}} \frac{\partial E}{\partial a_k^{(l+1)}} \frac{\partial a_k^{(l+1)}}{\partial a_i^{(l)}} = \sum_{k=1}^{M_{(l+1)}} \left(\delta_k^{(l+1)} \frac{\partial \left(\sum_{j=1}^{M_{(l)}} w_{kj}^{(l+1)} z_j^{(l)} \right)}{\partial a_i^{(l)}} \right) \\ &= (f^{(l)}(a_i^{(l)}))' \sum_{k=1}^{M_{(l+1)}} \left(\delta_k^{(l+1)} w_{ki}^{(l+1)} \right) \end{aligned}$$

Using both formulas, one can successfully compute the gradient of the loss function with respect to the weights. Using gradient descent, one can update the weights and with them, train the model.

2.3.6.2 Convolution layer

In the convolution layer, the input x is convolved with a kernel (or filter) to produce some output. This input is then subjected to an activation function. Figure 2.15 illustrates the convolution part. Let us call the input $X \in \mathbb{R}^{n_1 \times n_2}$, the kernel $K \in \mathbb{R}^{d_1 \times d_2}$. Let us call the output $Y \in \mathbb{R}^{m_1 \times m_2}$ with $m_1 = n_1 - d_1 + 1$ and $m_2 = n_2 - d_2 + 1$. We say that $Y = X * K$ which is calculated as:

$$Y[i, j] = \sum_{k_1=0}^{d_1-1} \sum_{k_2=0}^{d_2-1} X[i + k_1, j + k_2] \times K[k_1, k_2]$$

To finish the convolution layer, Y is subject to the activation function f . The final output Z has the same size as Y and is calculated as:

$$Z[i, j] = f(Y[i, j])$$

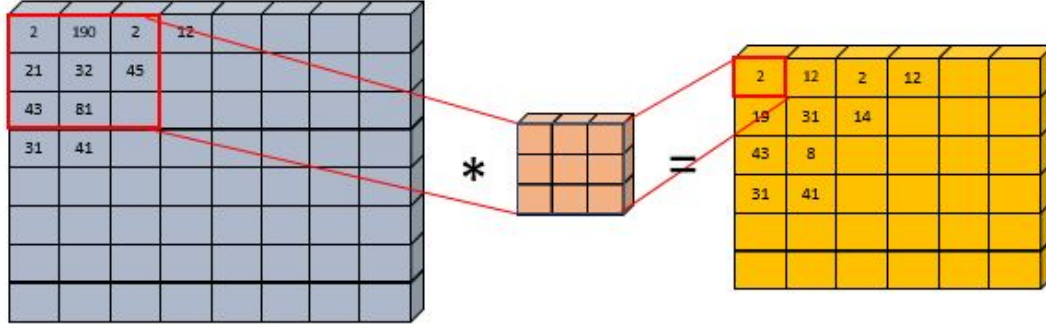


Figure 2.15: Convolution of some input (grey) with a kernel (orange) to produce some output (yellow) ([source](#)).

2.3.6.3 Pooling layer

The pooling layer takes this one dimensional vector z_i and outputs a single value q . The two main ways to do so are:

- Max pooling.

$$q = \max_{j=1}^{L-S_i} z_i[j]$$

- Average pooling.

$$q = \frac{1}{L - S_i} \sum_{j=1}^{L-S_i} z_i[j]$$

2.4 Conclusion

To address the subject of adversarial examples, one must first be given some background knowledge on machine learning and adversarial machine learning.

The first section of this chapter introduces machine learning as a whole. We begin the section by showing how machine learning models are learned while introducing basic vocabulary terms such as parameters and hyperparameters of a model. We then describe the three types of machine learning problems. We first discuss supervised learning, clarifying the difference between classification and regression and explain how hyperparameters can be tuned. We then turn to

unsupervised learning and mentioned a few applications. We finish the section by briefly describing reinforcement learning.

The second section of this chapter describes adversarial machine learning. The section begins by giving three dimensions along which attacks can be defined. We then describe the three most common strategies of attacks. To that end, we describe data poisoning attacks and its subcategories. We then detail evasion attacks and show how adversarial examples can be crafted. We finish by discussing model stealing, how it works and what its dangers are.

The third section describe the six machine learning algorithms of this thesis: (i) multinomial naive Bayes classifier, (ii) logistic regression, (iii) random forest, (iv) k-nearest neighbors, (v) support vector machine and (vi) convolution neural network. We chose these learning algorithms for their diversity in structure.

The book [31] on adversarial machine learning has been very inspiring to write sections 2.1.1, 2.2.1 and 2.2.2.1. References [13, 33] were useful to write section 2.2.2.3. References [8, 29] helped me to compose section 2.2.2.2. I warmly recommend any of those references to the reader who wants to know more about those subjects.

Chapter 3

Text classification background

This chapter is dedicated to giving the tools used for text classification. We discuss here two important aspects which are the preprocessing and the learning algorithms used.

Section 3.1 describes the preprocessing principles. In machine learning, data is of paramount importance. It is usually obtained in a raw form. For text related tasks, it is important to preprocess data correctly. Through tokenization, normalization and substitution, one obtains processed data which can then be used to build machine learning models.

Section 3.2 details how one can use the learning algorithms described in section 2.3 for text classification. In particular, it describes what inputs are given to the learning algorithms and give some more details on convolution neural networks specific to text classification.

Section 3.3 concludes the chapter in a few words.

3.1 Preprocessing

To be useful to machine learning models and learning algorithms raw data must first be transformed into processed data. Matthew Mayo distinguishes three main components in preprocessing for text-based classification such as spam detection[\[19\]](#). They are tokenization, normalization and substitution. It is important to note that preprocessing is a combination of all components. They interact with each other

and do not happen independently from each other.

To be useful to machine learning tools however, data must come as feature vectors. As such, we finish the section by presenting two types of feature representation for text.

3.1.1 Noise removal

Noise removal consists in removing noise from the data. This concept is very task-specific as the nature of the noise heavily depends on the application. For this reason, we will discuss the task of noise removal on the application at hand, that is emails.

Emails come in different formats. In particular HTML and XML formats, the most popular formats for emails, are structured with tags. Those tags enable much more than just text. For example, thanks to those formats, emails can contain hyperlinks, embedded images and tables. Addressing those points is the true essence of noise removal.

Concretely, for emails in HTML and XML formats, one can accomplish a number of transformations on the text.

- Emails possess headers and footers. It is of good practice to remove those.
- Emails in HTML and XML formats are structured by tags. One can choose to remove all those tags and ignore the metadata, e.g. hyperlinks and images.

3.1.2 Normalization

Through normalization, all texts are transformed and become similar in structure. For example, putting all characters in lower case is a type of normalization. Removing punctuation is another one. Normalization of text can mean a greater number of operations such as stemming, lemmatization or others.

Stemming consists in removing affixes (suffixes for example) from a word. For example, the word 'running' become the stem word 'run'.

Lemmatization differs in that it transforms a word in its citation form. The words 'was' and 'better' would return to their citation forms 'be' and 'good'.

Other types of normalization include:

- Converting numbers to a textual representation. Another possibility is to outright remove all numbers.
- Removing default stop words.

3.1.3 Tokenization

Tokenization is a process in which a text is decomposed in a list of tokens. So called tokens can be words, but also characters or sub-words.

Word tokenization is a very popular technique. Texts are divided into words based on delimiters. Delimiters are typically punctuation or whitespaces. In character tokenization, each character is a token. Compared to word tokenization, character tokenization results in more tokens by text. However, the size of the vocabulary is very small. Subwords are n-grams character, i.e. strings of n characters. Subword tokenization is midway between word and character tokenization.

3.1.4 Feature representation

After preprocessing, texts have been made into tokens. In text-based classification, there is a final step before data can be used for machine learning. This step is to make feature vectors from data. In this section, we describe two approaches to do so.

During experiments, depending on the learning algorithm, we used one or the other. We will use a bag of words approach for multinomial naive Bayes classifier, logistic regression, random forest, k-nearest neighbors, i.e. all learning algorithms except convolution neural network. For CNN, we will use a word embedding approach.

3.1.4.1 Bag of words

The method consists in viewing the data as a bag of words/tokens. By doing so, the word order is completely disregarded but its multiplicity is kept. Feature vectors are built through the following steps:

Raw data	John likes to watch movies. Mary likes movies too.
Preprocessed data	['john', 'likes', 'to', 'watch', 'movies', 'mary', 'likes', 'movies', 'too']
Vocabulary	{'bread', 'dislike', 'john', 'likes', 'mary', 'movies', 'watch', 'to', 'too'}
Feature vector	[0, 0, 1, 2, 1, 2, 1, 1, 1]

Figure 3.1: *Representation of data at various times. Data is shown in its raw form, after preprocessing and under the bag of words representation.*

1. Built a vocabulary of words V . Typically, this vocabulary is the set of all tokens in the training data after preprocessing. It can also be restricted to a portion of those tokens, the most common ones.
2. Each sample i is given its feature vector $x^i \in \mathbb{Z}_+^{|V|}$. Entry x_v^i ($v \in V$) gives the number of times the token v appears in the list of tokens of sample i .

Figure 3.1 illustrates data at various moments. Under its raw form, data is simply a long text. After preprocessing, data becomes a list of tokens. Under the bag of words representation, data becomes a vector of non-negative integers.

3.1.4.2 Word embedding

This method represents each word/token as a real-valued vector. This vector encapsulates the meaning of the word in such a way that words close in the vector space would have similar meaning [11]. In this section, we will present how one can transform preprocessed data into feature vectors using word embedding.

After preprocessing, each sample from data is represented as a list of tokens. Similarly to the bag of words method, we can build a vocabulary set V from all tokens in the training data. Let l^s denote the list of tokens for sample s of the data. Let l_i^s refer to the i -th token in the list of tokens l^s . Let $v \in V$ be a word from the vocabulary, $i = 1, \dots, |l^s|$. We can construct a feature vector x such that entry $x_{i,v}$ is 1 if the i -th token in l^s is v , 0 otherwise.

$$x_{i,v} = \begin{cases} 1, & \text{if } l_i^s = v \\ 0, & \text{otherwise} \end{cases}$$

Feature vector x is a sparse matrix of size $|l^s| \times |V|$. As l^s depends on the sample at hand, the size of the feature vector varies. A common way to address this issue is by padding. First we search for $L = \max_{s \in \mathcal{T}_p} |l^s|$ where $\mathcal{T}_p$ is the training set after preprocessing. We now proceed to padding. For $|l^s| < i \leq L$, for all $v \in V$, we impose $x_{i,v} = 0$. x is now a sparse matrix of size $L \times |V|$. This, in general, is very big. Fortunately, word embedding allows to reduce the feature dimension.

In this representation of x , each row $x_{i,:}$ is a sparse vector of size $|V|$ with one non-zero entry (or zero for added rows after padding). Embedding reduces the size of each row to $d \ll |V|$. The embedding operation can be seen as an embedding matrix. Let $E \in \mathbb{R}^{|V| \times d}$ be the embedding matrix, we have that $x^{\text{emb}} = xE$.

Building matrix E is trivial if we have access to a model M producing word embeddings. For any word $v \in V$, one has $E_{v,:} = M(v)$. The problem becomes to find models producing word embeddings. Word2vec consists in a group of related models that perform this exact task. More information on word2vec can be found in two papers [21, 22].

Figure 3.2 illustrates word embedding on a small example by tracing the evolution of data, from its raw form to a feature vector.

3.2 Learning algorithms

In this section, we detail how we use the learning algorithms presented in section 2.3 for text classification. Our approach is a bit different depending on whether we choose the bag of word representation of text or if we use word embedding.

This section also provides more details on the convolution neural networks. With the help of an illustration, we show the typical architecture of a CNN for text classification.

Raw data	John likes to watch movies.									
Preprocessed data	['john', 'likes', 'to', 'watch', 'movies']									
Vocabulary V	{ 'bread', 'dislike', 'john', 'likes', 'mary', 'movies', 'watch', 'to', 'too' }									
Feature vector x	'john'	0	0	1	0	0	0	0	0	0
	'likes'	0	0	0	1	0	0	0	0	0
	'to'	0	0	0	0	0	0	1	0	0
	'watch'	0	0	0	0	0	0	1	0	0
	'movies'	0	0	0	0	0	1	0	0	0
x after padding ($L=6$)	'john'	0	0	1	0	0	0	0	0	0
	'likes'	0	0	0	1	0	0	0	0	0
	'to'	0	0	0	0	0	0	1	0	0
	'watch'	0	0	0	0	0	0	1	0	0
	'movies'	0	0	0	0	0	1	0	0	0
	"	0	0	0	0	0	0	0	0	0
x after embedding ($d=2$)	'john'	0.8	0.7							
	'likes'	0.9	-0.1							
	'to'	0.1	0.2							
	'watch'	-0.6	-0.5							
	'movies'	-0.7	-0.3							
	"	0	0							

Figure 3.2: *Illustration of word embedding.*

3.2.1 Bag of word approach

For five out of six of our learning algorithms, we provide the texts under the bag of words representation. They are: (i) multinomial naive Bayes classifier, (ii) logistic regression, (iii) random forest, (iv) k-nearest neighbors and (v) linear support vector machine.

Using bag of words at the end of the preprocessing, any text t becomes a feature vector x of size $|V|$, where V is the vocabulary size. Also, we note that feature vector x is a vector of non-negative integers since it represents the number of occurrences of each word from the vocabulary V in t , the text of interest. We can then feed feature vector $x \in \mathcal{Z}_+^{|V|}$ to the learning models.

3.2.2 Word embedding approach

In contrast to the other learning algorithms presented above, the feature vectors for convolution neural networks (CNNs) do not come from the bag of words vectorization but follows the word embedding presented in section 3.1.4.2. The distinction is important as, in the case of bag of words, feature vectors each are of dimension $|V|$, the size of the vocabulary. In contrast, feature vectors after word embedding are of dimension $L \times d$ with L , the maximum number of words in text, and d , the embedding dimension.

3.2.3 Convolution neural network for text classification

Convolution neural networks (CNNs) are a particular type of neural networks characterized by their convolution layers. CNNs are composed of three parts, namely the convolution layers, the pooling layers and the fully connected layers. In text classification, CNNs possess a single convolution layer whereas in image classification, it is common to have multiple such layers. Pooling layers follow convolution layers. At the end of the CNNs are the fully connected layers whose structure is similar to the one described in section 2.3.6.1.

Figure 3.3 illustrates a simple CNN. Filters come in three region sizes (2, 3 and 4). There are 2 filters of each size. Filters perform convolutions on the input feature vector and output feature maps. Over each of the feature maps, the model performs the 1-max pooling, i.e., of each feature map, only the the largest number is kept. At this point, only 6 features remain (one for each filter). They are concatenated

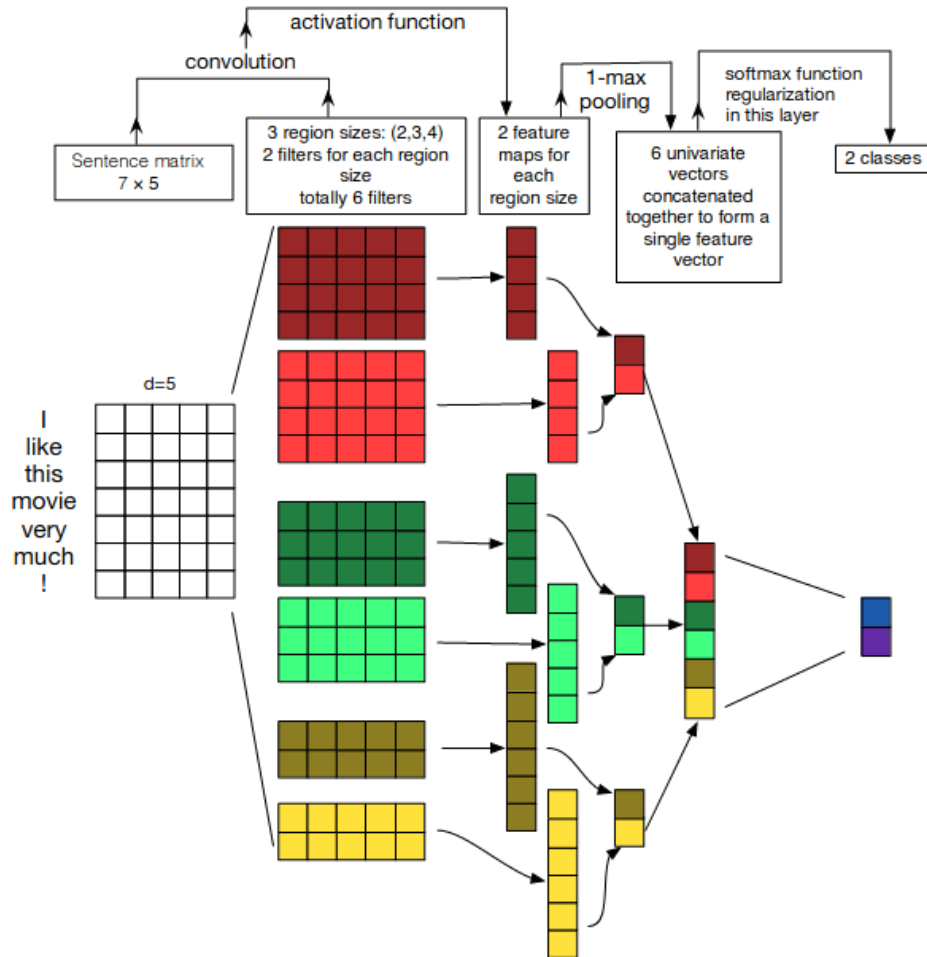


Figure 3.3: Example of a CNN for text-based classification. This model is composed of three layers. The first layer is a convolution layer that has 6 filters of 3 different sizes. The second layer is a pooling layer that does max pooling. The final layer is a fully connected layer (image from [36]).

into a feature vector and fed into a fully connected layer. The output of the model is the output of this final layer.

3.3 Conclusion

In this chapter, we discussed two important parts of text classification.

Section 3.1 discussed the first part that is the preprocessing of text. We explained the three main components proposed by Matthew Mayo [19]: noise removal, normalization and tokenization. On top of that, we talked about two feature representations of text: one based on bag of words, and the other based on word embedding.

Section 3.2 offered an insight in how one can use the learning algorithms previously described in section 2.3 in text classification. On top of that, this section presents through an example the typical structure a convolution neural network takes when applied to text classification.

With the background knowledge provided in this chapter, one can now build machine learning models to classify text. In two steps, one can (i) preprocess the raw data following the principles described in section 3.1, and then one can (ii) feed the preprocessed data into machine learning algorithms with section 3.2 for text specific knowledge and section 2.3 for details on the algorithms. This knowledge will be of great use in chapter 5 where we classify email spams using methods from text classification.

Chapter 4

Adversarial examples background

This chapter discusses the theory of evasion attacks on text classification.

Section 4.1 describes those attacks with four components: the goal function, the constraints, the transformations and the search method.

Some white-box attacks rely on the computation of a gradient. In section 4.2, we compute those gradients for some of the models presented in section 2.3. Random forests and k-nearest neighbors are not included since, because of their architectures, they cannot be the subject of mathematical derivations.

Section 4.3 describes two types of attacks from the literature: (i) text fooler and (ii) hot flip, a black-box and a white-box attack respectively. The description is made based on the four components presented in section 4.1.

4.1 Text attack

In section 2.2.1, we presented the three dimensions of attacks. In this categorization, attacks are separated depending on attack timing, attacker information and attack goals.

In this thesis, we focus ourselves on adversarial examples for spam detection. All evasion attacks are made at decision time, which gives us the attack timing. Spam detection is a type of binary classification. As there are only two classes, there is no real separation between targeted and untargeted attacks, which gives us the

attack goals. Regarding the last dimension of attack, the attacker information can be any in the spectrum going from black-box to white-box.

Specifying the attacker information will not give us a single type of attacks. To be more specific in qualifying an evasion attack on text classification, the authors of the library “TextAttack” [25] presented four components. We present those components in this section. For a better visualization, figure 4.1 decompose all components as a tree.

4.1.1 Goal function

The goal function gives the objective of the attack. We discussed it previously in section 2.2.1.3. Figure 4.1 shows five different types of goals. Of those, only targeted and untargeted classification are relevant for crafting adversarial examples.

4.1.2 Constraints

Constraints restrain the range of valid transformations. There exists different types of constraints:

- *Overlap*. Constraints of this category focus on character-based properties. They are useful when one wants the perturbation to look like the input.
- *Semantics*. Those constraints focus on the semantics, that is the meaning of words. Through those, one restrains perturbations to be close in meaning to the input.
- *Grammar*. These constraints focus on syntactic properties. One may want the perturbed word to have the same part-of-speech (POS), i.e. replace a verb by a verb, a pronoun by a pronoun, etc.
- *Pre-transformation*. This category regroups constraints on the input only, disregarding the perturbation. These constraints can also be on the index of word. One such constraint could be to disable the modification of stop words.

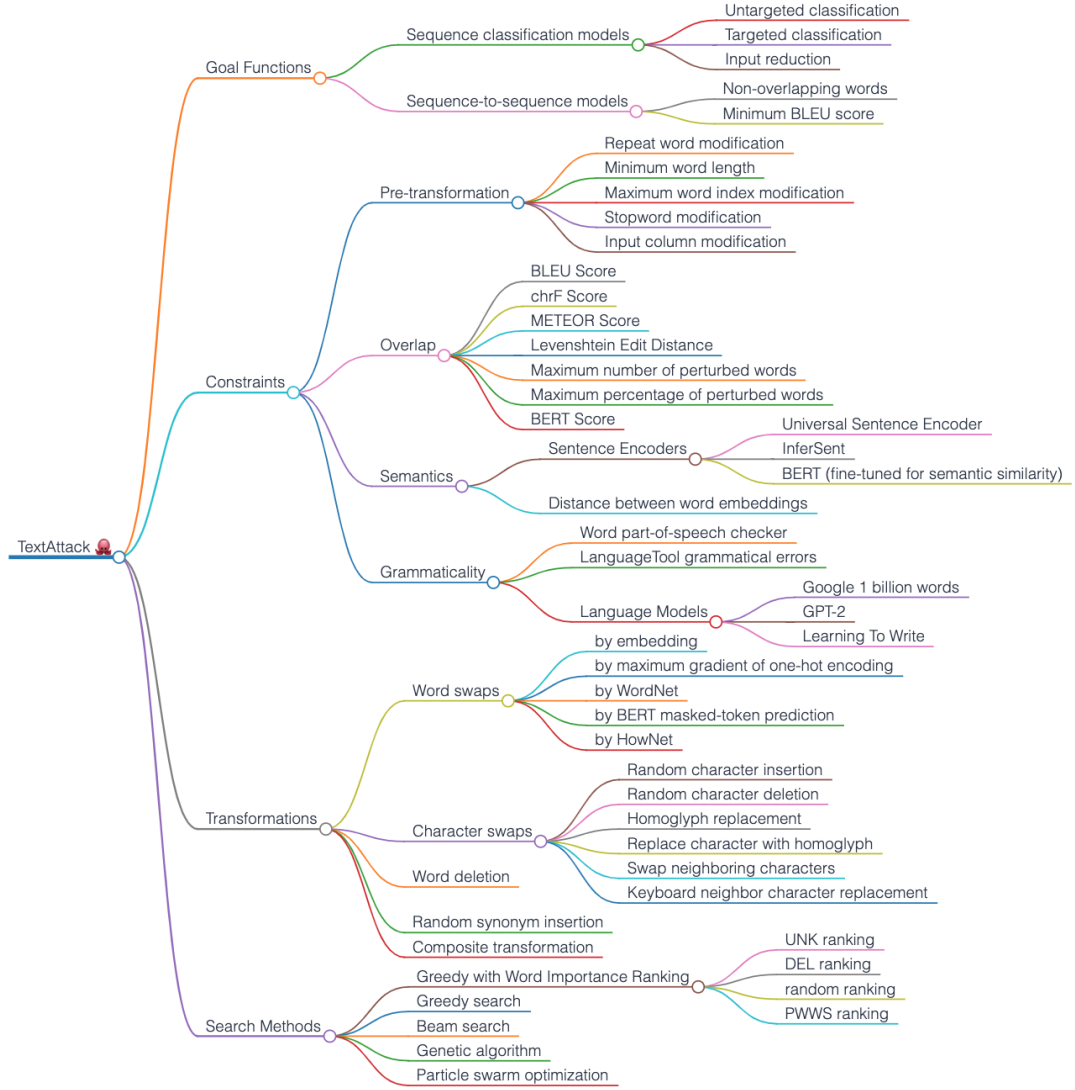


Figure 4.1: Summary diagram of the “TextAttack” ecosystem [25].

4.1.3 Transformation

This category considers the types of transformation a text undergoes. Figure 4.1 proposes 5 types of transformations:

- *Word swaps*. This type of transformation substitutes a word in a text by another one. The length of the text stays the same. The words considered as substitutes are different depending on the technique (embedding, maximum gradient of one-hot encoding, etc.).
- *Character swaps*. In this case, one doesn't change the whole words but inserts, removes or substitutes characters in it.
- *Word deletion*. Here, we modify the text by removing a word.
- *Word insertion*. We insert new words in a text.
- *Composite transformation*. This transformation allows for considering a variety of transformations at the same time. For examples, one may want to do word swaps and word deletion at the same time.

4.1.4 Search method

Given a goal function, constraints and transformations, an exhaustive search of the space for adversarial examples is most of the times infeasible. Multiple search methods have been proposed to offer adversarial examples of good quality. We will use two of those methods in this thesis:

- *Greedy search with Word Importance Ranking*. In this method, words are ranked according to their importance. This importance of a word is measured by deleting the word from the spam email and then measure the drop in confidence of the model. The bigger the drop, the more important the word is.
- *Beam search*. For a beam width of b and a maximum of r transformations, the beam search on text input x_0 proceeds as such:
 1. We construct $S := \{I\}$, the set of best transformations; I is the identity transformation, such that $I(x_0) = x_0$.
 2. We construct $C := \{\}$, the set of all candidate transformations.

3. For each transformation t in S :
 - We build x_t , the text x_0 after transformation t .
 - For each word w in x_t :
 - * Consider substitution of word w in text w_t . This gives us a single transformation t' .
 - * We add the composite transformation $t' \circ t$ to C
4. We reconstruct S as the b best transformations from C .
5. When the transformations in S are composed of r singular transformations, we return the best transformation in S . Otherwise we go back to step 2 and reiterate.

4.2 Computing gradient

It is common for white box attacks to rely on the gradients. These gradients are computed on a loss function with respect to the input. Some models, such as the multinomial naive Bayes classifier, don't rely on the notion of loss function. In those cases, we first propose a loss function and then compute the gradient.

This section aims at presenting the mathematical foundations to compute the gradient of all derivable models from section 2.3.

4.2.1 Multinomial Bayes classifier [2.3.1]

Let x be the spam email we want to modify. To classify x , the multinomial Bayes classifier proceeds by giving a score s_y to each class y , i.e. “spam” or “ham” in our case:

$$s_y(x) = b_y + \theta_y^T x$$

where b_y and θ_y^T are the parameters of the model.

The class predicted is then simply given by the best score:

$$y^* = \operatorname{argmax}_y s_y(x)$$

This comes from the multinomial Bayes classifier computing the probability $P(y|x)$

that the text belongs to class y , given the feature vector x :

$$P(y|x) = \frac{P(y)P(x|y)}{P(x)} = C \times e^{s_y(x)}$$

where $C = \frac{1}{P(x)}$ is constant with respect to the class y . Probabilities must sum to 1, that is $\sum_{y'} P(y'|x) = 1$. From this, we know that:

$$P(y|x) = \frac{e^{s_y(x)}}{\sum_{y'} e^{s_{y'}(x)}}$$

Let $l_x(a, b)$ be a 0-1 loss function:

$$l_x(a, b) = 1 - \delta_{ab} = \begin{cases} 1, & \text{if } a \neq b \\ 0, & \text{otherwise} \end{cases}$$

where δ_{ab} is the Kronecker delta. The loss function we will consider is $\mathcal{L}_x(y)$, the expected 0-1 loss function:

$$\begin{aligned} \mathcal{L}_x(y) &= \sum_{y'} l_x(y, y') P(y'|x) \\ &= \sum_{y'} (1 - \delta_{yy'}) P(y'|x) \\ &= \sum_{y'} P(y'|x) - \sum_{y'} \delta_{yy'} P(y'|x) \\ &= 1 - P(y|x) \\ &= 1 - \frac{e^{s_y(x)}}{\sum_{y'} e^{s_{y'}(x)}} \end{aligned}$$

Using the fact that $\nabla_x s_y(x) = \theta_y$, we can compute the gradient of the loss function with respect to the input x :

$$\begin{aligned} \nabla_x \mathcal{L}_x(y) &= \nabla_x \left(1 - \frac{e^{s_y(x)}}{\sum_{y'} e^{s_{y'}(x)}} \right) \\ &= -\nabla_x \frac{e^{s_y(x)}}{\sum_{y'} e^{s_{y'}(x)}} \\ &= -\frac{\theta_y e^{s_y(x)} (\sum_{y'} e^{s_{y'}(x)}) - e^{s_y(x)} (\sum_{y'} \theta_{y'} e^{s_{y'}(x)})}{(\sum_{y'} e^{s_{y'}(x)})^2} \\ &= P(y|x) (-\theta_y + \sum_{y'} P(y'|x) \theta_{y'}) \end{aligned}$$

When there are only two classes, one can further simplify the equation. Let us consider the use of $+$ and $-$ as subscripts to denote the two classes. Without loss of generality, one can write:

$$\begin{aligned}\nabla_x \mathcal{L}_x(y_+) &= P(y_+|x)(-\theta_{y_+} + P(y_+|x)\theta_{y_+} + P(y_-|x)\theta_{y_-}) \\ &= P(y_+|x)(-(1 - P(y_+|x))\theta_{y_+} + (1 - P(y_+|x))\theta_{y_-}) \\ &= P(y_+|x)(1 - P(y_+|x))(\theta_{y_-} - \theta_{y_+})\end{aligned}$$

4.2.2 Linear support vector machine [2.3.5]

To train support vector machines, we introduced the hinge loss function:

$$h(x_i, y_i) = \max(0, 1 - y_i(w^T x_i - b))$$

This loss function is somewhat lacking. Indeed, when $1 - y_i(w^T x_i - b) < 0$, that is, when x_i is well classified as y_i , the gradient $\nabla_x h(x_i, y_i) = \vec{0}$. In this thesis, we have a well classified spam on which we want to compute a gradient. This gradient would be 0 most of the times, which is not desirable.

We modify the loss function to become:

$$\mathcal{L}(x_i, y_i) = 1 - y_i(w^T x_i - b)$$

The difference is that $\mathcal{L}(x_i, y_i)$ always gives a non zero gradient. We can observe that for well classified spams, one has $1 - y_i(w^T x_i - b) < 0$ and therefore the loss is negative.

The gradient is simple to compute:

$$\begin{aligned}\nabla_x \mathcal{L}(x, y) &= \nabla_x (1 - y(w^T x - b)) \\ &= -yw\end{aligned}$$

4.2.3 Logistic regression [2.3.2]

The loss function minimized at training time is:

$$\min_{\theta, c} R(\theta) + C \sum_{(x^t, y^t) \in \mathcal{T}} \log(1 + \exp^{-y^t(\theta^T x^t + c)})$$

Let us consider the contribution of a single point to this loss function:

$$\mathcal{L}(y, x) = C \log(1 + \exp^{-y(\theta^T x + c)})$$

The gradient is:

$$\begin{aligned} \nabla_x \mathcal{L}(y, x) &= \nabla_x (C \log(1 + \exp^{-y(\theta^T x + c)})) \\ &= C \frac{1}{1 + \exp^{-y(\theta^T x + c)}} \exp^{-y(\theta^T x + c)} (-y\theta) \\ &= C (1 - P(y|x)) (-y\theta) \\ &= -y C (1 - P(y|x)) \theta \end{aligned}$$

$$\text{where } P(y|x) = \frac{1}{1 + \exp^{-y(\theta^T x + c)}}.$$

4.2.4 Convolution neural network [2.3.6]

In section 2.3.6.1, we introduced a loss function E . To keep our conclusion relevant for any valid loss function, we did not give a specific formula for E . To train our neural network, we then derived this loss function E with respect to the weights of the model w . Now, we will compute the gradient of this still generic loss function E with respect to input x .

A requirement we impose on the loss function E is to be differentiable with respect to the output $y = f(x)$. We note that y is a vector of dimension M , where M is the number of classes. From this, we can establish that:

$$\nabla_x E = \nabla_y E \cdot \nabla_x y = \nabla_y E \cdot \nabla_x f(x)$$

We chose some loss function E so that the term $\nabla_y E$ is easy to compute. We will now focus on the term $\nabla_x f(x)$. It is worth noting that the gradient $\nabla_x E$ is a vector of same size as x , that is of dimension N . This stems from the fact that $\nabla_y E \in \mathbb{R}^M$ and $\nabla_x f(x) \in \mathbb{R}^{M \times N}$.

In our implementation, the convolution neural network is composed of an embedding layer (producing z_1), a convolution layer (producing z_2), and two hidden layers (z_3 and z_4).

$$\begin{array}{ccccc}
x & \xrightarrow[E]{\text{Embedding}} & z_1 & \xrightarrow[C+\text{Maxpool}]{\text{Convolution}} & z_2 \\
& & & & \\
& \xrightarrow[D_1+\text{ReLU}]{1^{st} \text{ hidden}} & z_3 & \xrightarrow[D_2+\text{ReLU}]{2^{nd} \text{ hidden}} & z_4 \\
& & & & \\
& \xrightarrow[D_3+\text{Softmax}]{\text{Output}} & & & f(x)
\end{array}$$

We have that E , C , D_1 , D_2 and D_3 are matrices. Let us denote the functions Maxpool, ReLU and Softmax as m , r and s . We have:

$$f(x) = s\left(D_3 \cdot r\left(D_2 \cdot r\left(D_1 \cdot m(C \cdot Ex)\right)\right)\right)$$

By using the classical derivative law for composed functions

$$\frac{\partial g_1(g_2(x))}{\partial x_i} = \frac{\partial g_1(t)}{\partial t} \Big|_{t=g_2(x)} \frac{\partial g_2(x)}{\partial x_i}$$

we have that:

$$\frac{\partial f(x)}{\partial x_i} = \frac{\partial s(t)}{\partial t} \Big|_{t=z_4} \cdot D_3 \cdot \frac{\partial}{\partial x_i} r\left(D_2 \cdot r\left(D_1 \cdot m(C \cdot Ex)\right)\right)$$

After applying the same trick a couple more times, we obtain:

$$\frac{\partial f(x)}{\partial x_i} = \frac{\partial s(t)}{\partial t} \Big|_{t=z_4} \cdot D_3 \cdot \frac{\partial r(t)}{\partial t} \Big|_{t=z_3} \cdot D_2 \cdot \frac{\partial r(t)}{\partial t} \Big|_{t=z_2} \cdot D_1 \cdot \frac{\partial m(t)}{\partial t} \Big|_{t=z_1} \cdot C \cdot E_{i,:}$$

It is here worth recalling that, as already mentioned in section 2.3.6, one can always compute at least subderivatives for all activation functions.

Softmax derivative. Let us denote s as the softmax function. For some input x , $s(x) = S$ with S , the output. S and x are of same size N . For any $j = 1, \dots, N$ one has

$$S_j = \frac{e^{x_j}}{\sum_{k=1}^N e^{x_k}}$$

To simplify the expression, we will denote the denominator $\sum_{k=1}^N e^{x_k}$ as D for the rest of the development. We wish to know the partial derivatives:

$$\begin{aligned}
\frac{\partial S_j}{\partial x_i} &= \frac{\partial \frac{e^{x_j}}{D}}{\partial x_i} \\
&= \frac{\frac{\partial e^{x_j}}{\partial x_i} D - e^{x_j} \frac{\partial D}{\partial x_i}}{D^2} \\
&= \frac{\delta_{ij} e^{x_j} D - e^{x_j} e^{x_i}}{D^2} \\
&= \delta_{ij} S_j - S_j S_i
\end{aligned}$$

where δ_{ij} is the Kronecker delta, characterized by $\delta_{ij} = 1$ if $i = j$ and 0 otherwise.

ReLU derivative. Let us denote by r the ReLU activation function. For some input x , $r(x) = R$ with R , the output. R and x are of same size N . For any $j = 1, \dots, N$ one has

$$R_j = \max(0, x_j)$$

The partial derivatives are then simply:

$$\frac{\partial R_j}{\partial x_i} = \begin{cases} 1, & \text{if } i = j \text{ and } x_i > 0 \\ 0, & \text{otherwise.} \end{cases}$$

Max pooling derivative. Let x be the input to the max pooling layer. In our implementation, x is composed of three sub inputs x_1 , x_2 and x_3 . This comes from the fact that we use three different sizes of kernels.

Let L be the length of texts. Let k_i be the i th size of the kernel. Let $nfilters$ be the number of filters of each size. The input x_i is a vector of size $(L - k) \times nfilters$. The output y_i is of size $nfilters$. Let $b = 1, \dots, nfilters$, we have:

$$y_i[b] = \max(x_i[:, b])$$

For element $x_i[a, b]$, the derivative is thus:

$$\frac{\partial y_i[c]}{\partial x_i[a, b]} = \begin{cases} 1, & \text{if } c = b \text{ and } a = \operatorname{argmax}_{a'} x_i[a', b] \\ 0, & \text{otherwise.} \end{cases}$$

4.3 Attacks

To build adversarial examples, we used multiple attack recipes. We provide a description of each of those attacks accordingly to the four components described in section 4.1.

4.3.1 Text fooler

This attack comes from the paper “Is BERT Really Robust? A Strong Baseline for Natural Language Attack on Text Classification and Entailment” [10] published in 2019. The main points of this attack are:

- It is an untargeted attack.
- The transformations considered are word substitutions. Words are substituted using word embedding to only consider the 50 closest candidates. To measure which of the candidates is the best, one observes the confidence given by the model. The altered text with the best candidate will give the lowest confidence for the spam class.
- The constraints are multiple:
 - There is a minimum threshold for the cosine similarity between the original word and its substitution in the word embedding space. We set this threshold to 0.5.
 - The adversarial example need to share a similarity of 84% with the original spam, i.e. the number of modified words must not exceed 16%.
 - We conserve the part of speech when replacing words, i.e. verbs are replaced by verbs. The only exception are nouns that can also be replaced by verbs.
- The search for adversarial examples is done through the Greedy search with Word Importance Ranking from section 4.1.4.

This attack only makes use of some confidence measure from the model it attacks. As the information of the attacker stays limited, the attack can be considered as black box.

4.3.2 Hot-flip

This attack comes from the paper “HotFlip: White-Box Adversarial Examples for Text Classification” [6] published in 2017. The main points of this attack are:

- It is an untargeted attack.
- The transformations are word substitutions. We use the gradient of the model with respect to the one-hot input to estimate which individual change has the highest estimated loss.
- There are a number of constraints:
 - Stop words are not modified.
 - The maximum number of words perturbed is 2.
 - As for the previous attack, there is a minimum threshold for cosine similarity. Here, we set it to 0.8.
 - When replacing words, we conserve the part of speech.
- It uses the beam search presented in section 4.1.4.

As this attack uses the gradient of the model attacked, it is a white box attack. We presented an intuitive but naive implementation for the beam search in $O(br|x_0|)$. The authors of the paper, however, proposed an implementation of the beam search in $O(br)$.

4.4 Conclusion

In this chapter, we went through the theory of adversarial examples for text classification.

In section 4.1, we presented a framework for attacks that generate those adversarial examples. This framework comes from the TextAttack python library [25]. Attacks have an objective which, in TextAttack, translates in a goal function; in our case, it is untargeted classification. Attacks operate a series of transformations on the original text. Examples of such transformations are word deletion and insertion. Attacks impose constraints on those transformations. And finally, attacks follow a search method to reach our objective.

In the literature, some attacks [6] are gradient based. In section 4.2, we showed how such a gradient can be found for all models that can be derived.

In section 4.3, we presented two attacks from the literature that are very different in their composition.

This chapter gives the tools to create adversarial examples. Using the attacks described in section 4.3, one can attack machine learning models described in section 2.3. Section 4.2 is essential for white-box attacks such as hot-flip described in section 4.3.2. Using the principles in section 4.1, one can modify the attacks to fit the particular use case. In our case, we later introduce particular structures in section 5.2.2. These particular structures contain precious information and attacks cannot delete them. Using the previously mentioned principles, we can add this constraint to our attacks.

All the theory discussed here will be particularly useful in chapter 6 where we will attack our models built in chapter 5.

Chapter 5

Spam detection

In this chapter, we address the question of detecting email spam: can we build machine learning models that reliably detect email spams?

To answer the question, one must know what exactly email spams are. To this end, the first section introduces email spams in general and describes the task of spam detection.

The core of this chapter is the second section, in which we put in practice all bits of knowledge discussed in chapter 3. Section 5.2 first gives the methodology we follow to answer the question: (i) data collection, (ii) data preprocessing, (iii) data exploration and visualisation, (iv) model building and (v) model evaluation. The rest of the section describes the methodology and gives the results obtained from it. We get our data from both the Enron email archive and Apache spam email dataset [7]. To be usable by machine learning models, we preprocess this data by using the techniques from section 3.1. This preprocessed data is then used to train the models described in section 2.3. Once done, we take the occasion to discuss the first results. In particular, we observe that our models perform well above random guess (94% for all), with the exception of our logistic regression model.

The last section presents a brief conclusion of the chapter. In particular, we answer the question of this chapter by the affirmative, by concluding that, except for our logistic regression model, all our models reliably detect email spams.

5.1 Spams

Spams are unsolicited messages. They can serve a wide range of purposes, from commercial advertising to non-commercial proselytizing or even phishing. The name come from Spam, a brand of canned cooked pork, after the sketch Spam by Monty Python in their show Monty Python's flying circus [1].

In 2020, spams accounted for 50.37% of email traffic [15]. This shows that it is essential to sort out spam from ham, i.e. good emails. Machine learning tools offer great accuracy in this task.

In machine learning, the task of spotting spams is known as spam detection. It is a supervised learning problem and a type of text-based classification. Inputs are texts, e.g. emails, and outputs can be one of the two classes spam or ham.

5.2 Spam detection in practice

In this section, we put theory into practice. We devise an experiment to answer the question of this chapter: can we build machine learning models that can reliably detect email spams? The experiment follows the framework illustrated in figure 5.1:

1. *Data collection* (section 5.2.1). One cannot do supervised machine learning (see section 2.1.2) without data. We devise a dataset with the Enron email archive and the Apache spam email dataset [7].
2. *Data preprocessing* (section 5.2.2). To be of any use to machine learning models, data is preprocessed.
3. *Data exploration and visualization* (section 5.2.3). Data exploration and visualization is important to monitor the quality of our preprocessed data. We iterate between this step and the preprocessing step until we obtain a satisfying preprocessed data.
4. *Model building* (section 5.2.4). Our data can now be used. We split the data into a training set and a test set following a 4:1 ratio. We use the learning algorithms from section 2.3 and the training set to train our models. To set the hyperparameters of our models, we use various techniques of cross-validation (see section 2.2). At this step, the test set is left untouched.

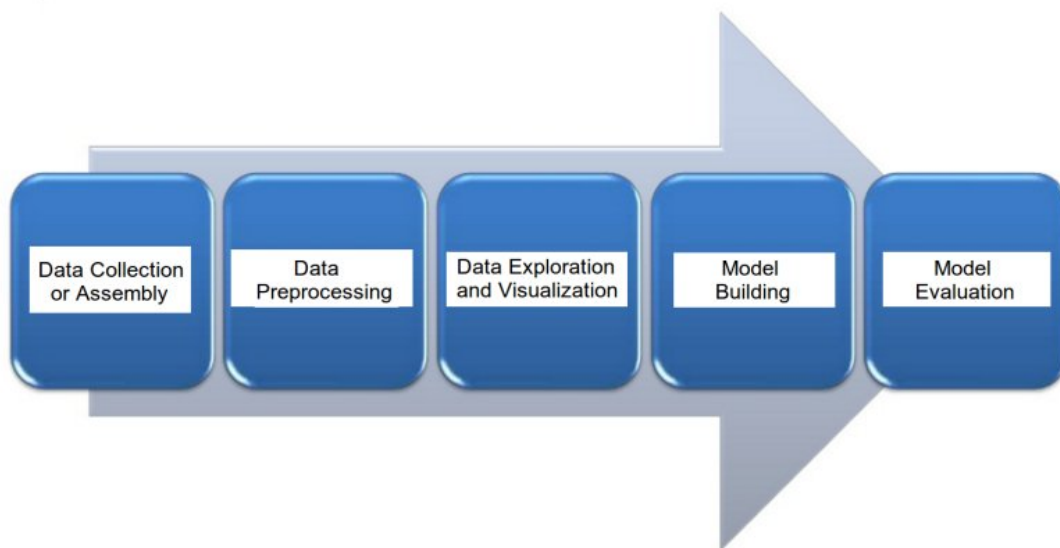


Figure 5.1: *Framework of a textual data task (from [12]).*

5. *Model evaluation* (section 5.2.5). We evaluate our models trained in the precedent step on the test set.

5.2.1 Data collection

By definition, email spams are emails unsolicited by the user. It is worth noting that what is considered to be a spam by a user is not necessarily considered to be a spam by another user. For instance, an email related to course X is of interest for students following it but is mostly of no interest to other students.

The issue with the definition of spam is thus that it depends on the user's categorization, meaning that a spam for user A could be a ham for user B. As a consequence, labeling emails is not an absolute task but is relative to a group of users. In order to proceed, we make the hypothesis that, the majority of time, email spams are considered as such by the crushing majority of users. With this hypothesis in mind, the dataset that will be of use to this document was found in [one of M. Elhadad's webpages](#) (see [7]). It is composed of two datasets, the Enron email archive and the Apache Spam Assassin dataset. In those dataset, the task of labeling emails as spams or hams is done by user reports.

We observe later that, with those datasets as a basis, our models that manage to

correctly classify emails do it with high accuracy ($>94\%$). We work out from this that our hypothesis is reasonable.

In total, the whole dataset is composed of 58,910 emails. Of those, there are 35371 (60.04%) spams and 23539 (39.96%) hams.

5.2.2 Data preprocessing

We began preprocessing by only keeping the subject and content of the email. Other pieces of information of the file header have thus been ignored. For instance, the date at which the email was sent, its sender and receiver are all ignored. We choose to do so because the purpose of this experience is to build machine learning models for spam detection based on text.

Of the content of the email, we then process tags coming from HTML and XML structures. Using the python tool [email.parser](#), we retrieve the text and some meta data information. Meta data is defined as data that provides information about other data. In our case, the meta data are the frames, images and hyperlinks which are transformed in the strings ‘frameStruct’, ‘imageStruct’ and ‘hyperlinkStruct’ respectively. We keep this information because we suspect that images, frames and hyperlinks are very common in emails and could potentially be crucial information in detecting spams.

At this point, the data is composed of a simple text. There are no images coded in it (that said ‘imageStruct’ strings can be found), nor are there tags. However, there still exists what we will call particular structures which cannot be used by machine learning as they are. Of those structures, we identified URLs, phone numbers, email addresses, dates, times and numbers which we converted to the strings ‘urlStruct’, ‘phoneNumberStruct’, ‘emailAddressStruct’, ‘dateStruct’, ‘timeStruct’ and ‘numberStruct’ respectively. To find those structures within a text, we heavily made use of the regular expression (regex) syntax.

Now that we have kept the information we wanted, i.e. the particular structures, we proceed with noise removal (see section 3.1.1): we remove (i) the remaining html tags, (ii) punctuation and (iii) any character that is not a letter or a whitespace. We now normalize our text (see section 3.1.2): we (i) set all characters to lowercase and we (ii) lemmatize all words. In the end, we use word tokenization (see section 3.1.3).

	Word	Count (%)	Spam/Ham ratio
1	numberstruct	47,617 (80.8%)	57.8% / 42.2%
2	to	44,942 (76.3%)	57.7% / 42.3%
3	be	44,857 (76.1%)	57.0% / 43.0%
4	the	43,496 (73.8%)	55.7% / 44.3%
5	and	40,011 (67.9%)	59.5% / 40.5%
6	for	38,725 (65.7%)	57.0% / 43.0%
7	you	36,739 (62.4%)	61.7% / 38.3%
8	urlstruct	36,513 (62.0%)	67.7% / 32.3%
9	of	35,527 (60.3%)	57.1% / 42.9%
10	in	32,782 (55.6%)	56.2% / 43.8%

Table 5.1: 10 most common words in the data. In the first column appears the k -th most common word w . The second column gives the number of emails in which w appears (and the corresponding percentage over all emails). The third column shows the percentage of spam/ham emails for the emails in which w appears.

5.2.3 Data exploration and visualisation

Appendix A gathers the results we will discuss here. They serve to justify the preprocessing choices made above as well as to look for any abnormalities in the data.

Table 5.1 shows that special structures and metadata structures discussed in the previous sections are extremely common in emails. Indeed, the strings ‘numberstruct’ and ‘urlstruct’ which represent special structures are respectively the 1st and 8th most common words/tokens in the data. The ‘hyperlinkstruct’ string, which represents a form of metadata is the 16th most common word in the data (as can be seen in table A.1). In particular, it is interesting to see that ‘hyperlinkstruct’ is heavily connoted with spams, which is an important source of knowledge for spam detection.

Table 5.4 shows the 10 most common spam words and ham words. We define spam (resp. ham) words as words for which spam (resp. ham) emails represent more than 90% of the emails in which they appear. Once again we see that hyperlinks and particular structures such as dates and images are important factors in determining the class of an email. A disturbing fact is the presence of the word ‘enron’ as being the second most popular ham word. Remembering that the emails partly come from the enron mail archive, we can see that this word is a clear bias in the emails. For the rest of the experiment, we will remove this word from all texts.

k	Most common spam words			Most common ham words		
	Spam words	Spam count (%)	Ham count (%)	Ham words	Spam count (%)	Ham count (%)
1	hyperlinkstruct	25,355 (98.5%)	385 (1.5%)	datestruct	580 (6.9%)	7,771 (93.1%)
2	imagestruct	9,922 (96.8%)	332 (3.2%)	enron	1 (0.0%)	7,369 (100.0%)
3	thousand	3,112 (93.1%)	230 (6.9%)	cc	136 (2.3%)	5,858 (97.7%)
4	advertisement	2,936 (97.7%)	70 (2.3%)	pm	561 (8.9%)	5,715 (91.1%)
5	solicitation	2,799 (97.0%)	88 (3.0%)	schedule	239 (6.0%)	3,760 (94.0%)
6	prescription	2,370 (98.8%)	30 (1.2%)	ect	224 (6.1%)	3,436 (93.9%)
7	viagra	2,221 (99.7%)	6 (0.3%)	hou	150 (4.5%)	3,204 (95.5%)
8	xp	1,967 (93.0%)	148 (7.0%)	vince	56 (1.9%)	2,834 (98.1%)
9	hundred	1,920 (90.4%)	204 (9.6%)	houston	86 (3.4%)	2,470 (96.6%)
10	shipping	1,806 (93.0%)	135 (7.0%)	kaminski	0 (0.0%)	1,855 (100.0%)

Table 5.2: 10 most common spam words and ham words. Spam (resp. ham) words are words heavily present in spam (resp. ham) emails, that is 90% of the emails in which the spam (resp. ham) word appears must be spam (resp. ham) emails.

“re instant quote now you can have hundred of lender compete for your loan refinance new home loan debt consolidation debt consultation auto loan credit card student loan second mortgage home equity dear homeowner interest rate be at their low point in numberstruct year we help you find the best rate for your situation by match your need with hundred of lender home improvement refinance second mortgage home equity loan and more even with less than perfect credit this service be numberstruct free to home owner and new home buyer without any obligation just fill out a quick simple form and jump start your future plan today hyperlinkstruct please know that we do not want to send you information regard our special offer if you do not wish to receive it if you would no longer like u to contact you or feel that you have receive this email in error please hyperlinkstruct click here to unsubscribe”

Figure 5.2: Example spam email after preprocessing.

In its raw form, the content of an email is filled with tags. One has to dig the textual information, navigating through the html tree. Figure 5.2 shows an email but after preprocessing. After preprocessing, the email shortens and seems to contain most of the relevant information from its raw form. One can compare this spam email to its raw form with figure A.1.

5.2.4 Model building

Evaluating models is not done without a plan. First, we have to ask ourselves on what we evaluate models. To answer this question, we divide our data into two parts. On one hand, we have the training data composed of 80% of the data which will be used to train our models. On the other hand, we have the test data composed of the remaining 20 % of the data on which we will test the data.

In a second place, one has to compare models with a metric. The metric we choose is simply the accuracy. This metric is very intuitive and pertinent when classes are balanced. Another very popular choice of metric is the F1-score. This metric is especially important when classes are heavily imbalanced. In our case, the classes are rather balanced. Indeed, spam emails amount for 60% of the data whereas ham amount for 40% of the data. As a consequence, the choice of the metric was rather straightforward.

Building models includes fitting the right hyperparameters. To do so, we use cross-validation. The method of cross-validation we use is different from the others in the case of convolution neural networks. Convolution neural networks train a lot slower than our other models. As a consequence, we use a different method of cross-validation. In the next paragraphs, we present (i) our cross-validation approach for the bag of words models, (ii) our cross-validation approach for the convolution neural networks and (iii) our cross-validation results.

Cross-validation for our bag of words models. At training time, to choose between different possible hyperparameters, we do K-fold cross-validation described in section 2.1.2 with $K = 5$. To do so, we use the tool [GridSearchCV](#) from scikit-learn.

For each of the learning algorithms described in section 2.3, we present the hyperparameters that we consider during K-fold crossvalidation:

- Multinomial naive Bayes classifier (see section 2.3.1). This model has only the one hyperparameter α . We use the tool [MultinomialNB](#) from scikit-learn.
- Logistic regression (see section 2.3.2). The two hyperparameters of this model are C , the inverse of the regularization strength and the type of regularization (l_1 , l_2 or Elastic-Net). We use the tool [LogisticRegression](#) from scikit-learn.
- Random forest (see section 2.3.3). The first hyperparameter we consider is B , the number of decision trees in the random forest. The second hyperparameter is the minimum number of samples required before splitting a node. We use the tool [RandomForestClassifier](#) from scikit-learn.
- K-nearest neighbors (see section 2.3.4). There are two hyperparameters for this method. The first hyperparameter is k , the number of neighbors. The second hyperparameter is whether or not we do weighted KNN. We use the tool [KNeighborsClassifier](#) from scikit-learn.

- Linear support vector machine (see section 2.3.5). The only hyperparameter considered is C , the inverse of the regularization strength. The tool used is [LinearSVC](#) from scikit-learn.

Cross-validation for our convolution neural networks. Convolution neural networks (see section 2.3.6) are considered on their own. It is unrealistic to train convolution neural networks on many different hyperparameters as training such a model is much more expensive than training any of the previous ones (it took 6 hours to train a single model). For the same reason, we do not do K-fold cross-validation but the Holdout method. We construct the validation set by drawing 20% of the training set. We will propose four model architectures for CNN and pick the best one based on their accuracy on the validation data. We consider the embedding dimension values $d = 50, 300$. We also consider models with a single or two fully connected layers. Models are trained with the python deep learning API [Keras](#).

In convolution neural networks, the input size is $L \times d$ where L is the length of the longest tokenized email in the training set. Figure 5.3 shows that there are very few emails of large size. We set $L = 4000$ (instead of the otherwise $L = 39315$) as there are only 112 documents of size greater than 4000. By doing so, we effectively reduce the input size by a factor of 10.

Cross-validation results for all models. We give here the best hyperparameters of our models. In appendix B, we provide more details on the range of hyperparameters considered.

The multinomial naive Bayes classifier showcases the best accuracy with $\alpha = 0.002$. The lower the value of α is, the less smoothed the probabilities are. One can expect the effect of spam and ham words, i.e. words heavily connoted to spam or ham emails, to be magnified. More details are given in figure B.1.

Random forest performs better the higher the number of decision trees is. However, after 200, the increase in accuracy is very little. For this reason, we did not consider even higher number of decision trees. We can also see that limiting the depth of decision trees by introducing a minimum number of samples before splitting a node did not improve the quality of prediction. More details are given in table B.1.

Linear SVM performs best for $C = 0.068129$. More details are available in figure B.2.

There are no big performance variations among the CNN models we considered. The

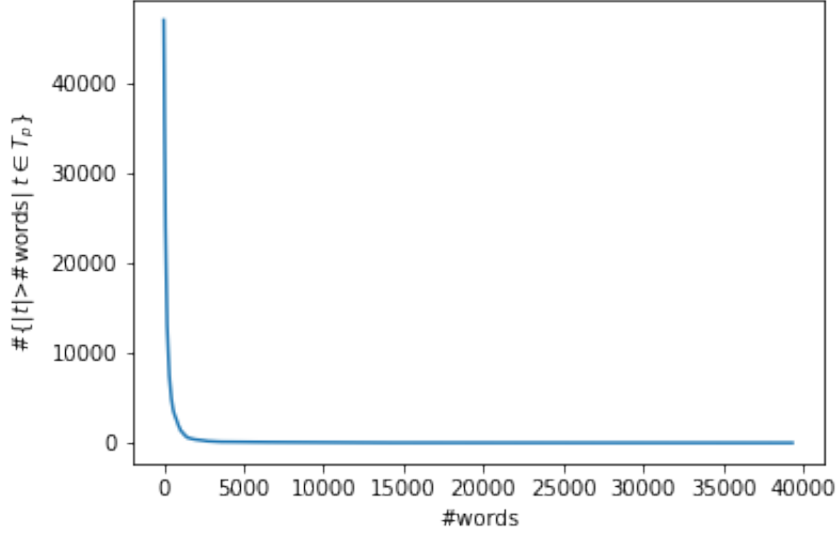


Figure 5.3: Number of texts of size greater than some x . We only consider texts belonging to the training set after preprocessing T_p .

best model has $d = 300$ and 2 fully connected layers. We present the architecture of this model in figures 5.4 and 5.5. One can notice the presence of dropout layers. Those layers are introduced in order to prevent overfitting during training. A dropout layer of rate r works by setting each entry of its input vector to 0 with a probability of r . The dropout layer is only activated during training. More details can be found in table B.2.

For the k-nearest neighbor classifier, setting $k = 1$ allows for the best performance. Increasing k makes the accuracy drop rapidly. In general, weighted KNN seems to perform better than regular KNN. However, this is irrelevant for $k = 1$ and there are no vote, weighted or not, when only one entity makes a decision. More details can be found in figure B.3.

With logistic regression, the best choice of hyperparameters are l_2 for the regularization norm and $C = 9$. More details are given in figure B.4.

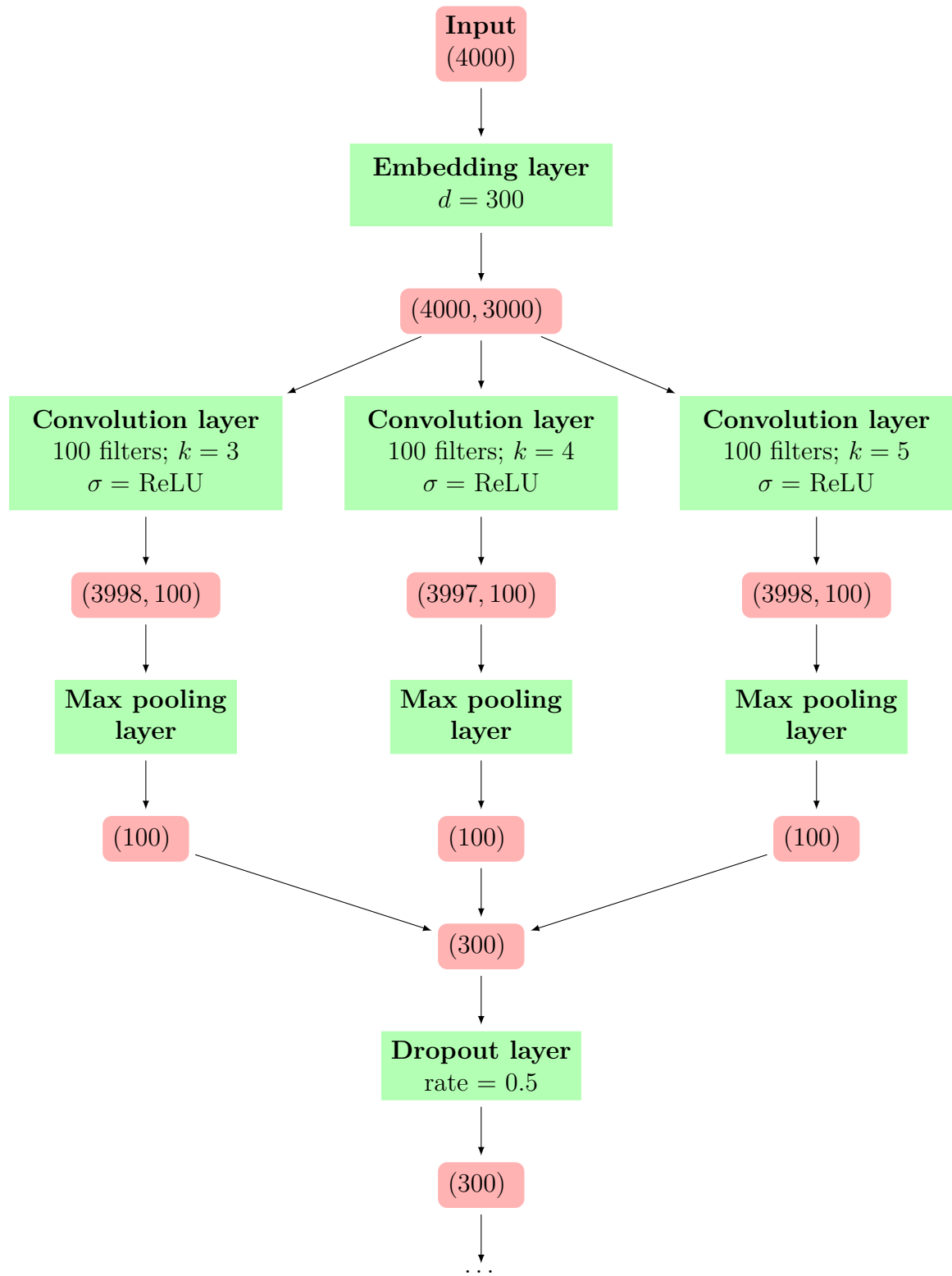


Figure 5.4: First part of our architecture of the Convolution Neural Network. In this part, text is first transformed by the embedding layer before being processed by the convolution layers (with activation function $\sigma = \text{ReLU}$). The outputs of the convolution layers then go through a max pooling layer, effectively resulting in a feature vector of size 300. At training time, those feature vectors are fed to a dropout layer.

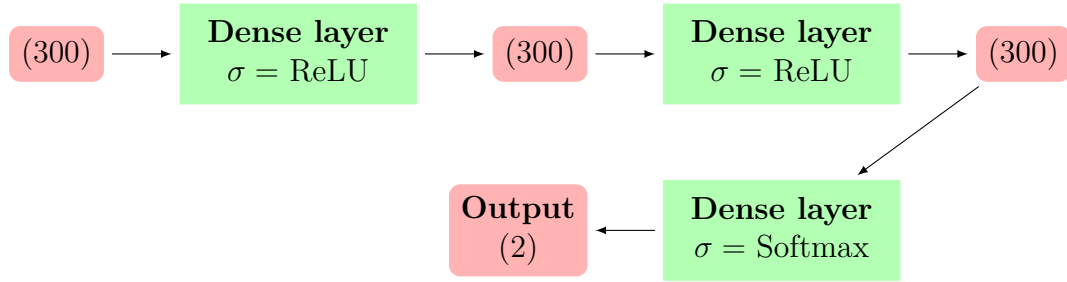


Figure 5.5: *Second part of our architecture of the Convolution Neural Network. In the first part, text was transformed into a feature vector of size 300. In this part, this feature vector goes through two dense layers with ReLU activation function and a single layer with the softmax activation function. This corresponds to a multi-layer perceptron with two hidden layers.*

Models	accuracy (%)	f1 score (%)	TP	TN	FP	FN
MultinomialNB	98.09	98.41	6961	4596	123	102
LinearSVC	98.70	98.92	7007	4622	97	56
RandomForestClassifier	99.22	99.35	7014	4676	43	49
KNeighborsClassifier	94.38	95.42	6902	4218	501	161
LogisticRegression	77.72	83.97	6877	2280	2439	186
CNN	97.15	97.65	6978	4468	251	85

Table 5.3: *Comparison of all models on the test set.*

5.2.5 Model evaluation

We compared the models over the test set. The results are shown in Table 5.3. The random forest classifier performs best with very high accuracy of 99.2%. The linear SVM, multinomial naive Bayes classifier, and CNN model come very close behind with 98.70%, 98.07% and 97.15% of accuracy respectively. The KNN model performs also with very high accuracy (94.38%).

However, the logistic regression model performs surprisingly badly with an accuracy of 77.72%, but more importantly 2439 false positive for 2280 true negatives. This translates to an accuracy of 48.31% for ham emails, that is less than random guess. Considering those results, we decided to discard the logistic regression model and not to use it in chapter 6.

It is interesting to see models such as random forest performing so well. This can be explained by the presence of “spam words”, i.e. words heavily connoted to the spam class. When one looks at table 5.4, one can see that the presence of words such as *thousand*, *advertisement* or even meta data such as *hyperlinks* and *images* are very informative on the classification of words. Those sort of relations are easy

k	Most common spam words			Most common ham words		
	Spam words	Spam count (%)	Ham count (%)	Ham words	Spam count (%)	Ham count (%)
1	hyperlinkstruct	25,355 (98.5%)	385 (1.5%)	datestruct	580 (6.9%)	7,771 (93.1%)
2	imagestruct	9,922 (96.8%)	332 (3.2%)	enron	1 (0.0%)	7,369 (100.0%)
3	thousand	3,112 (93.1%)	230 (6.9%)	cc	136 (2.3%)	5,858 (97.7%)
4	advertisement	2,936 (97.7%)	70 (2.3%)	pm	561 (8.9%)	5,715 (91.1%)
5	solicitation	2,799 (97.0%)	88 (3.0%)	schedule	239 (6.0%)	3,760 (94.0%)
6	prescription	2,370 (98.8%)	30 (1.2%)	ect	224 (6.1%)	3,436 (93.9%)
7	viagra	2,221 (99.7%)	6 (0.3%)	hou	150 (4.5%)	3,204 (95.5%)
8	xp	1,967 (93.0%)	148 (7.0%)	vince	56 (1.9%)	2,834 (98.1%)
9	hundred	1,920 (90.4%)	204 (9.6%)	houston	86 (3.4%)	2,470 (96.6%)
10	shipping	1,806 (93.0%)	135 (7.0%)	kaminski	0 (0.0%)	1,855 (100.0%)

Table 5.4: 10 most common spam words and ham words. Spam (resp. ham) words are words heavily present in spam (resp. ham) emails, that is 90% of the emails in which the spam (resp. ham) word appears must be spam (resp. ham) emails.

to reproduce for bag of words models. Word embedding models such as CNN are designed to perform well when the context of words is very important.

Intuitively, we argue that particular structures such as *hyperlinks* are important in the sense that they carry important information about emails. To ascertain our claim, we made another experiment, ignoring all particular structures this time. Table 5.5 regroups our results, showing the accuracy of models both with and without those particular structures. The logistic regression model is the only case of decrease in error rate. However, it still performs worse than all our other models with an accuracy under 90%. All other models perform worse without particular structures. For those models, the increase in error rate is always superior to 10%. On average, the increase in error rate for all models excluding logistic regression is 50.07%.

Models	Accuracy (%) with P.S.	Accuracy (%) without P.S.	Error increase (%)
MultinomialNB	98.09	97.95	+13.26
LinearSVC	98.70	98.31	+30.00
RandomForestClassifier	99.22	98.56	+84.62
KNeighborsClassifier	94.38	93.02	+24.20
LogisticRegression	77.72	89.09	-51.48
CNN	97.15	94.35	+98.25

Table 5.5: Comparison of models with and without use of particular structures (P.S.).

5.3 Conclusion

In this chapter, we wanted to know if we could build models that detect email spams reliably.

In a first section, we briefly first presented what spams are. We settled on the notion that, in the case of email spam detection, spams are undesirable emails.

In the second section, we put the theory presented in chapter 3 in practice on a concrete case. We mentioned that classifying an email as spam or ham is difficult as the answer may depend on the user. We made the hypothesis that in most cases, the answer is common to the users. Then, we collected our data from two sources: the Enron email archive and the Apache Spam Assassin dataset. Exploring our data a first time, we observed the presence of particular structures such as *hyperlinks*, *images* and *urls*. We argued that they contained central information on emails and kept them during the preprocessing of our emails. After that, we preprocessed our data and analyzed the newly preprocessed data. By doing so, we found some bias present with the word ‘enron’. We continued and built the variety of machine learning algorithms. We chose the best hyperparameters on a validation set through K-fold cross-validation or the Hold-out method. We compared the best models on the test set. What resulted from this comparison was that the random forest classifier performed best, closely followed by the linear SVM, the multinomial naive Bayes classifier and the convolution neural network. Our logistic regression model performed badly.

To verify our claim that particular structures are central information on emails in the case of spam filtering, we trained our models again, this time removing all particular structures from the text. The results showed an average increase of 50.07% in the error rate for all but the logistic regression model. For the previous mentioned reason, we can ignore the results for the logistic regression model.

In light of the results obtained this chapter, we first conclude that our preprocessed data (i) do not contain obvious bias anymore, (ii) is of good quality and (iii) can be used to train our machine learning models. In a second step, we conclude that (i) the particular structures introduced in section 5.2.2 contain precious information as predicted and that (ii) our models perform better when taking into account these structures. Thirdly, we conclude with conviction that (i) our models detect reliably email spams and that (ii) these models can be used in the next chapter “Adversarial examples”. The only exception we make is our logistic regression model, which performs badly. For this model, our conclusion is that it cannot be used to perform spam detection reliably. As a consequence, we won’t use it in chapter 6.

Chapter 6

Adversarial examples

In this chapter, we study the impact of adversarial examples in our particular setting that is email spam detection. In the first place, we ask ourselves an essential question: can we craft adversarial examples to fool machine learning models? Only then can we ask the main question of this thesis: do previously crafted adversarial examples transfer across models? For this purpose, one must have (i) models on which to launch attacks, (ii) email spams that can be transformed into adversarial examples and (iii) attack methods to craft the adversarial examples.

Chapter 5 was devoted to building models that detect email spams reliably. At the end of this chapter, we concluded that, of six architectures considered, five of them gave birth to models that reliably detect email spams: (i) multinomial naive Bayes classifier, (ii) random forest, (iii) k-nearest neighbors, (iv) linear support vector machine and (v) convolution neural networks. Thus, the first requirement is met.

Before training the above mentioned models, we partitioned our dataset into a training set and a test set (see section 5.2.4). We can and will use the spam emails among the test set as our dataset for this chapter. Therefore, the second requirement is met.

Section 6.1 discusses our methodology for crafting adversarial examples. In particular, this section serves (i) to introduce new constraints that handle the novelty brought by the particular structures introduced in section 5.2.2, (ii) to specify the attack methods we will use to craft adversarial examples and (iii) to explain the details of our experiments for answering the questions. With this section, the third requirement is met and a strategy is given to tackle the chapter's questions.

Now that all our requirements are met and that we devised a methodology to answer our two questions, section 6.2 presents our results. We present our results in three parts. Part 6.2.1 showcases the success rates of our attacks. This serves to answer our first question of this chapter. Part 6.2.2 focuses on exploring the obtained adversarial examples. It serves to verify that our adversarial examples respect the constraints we imposed on them and, more generally, their quality. Part 6.2.3 shows our results for the main question of the thesis: do the adversarial examples crafted on one model transfer to all our other models?

Section 6.3 discusses the results above. In particular, we give our answers to this chapter’s two questions. We include in this discussion the various factors that may have an impact on our results.

Finally, section 6.4 concludes the chapter.

6.1 Methodology

The motivation of this section is two-fold. The first motive is to present the two attacks we use to craft adversarial examples. To take into account the particularities of our setting, we adapt these attacks by modifying the set of constraints. In particular, we introduce new constraints to handle the particular structures introduced in section 5.2.2. The attacks are discussed in section 6.1.2 and the new constraints are introduced in section 6.1.1.

The second motive is to present the details of the experiments we made to answer the two questions of this chapter, now that we have fulfilled their requirements. We provide those details in section 6.1.3.

6.1.1 Constraints imposed

When transforming a text into an adversarial example, not all transformations are valid. In particular, the transformations we apply on email spams must not transform them into genuine hams. For this purpose, we will impose three important constraints. For more information on constraints, we redirect the reader to section 4.1.2.

The first constraint makes sure that the words of an email are replaced by words of similar meaning, so as to limit the impact of our transformation on the true

Origin	Transformations
'framestruct'	/
'imagestruct'	'urlstruct', 'hyperlinkstruct'
'hyperlinkstruct'	'urlstruct'
'urlstruct'	'hyperlinkstruct'
'phonenumberstruct'	'emailaddressstruct'
'emailaddressstruct'	'phonenumberstruct', 'urlstruct', 'hyperlinkstruct'
'datestruct'	/
'timestruct'	/
'numberstruct'	/

Table 6.1: *Possible transformations for particular transformations.*

meaning of the email. Thus it is a semantic constraint. For this, we make use of word embedding (section 3.1.4.2). The idea is that words whose vectors are close in the embedding space are expected to have similar meanings. We usually measure this similarity through the cosine similarity.

The second constraint concerns the particular structures introduced in section 5.2.2. These particular structures represent some meta data such as hyperlinks, images, urls and others. One cannot replace meta data blindly by regular words as this would result in a loss of information. Some email spams have commercial goals. They try to sell some product, e.g. medicine, from their own website. Removing the hyperlink linking to it would result in the email spam to lose much of its meaning. We restrain the transformations of those structures according to table 6.1. This is a type of grammar constraint.

The third constraint is closely linked to the previous one. This constraints states that particular structures cannot be removed from a text for the same reason as the second constraint: it would result in a too big loss of information. This is a type of pre-transformation constraint.

When deploying attacks, we also consider other constraints such as a maximum number of words perturbed. However, the word embedding and particular structures constraints are always present.

6.1.2 Implementing the attacks

In this section, we discuss the difficulties encountered and choices made during the implementation of the attacks in section 4.3.

The library TextAttack gives a solid base on which one can build attacks. For our purposes however, we need to create a number of complementary modules. Only then can we apply the attacks mentioned above. These modules contain:

- Two new constraints. The first new constraint makes sure that we do not delete words that represent particular structures such as hyperlinks. The second new constraint impose that any particular structure must be replaced by another compatible particular structure. These two constraints are very strong as we saw in section 5.2.3. This is even more important since the particular structures such as hyperlinks and images are heavily connoted to spam emails. They, however, make sure that we do not lose vital information.
- A new word embedding class. TextAttack has a class named *WordEmbedding* that compares the proximity of words in the embedding space at running time. We adapted the class *WordEmbedding* to handle the special case of particular structures. For example, the word ‘hyperlinkstruct’ would be seen as closer to ‘urlstruct’ than any other word.
- A wrapper class for each model. TextAttack uses these wrappers to call models under the same method during attacks. During attacks, TextAttack often asks estimated probabilities measures for inputs instead of simply their predicted classes. This allows to compare the quality of transformations. We provided those probabilities for each of our models as follows:
 - The multinomial naive Bayes classifier computes probabilities before classifying an input. We use those probabilities as our predicted probabilities.
 - For the linear SVM, we can compute a confidence measure as proportional to the distance to the hyperplane. Through a logistic function, one can transform the confidence measure into probabilities.
 - For the random forest model, it is computed as the average of confidence of each tree. For some input x and class y , the predicted probability for a single tree is the fraction of samples of class y in the leaf node.
 - For KNN, the predicted probability for class y is given by the number of neighbors labeled as y among the k nearest.

- For CNN, those predicted probabilities are found at the last layer of the model.

With those modules in hand, we first adapt the attacks to take into account our new constraints. We do this by simply providing the new constraints to the list of already present constraints.

We then introduce our new word embedding to the attacks. In particular, in the constraint ‘WordEmbeddingDistance’ that imposes a minimal distance between words in the case of a substitution and in the ‘WordSwapEmbedding’ transformation which substitutes words for the nearest words in the word embedding space.

We also modify a bit the *hot-flip* attack. The constraint “the maximum number of words perturbed is 2” is replaced by the constraint “The adversarial examples need to share a similarity of 84% with the original spam”. In addition, the threshold for cosine similarity in word embedding space between a word in text and its substitute is set to 0.5 instead of 0.8. These modifications are made so that the constraints for the *text fooler* attack and the *hot-flip* attack are more similar. Otherwise, the *hot-flip* attack would be unfairly more constrained than the text fooler attack.

With all those changes, the *text fooler* attack can be launched. There is still one issue with regard to the *hot-flip* attack. The *hot-flip* attack makes use of gradients given by the function ‘get_grad’ through the transformation ‘WordSwapGradientBased’. Due to details of implementation, this transformation has been implemented to only handle pytorch modules. The transformation uses more than the ‘get_grad’ function from the wrapper. It also uses functions of the model in the wrapper that are specific to the pytorch library. Skicit-learn models are also unusable because the transformation requires for a word embedding representation of text.

Despite all those difficulties, we can still use our convolution neural networks model from Keras by first re-implementing it into Pytorch.

6.1.3 Experiment details

Our experiments aim to respond to the two questions of this chapter. The first is: can we craft adversarial examples to fool our machine learning models? The second is: do our adversarial examples transfer across models?

To help in answering the questions, we have: (i) five models that detect email spams reliably (see section 5.3), (ii) a set of email spams taken from the test set

		Models attacked				
		CNN	KNN	RF	SVM	MNB
Attack recipes	Text fooler	24%	4%	34%	70%	16%
	Text fooler*	57%	5%	67%	95%	26%
	Hot flip	2%	/	/	/	/
	Hot flip*	16%	/	/	/	/

Table 6.2: *Percentage of success of different attack recipes on a variety of models. The star (*) informs that the model is given the option of deletion.*

(see section 5.2.4) and (iii) two different types of attacks (see section 6.1.2).

To answer the first question, we need to craft adversarial examples. We consider each attack individually. For the *text fooler* attack, we attack each of our five models individually. This gives us a new set of adversarial examples for each of our models. For the hot-flip attack, we only attack the CNN model. All adversarial examples are transformations of the same email spams from the test set.

To answer the second question, we evaluate the accuracy of our models on all sets of adversarial examples.

6.2 Results

We present our results in three parts. The first part focuses on the success rates of our attacks. The second part explores the composition of the produced adversarial examples. The third part gives the results concerning the transferability of our adversarial examples, i.e. capacity of adversarial examples to fool other models.

6.2.1 Building adversarial examples

Table 6.2 gives the rates of success for each attack on all models. The low success rate for attacks on KNN are surprising. It is, however, explained by the optimal value of its hyperparameter $k = 1$. This value makes the predicted probability given by the model always 0 or 1. *Text fooler* has then very few information to use when building adversarial examples, hence the results.

Hot-flip performs surprisingly worse than *text fooler*. One would expect from

<i>Text fooler</i>	CNN	KNN	RF	SVM	MNB
No deletion	6.6	1.0	33.5	15.6	9.4
With deletion	11.7	1.0	34.5	17.1	9.7

Table 6.3: *Number of transformations on average when a Text fooler attack is successful.*

a white-box attack to outperform a black-box attack. We attribute this to our particular structures. Indeed, the *hot-flip* attack relies on the gradient of a model’s loss function for the substitution of words. This gradient does not know of particular structures and, as a result, does not perform to the expectations.

From our observations, we infer that the *text fooler* attack on our KNN model and the *hot-flip* attack on our CNN model did not succeed. As such, we will not consider the transferability of their respective sets of adversarial examples.

6.2.2 Composition of adversarial examples

Figure 6.1 gives an insight into the types of transformations one can expect from the *text fooler* attacks. In most cases, the adversarial examples were built with few modifications. Random forest and SVM need the most transformations, that is 8 and 7 respectively. Interestingly enough, adversarial examples generated from different models are very different from each other. The most similar transformations are between the random forest classifier and SVM. Even then, the word substitutes are different.

On average, the number of transformations is higher than shown in figure 6.1. Table 6.3 gives the average number of transformations one would expect to create an adversarial examples.

6.2.3 Transferability of adversarial examples

From section 6.2.1 we have that the only successful attacks were the *text fooler* attacks on four of our models: (i) convolution neural networks – CNN, (ii) random forest – RF, (iii) linear support vector machine – SVM, and (iv) multinomial naive Bayes – MNB. Moreover, allowing deletion gives clearly better adversarial examples, so that we will only consider the *text fooler* attacker with deletion. We observe here whether or not the corresponding adversarial examples transfer to all our models.

CNN: get software cd and download under numberstruct numberstruct get [cd→demo] and downloads all software under numberstruct numberstruct urlstruct confound those who have say our remark before whom [be→would] you he ask for he have [attend→visit] business college

CNN*: get software cd and download under numberstruct numberstruct get **cd** and downloads all software under numberstruct numberstruct urlstruct confound those who have say our remark before whom be you **he** ask for he have attend business college

KNN: [get→come] software cd and download under numberstruct numberstruct get cd and downloads all software under numberstruct numberstruct urlstruct confound those who have say our remark before whom be you he ask for he have attend business college

KNN*: [get→come] software cd and download under numberstruct numberstruct get cd and downloads all software under numberstruct numberstruct urlstruct confound those who have say our remark before whom be you he ask for he have attend business college

RF: [get→know] [software→technology] [cd→audio] and [download→internet] under numberstruct numberstruct [get→come] cd and downloads all [software→user] under numberstruct numberstruct urlstruct confound those who have say our remark before whom [be→should] you he ask for he have attend business [college→university]

RF*: [get→know] [software→technology] cd and [download→internet] under numberstruct numberstruct [get→go] cd and downloads all [software→user] under numberstruct numberstruct urlstruct confound those **who** have say **our** remark before whom be you he ask for he have attend business college

SVM: [get→go] [software→computer] [cd→album] and download under numberstruct numberstruct [get→know] [cd→compilation] and downloads all [software→user] under numberstruct numberstruct urlstruct confound those who have say our remark before whom be you he ask for he have [attend→participate] business college

SVM*: get **software** **cd** and download **under** numberstruct numberstruct get [cd→demo] and downloads all [software→user] **under** numberstruct numberstruct urlstruct confound those who have say **our** remark **before** whom be **you** he ask for he have attend **business** college

MNB: get software [cd→compilation] and download under numberstruct numberstruct get [cd→compilation] and downloads all software under numberstruct numberstruct urlstruct confound those who have say our remark before whom be you he ask for he have attend business college

MNB*: get software [cd→compilation] and download under numberstruct numberstruct get [cd→compilation] and downloads all software under numberstruct numberstruct urlstruct confound those who have say our remark before whom be you he ask for he have attend business college

Figure 6.1: *Adversarial examples generated with the Text fooler attack. In bold are given the initials of each model. The star (*) indicates that the attack includes deletion on top of substitution. In red are deleted words and in green are added words. [x→y] indicates that word x was replace by word y.*

		Defense models				
		CNN	KNN	RF	SVM	MNB
Attack models	CNN	57	4	1	4	2
	RF	12	6	67	14	5
	SVM	10	7	3	95	4
	MNB	2	2	3	4	26

Table 6.4: *Transferability results for adversarial examples generated from the text fooler attack.*

		Defense models				
		CNN	KNN	RF	SVM	MNB
Attack models	CNN	57	5	4	6	4
	RF	18	10	67	41	16
	SVM	18	13	23	95	16
	MNB	4	2	3	5	26

Table 6.5: *Transferability results for adversarial examples generated from the text fooler⁺ attack.*

In this section, we consider two different cases. The first case is the one in which the search for an adversarial example is stopped when it successfully fools the attacked model. The second case is the one in which the search for an adversarial example is only stopped when either (i) the attacked model is more than 90% confident in his wrong guess or (ii) the texts can no longer be modified due to constraints. We will refer to the first case as *text fooler* and the second case as *text fooler⁺*.

Transferability results are given in table 6.4 for *text fooler* and in table 6.5 for *text fooler⁺*. Table 6.6 compares the two by giving the percentages of adversarial examples that transfer on average. We see that for *text fooler*, the adversarial examples do not transfer well. For the *text fooler⁺* attack, however, results are more positive. In particular, when considering the adversarial examples related to the random forest and linear support vector models, the success rates on the other models is $> 10\%$.

6.3 Discussion

We constrained attacks to not delete particular structures and always replace them by other compatible particular structures to avoid loss of information. Indeed,

	Attack models			
	CNN	RF	SVM	MNB
Text fooler	4.8%	13.8%	6.3%	10.6%
Text fooler ⁺	8.3%	31.7%	18.4%	13.5%

Table 6.6: *Percentage of transfer success. Attack models refers to the models on which were crafted the adversarial examples.*

we think that *hyperlinks*, *images* and others contain precious information. We think that removing that information would perturb the original text too much. This is a point of discussion, however, and some adversaries could allow such transformations.

Of all our attacks, three stood out with a success rate greater than 50%. They are the *text fooler* attack with deletion for three of our models: (i) convolution neural networks, (ii) random forest and (iii) linear support vector machine. The *text fooler* attack (with deletion) on our multinomial naive Bayes model was successful to a lesser extent with 26% success. We failed our attacks on our k-nearest neighbor model. We attribute this result to the value of its hyperparameter $k = 1$. During the *text fooler* attack, the transformations are ranked according to the drop in confidence of the model. For the k-nearest neighbor model with $k = 1$, its confidence in its prediction is always 0 or 1, rendering the *text fooler* attack ineffective. The *hot-flip* attack has a low success rate. We attribute that to the fact that the *hot-flip* attack relies on a gradient, which in our implementation, cannot handle well particular structures.

Considering the strong constraints we imposed on adversarial examples, fooling three models more than 50% of the time can be considered quite a good result.

Figure 6.1 gives a good idea of what adversarial examples are. Emails seem to conserve their meaning. It is true, however, that some adversarial examples have awkward looking bits of phrases. For example, the adversarial example for random forest without deletion begins with ‘Know technology audio’ instead of ‘Get software cd’. All in all, however, we claim that our adversarial examples are of good quality and can indeed be used for our experiments.

In light of figure 6.6, it seems that our adversarial examples do transfer across models to some extent. Articles show a higher transferability of adversarial examples [32, 35]. Those articles, however, showcase transferability results for text classification using n-grams (see section 3.1.3 for more information on n-grams). In this thesis, we decided to chose word tokenization. In the case of n-grams, it is possible to

change a single character in a text whereas our implementation does not allow it. Overall, the range of transformation available to n-grams is superior to the case of word tokenization. Moreover, the introduction of particular structures is another difference with those articles. For all these reasons, our findings are not so surprising.

Our work brings to light that the fact that adversarial examples transfer across models is not absolute. With our strong constraints and word tokenization, we observed that adversarial examples do transfer, but not to the extent of the literature.

The first question of this chapter was: can we craft adversarial examples to fool machine learning models? We answer this question by the affirmative. For three of our models, we managed a success rate superior to 50%. The fourth one was fooled to a lesser degree (26%). Our last model, k-nearest neighbors, could not really be attacked up front. However, using the transferability of adversarial examples, we could fool it up to 13% of the time.

The second and main question of this chapter was: do our previously crafted adversarial examples transfer across models? Our answer is: in our context, adversarial examples do transfer albeit to a lesser extent than what is shown in the literature.

6.4 Conclusion

In chapter 5, we built models to perform email spam detection. Five of our models did it reliably: (i) multinomial naive Bayes classifier, (ii) random forest, (iii) k-nearest neighbors, (iv) linear support vector machine and (v) convolution neural networks.

In this chapter, we had two goals: (i) fool each of these models with adversarial examples and (ii) transfer the adversarial examples that successfully fooled a model to fool other models.

In section 6.1, we proposed experiments to answer the two questions. In particular, we used two attacks from the literature: text fooler (see section 4.3.1), a black box attack, and hot-flip (see section 4.3.2), a white box attack. We modified the attacks a little in order to take into account our setting of email spam detection with particular structures. This mostly translated into new constraints.

In section 6.2, we presented the results of our experiments in three parts: (i) results of the previously mentioned attacks on our models, (ii) composition of the resulting adversarial examples and (iii) transferability results for these adversarial examples. Section 6.3 discussed those results. Our conclusion on this chapter is threefold. Firstly, we managed to fool our models. We had our best results with the text fooler attack when including deletion. We fooled with great success three of our models: convolution neural networks, random forest and linear support vector machine. We fooled but to a lesser extent our multinomial naive Bayes model. We did not manage to fool our k-nearest neighbors as easily but managed to fool it up to 13% of the time through transferability. The hot-flip attack performed badly. We attributed this result to the fact that the hot-flip attack uses the gradient. In our implementation, however, the gradient does not take into account particular structures, resulting in a bad performance. Secondly, our adversarial examples are of good quality. Thirdly, our adversarial examples do transfer across models albeit to a lesser extent than presented in the literature. We justify this result by the fact that, in the literature, transferability for text classification is mostly done with n-grams, whereas we use word tokenization. Another difference is the presence of particular structures in our emails.

Chapter 7

Conclusion

Machine learning techniques are nowadays used widely. However machine learning models are naturally the target of attacks. In fact, the main idea behind so-called adversarial machine learning is to produce faulty results by manipulating trained learning models. In that context, the central question of this thesis is “Are adversarial examples transferable from neural networks structures to more classical machine learning models in the context of email spam detection?” In order to answer this question, we identified two supplementary questions. Firstly, can we build models that detect email spam reliably? Secondly, can we fool these models using adversarial examples? To answer these questions, we structured our work in 6 chapters.

After an introduction in chapter 1, we defined in chapter 2 machine learning, adversarial machine learning and how the adversarial examples relate to them. In chapter 3 we discussed text classification, from preprocessing all the way to learning algorithms. We finished discussing background material in chapter 4 with adversarial examples in text classification.

In chapter 5, we analyzed email spam detection using the Enron email archive and the Apache Spam Assassin dataset. We observed that information other than text is relevant for the classification of spams. Among others, we saw that the presence of *hyperlinks* in emails was highly correlated to spams. We subsequently verified our claim by showing that our models performed worse when ignoring this information. We built our machine learning models by using (i) a training set to train models parameters, and (ii) a validation set to choose among a grid of hyperparameters. We observed the performance of our models on a test set. The logistic regression model performed badly, having a worse accuracy than random

guess on classifying ham emails. For this reason, we discarded the model. All other models had high accuracy ($> 94\%$) and were kept for chapter 6. At the end of this chapter, we responded to the first supplementary question by the affirmative: we built five models that reliably detect email spams.

In chapter 6, we first established ground rules for adversarial examples. The rules include (i) substituting a word by synonym, (ii) not deleting words corresponding to meta data such as *hyperlinks* and (iii) imposing a high similarity between the original text and the adversarial example.

When launching our attacks, we observed a success rate superior to 50% for three of our models. Attacks on the K-nearest neighbors model were very ineffective, never surpassing 5% in success rate. We attributed this phenomenon to the value of its hyperparameter $k = 1$. Attacks on the multinomial naive Bayes classifier were mitigated with 26% of success rate in the best case.

In an attempt to answer the main question of this thesis, we reach the conclusion that our adversarial examples do transfer across models but not to the extent of the literature. We attributed this result to (i) the use of metadata such as *hyperlinks* and *images* on which we impose strong constraint, as well as (ii) the fact that, in the referenced articles, the n-gram representation of text is used instead of the word tokenization we use.

In this thesis, we could not cover all points. In particular, we could only consider two types of attacks. Discussing the results for other types of attacks present in literature is appealing and certainly deserve attention for future work.

Moreover, in section 4.2, we presented how one can compute the gradients for the multinomial naive Bayes classifier and linear support vector machine. Given these gradients, it would be interesting to see how one can use the hot flip attack on those models, considering that they use the bag of words representation. This certainly deserves future work as well.

Finally, another approach to classifying text is the use of n-grams instead of word tokenization. Investigating the same dataset with this new approach to tokenization is an interesting subject for future work.

Appendix A

Preprocessing

This appendix presents the results for section 5.2.3.

	Word	Count (%)	Spam/Ham ratio
1	numberstruct	47,617 (80.8%)	57.8% / 42.2%
2	to	44,942 (76.3%)	57.7% / 42.3%
3	be	44,857 (76.1%)	57.0% / 43.0%
4	the	43,496 (73.8%)	55.7% / 44.3%
5	and	40,011 (67.9%)	59.5% / 40.5%
6	for	38,725 (65.7%)	57.0% / 43.0%
7	you	36,739 (62.4%)	61.7% / 38.3%
8	urlstruct	36,513 (62.0%)	67.7% / 32.3%
9	of	35,527 (60.3%)	57.1% / 42.9%
10	in	32,782 (55.6%)	56.2% / 43.8%
11	have	30,586 (51.9%)	52.6% / 47.4%
12	this	29,272 (49.7%)	56.0% / 44.0%
13	on	29,038 (49.3%)	49.8% / 50.2%
14	your	28,033 (47.6%)	71.3% / 28.7%
15	with	26,013 (44.2%)	56.3% / 43.7%
16	hyperlinkstruct	25,740 (43.7%)	98.5% / 1.5%
17	it	25,602 (43.5%)	57.7% / 42.3%
18	we	24,914 (42.3%)	62.1% / 37.9%
19	from	24,413 (41.4%)	53.7% / 46.3%
20	that	24,409 (41.4%)	51.7% / 48.3%
21	at	22,020 (37.4%)	53.3% / 46.7%
22	or	21,770 (37.0%)	61.0% / 39.0%
23	not	21,134 (35.9%)	58.6% / 41.4%
24	will	20,819 (35.3%)	51.1% / 48.9%
25	if	20,304 (34.5%)	49.2% / 50.8%
26	by	19,653 (33.4%)	53.3% / 46.7%
27	all	19,349 (32.8%)	65.1% / 34.9%
28	do	19,341 (32.8%)	58.8% / 41.2%
29	can	18,820 (31.9%)	57.6% / 42.4%
30	our	18,661 (31.7%)	69.7% / 30.3%

Table A.1: 30 most common words in the data. In the first column appears the k -th most common word w . The second column gives the number of emails in which w appears (and the corresponding percentage over all emails). The third columns shows the percentage of spam/ham emails for the emails in which w appears.

k	Most common spam words			Most common ham words		
	Spam words	Spam count (%)	Ham count (%)	Ham words	Spam count (%)	Ham count (%)
1	hyperlinkstruct	25,355 (98.5%)	385 (1.5%)	datestruct	580 (6.9%)	7,771 (93.1%)
2	imagestruct	9,922 (96.8%)	332 (3.2%)	enron	1 (0.0%)	7,369 (100.0%)
3	thousand	3,112 (93.1%)	230 (6.9%)	cc	136 (2.3%)	5,858 (97.7%)
4	advertisement	2,936 (97.7%)	70 (2.3%)	pm	561 (8.9%)	5,715 (91.1%)
5	solicitation	2,799 (97.0%)	88 (3.0%)	schedule	239 (6.0%)	3,760 (94.0%)
6	prescription	2,370 (98.8%)	30 (1.2%)	ect	224 (6.1%)	3,436 (93.9%)
7	viagra	2,221 (99.7%)	6 (0.3%)	hou	150 (4.5%)	3,204 (95.5%)
8	xp	1,967 (93.0%)	148 (7.0%)	vince	56 (1.9%)	2,834 (98.1%)
9	hundred	1,920 (90.4%)	204 (9.6%)	houston	86 (3.4%)	2,470 (96.6%)
10	shipping	1,806 (93.0%)	135 (7.0%)	kaminski	0 (0.0%)	1,855 (100.0%)
11	drug	1,798 (95.9%)	76 (4.1%)	louise	15 (0.8%)	1,820 (99.2%)
12	brand	1,778 (93.2%)	129 (6.8%)	portland	23 (1.4%)	1,642 (98.6%)
13	pill	1,729 (98.9%)	19 (1.1%)	hourahead	0 (0.0%)	1,390 (100.0%)
14	med	1,664 (98.5%)	26 (1.5%)	mike	113 (8.0%)	1,294 (92.0%)
15	cialis	1,632 (100.0%)	0 (0.0%)	parse	17 (1.3%)	1,253 (98.7%)
16	adobe	1,610 (96.2%)	63 (3.8%)	txt	25 (2.0%)	1,219 (98.0%)
17	safe	1,593 (90.3%)	172 (9.7%)	westdesk	0 (0.0%)	1,206 (100.0%)
18	meaning	1,592 (92.3%)	133 (7.7%)	detect	112 (9.0%)	1,138 (91.0%)
19	mailing	1,563 (98.8%)	19 (1.2%)	variance	2 (0.2%)	1,125 (99.8%)
20	creative	1,563 (92.6%)	125 (7.4%)	kitchen	44 (3.8%)	1,118 (96.2%)
21	differ	1,539 (96.4%)	58 (3.6%)	jeff	75 (6.3%)	1,113 (93.7%)
22	soft	1,497 (93.2%)	110 (6.8%)	hpl	5 (0.5%)	1,104 (99.5%)
23	men	1,466 (91.3%)	139 (8.7%)	fyi	62 (5.4%)	1,093 (94.6%)
24	herein	1,447 (97.2%)	41 (2.8%)	ancillary	34 (3.0%)	1,091 (97.0%)
25	studio	1,420 (95.9%)	61 (4.1%)	kevin	29 (2.7%)	1,036 (97.3%)
26	photoshop	1,408 (98.6%)	20 (1.4%)	bob	72 (6.6%)	1,027 (93.4%)
27	subscriber	1,403 (92.5%)	114 (7.5%)	daren	4 (0.4%)	1,012 (99.6%)
28	africa	1,357 (96.0%)	56 (4.0%)	chris	71 (6.8%)	979 (93.2%)
29	macromedia	1,341 (99.2%)	11 (0.8%)	sally	35 (3.5%)	952 (96.5%)
30	advisor	1,338 (90.9%)	134 (9.1%)	meter	60 (6.0%)	943 (94.0%)

Table A.2: 30 most common spam words and ham words. Spam (resp. ham) words are words heavily present in spam (resp. ham) emails, that is 90% of the emails in which the spam (resp. ham) word appears must be spam (resp. ham) emails.

```

From uktfu@eei.ericsson.se Fri Jul 19 02:11:31 2002
Return-Path: <uktfu@eei.ericsson.se>
Delivered-To: yyyy@localhost.netnoteinc.com
Received: from localhost (localhost [127.0.0.1])
    by phobos.labs.netnoteinc.com (Postfix) with ESMTP id BD26743F90
    for <jm@localhost>; Thu, 18 Jul 2002 21:11:30 -0400 (EDT)
Received: from dogma.slashnull.org [212.17.35.15]
    by localhost with IMAP (fetchmail-5.9.0)
    for jm@localhost (single-drop); Fri, 19 Jul 2002 02:11:30 +0100 (IST)
Received: from mandark.labs.netnoteinc.com ([213.105.180.140]) by
    dogma.slashnull.org (8.11.6/8.11.6) with ESMTP id g6J0rxJ14046 for
    <jm@jason.org>; Fri, 19 Jul 2002 01:53:59 +0100
Received: from kale.meridenbilisim.com ([213.204.65.66]) by
    mandark.labs.netnoteinc.com (8.11.6/8.11.6) with ESMTP id g6J0ruR24064 for
    <jm@netnoteinc.com>; Fri, 19 Jul 2002 01:53:57 +0100
Received: from darkside.demon.co.uk ([64.173.75.218]) by
    kale.meridenbilisim.com with Microsoft SMTPSVC(5.0.2195.2966);
    Mon, 22 Jul 2002 03:54:41 +0300
To: <Undisclosed.Recipients@netnoteinc.com>
From: "Julie" <uktfu@eei.ericsson.se>
Subject: Re: Instant Quote
Date: Thu, 18 Jul 2002 17:59:10 -1900
MIME-Version: 1.0
Message-Id: <KALET6658EMJmAkxWne000001a2@kale.meridenbilisim.com>
X-Originalarrivalttime: 22 Jul 2002 00:54:42.0462 (UTC) FILETIME=[5055F3E0:01C2311A]
Content-Type: text/html; charset="iso-8859-1"
Content-Transfer-Encoding: quoted-printable

<HTML>
<HEAD>
<TITLE></TITLE>
</HEAD>
<BODY>
<table width=3D520 cellpadding=3D0 cellspacing=3D0 border=3D4 bordercolor=
=3D00000A0>
<tr><td bgcolor=3D00000A0>
<font face=3DTahoma,Arial,Helvetica size=3D5 color=3DFFFFFF><b>Now you can=
have HUNDREDS of lenders compete for your loan!</b>
<br>
</font>
<font face=3DTahoma,Arial,Helvetica size=3D2 color=3DFFFFFF><BR><b>
<li>Refinancing
<li>New Home Loans
<li>Debt Consolidation
<li>Debt Consultation
<li>Auto Loans
<li>Credit Cards
<li>Student Loans
<li>Second Mortgage
<li>Home Equity
</li></font>
</td>
<td width=3D300>
<font face=3DArial,Helvetica size=3D3 color=3D0000050><b>Dear Homeowner,</b>
>
<p align=3D"justify"><font face=3D"times new roman, times" size=3D3 color=3D=
0000050>Interest Rates are at their lowest point in 40 years!
We help you find the best rate for your situation by matching your needs w=
ith hundreds of lenders! <u><b>Home Improvement, Refinance, Second Mortgage=
e, Home Equity Loans, and More!</b></u> Even with less than perfect credit=
!<br><br>This service is <b>100% FREE</b> to home owners and new home buye=
rs without any obligation.
<br><br>Just fill out a quick, simple form and jump-start your future plan=
s today!</font><br><br>
<a href=3Dhttp://61.129.68.19/user0201/index.asp?Afft=3DQM12><font face=3D=
verdana,arial,Helvetica size=3D5 color=3D0000000><b>Click Here To Begin</b>=
</FONT></A>
</font></p>
</td></tr></table><br>
<table width=3D516>
<tr><td><font face=3D"arial,Helvetica" size=3D1>Please know that we do not
want to send you information regarding our special offers if you do not =
wish to receive it. If you would no longer like us to contact you or feel=
that you have received this email in error, please <a href=3D"http://61.1=
29.68.19/light/watch.asp">click here to unsubscribe</a>.</font></td></tr>
</table>
</BODY>
</HTML>

```

Figure A.1: Example of spam email in its raw form.

re instant quote now you can have hundred of lender compete for your loan
refinance new home loan debt consolidation debt consultation auto loan credit
card student loan second mortgage home equity dear homeowner interest rate be
at their low point in numberstruct year we help you find the best rate for your
situation by match your need with hundred of lender home improvement refinance
second mortgage home equity loan and more even with less than perfect credit
this service be numberstruct free to home owner and new home buyer without any
obligation just fill out a quick simple form and jump start your future plan today
hyperlinkstruct please know that we do not want to send you information regard
our special offer if you do not wish to receive it if you would no longer like u to
contact you or feel that you have receive this email in error please hyperlinkstruct
click here to unsubscribe

Figure A.2: *Example spam email after preprocessing.*

Appendix B

Cross-validation results

This appendix presents the cross-validation results discussed in section 5.2.4.

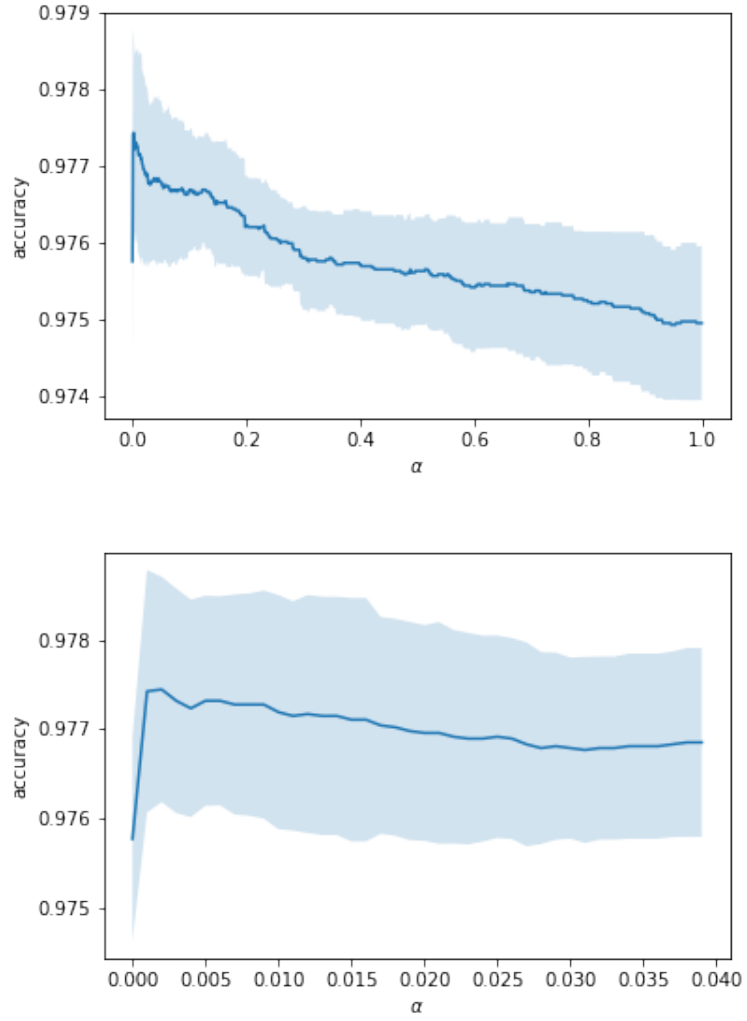


Figure B.1: Cross-validation results for multinomial naive Bayes classifier. The graphics depict the accuracy on the validation data (± 1 standard deviation) with respect to the hyperparameter α .

		#decision trees			
		50	100	200	400
min #samples before splitting	2	98.805	99.015	99.049	99.054
	4	98.841	98.962	99.028	99.043
	7	98.820	98.962	99.003	99.020
	10	98.814	98.935	98.967	99.035
	40	98.735	98.863	98.926	98.962
	100	98.636	98.765	98.757	98.782

Table B.1: Cross-validation results for random forest. The table gives the accuracy for a grid of hyperparameters. The hyperparameters taken into account are the number of decision trees in the random forest and the minimum number of samples for splitting a node in a decision tree.

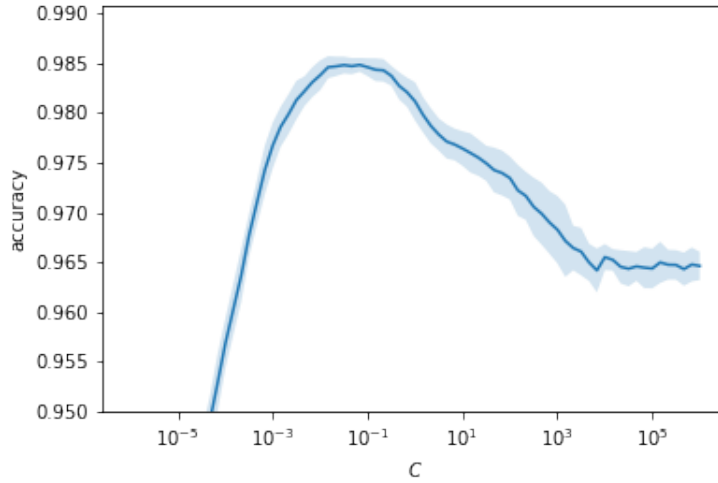


Figure B.2: Cross-validation results for the linear support vector machine. The graphics depict the accuracy on the validation data (± 1 standard deviation) with respect to the hyperparameter C .

		#fully connected layers	
		1	2
d	50	96.319%	96.616%
	300	96.340%	96.722%

Table B.2: Accuracy on validation set for various convolution neural networks models.

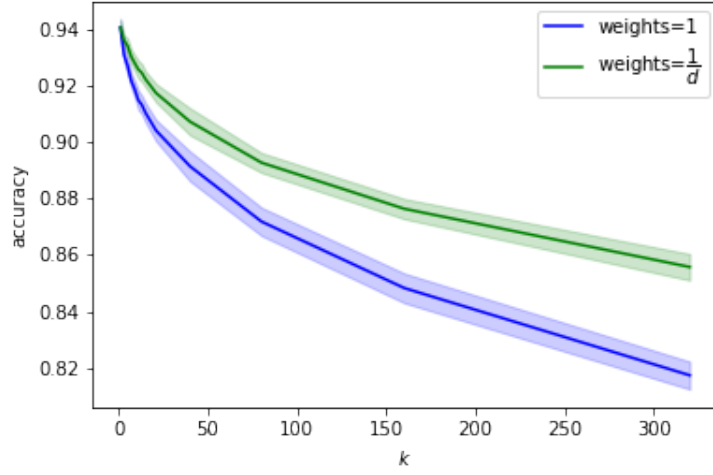


Figure B.3: Cross-validation results for k -nearest neighbors. The graphics depict the accuracy on the validation data (± 1 standard deviation) with respect to the hyperparameter k . We consider regular KNN and weighted KNN.

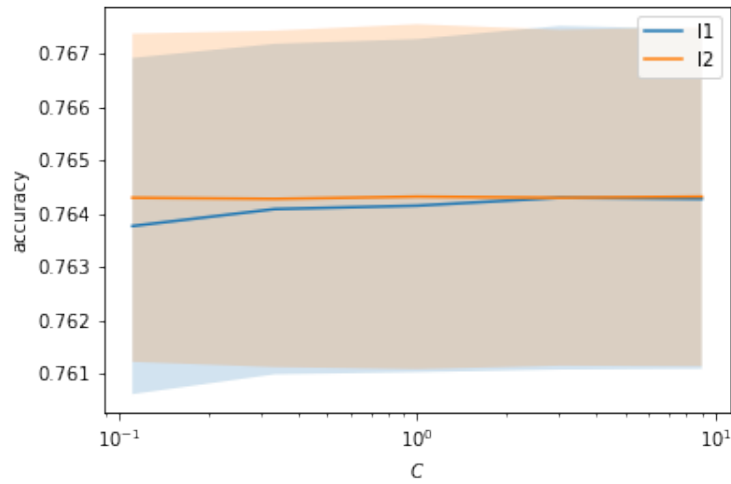


Figure B.4: Cross-validation results for logistic regression. The graphics depict the accuracy on the validation data (± 1 standard deviation) with respect to the hyperparameter C . We consider l_1 and l_2 regularization.

Bibliography

- [1] Spam. URL <https://www.merriam-webster.com/dictionary/spam>.
- [2] Computer security principles, January 2007. URL <https://www.cs.clemson.edu/course/cpsc420/material/Design%20Principles/Design%20Principles.pdf>.
- [3] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *CoRR*, abs/1712.03141, 2017. URL <http://arxiv.org/abs/1712.03141>.
- [4] Harris Drucker, Christopher Burges, Linda Kaufman, Alexander Smola, and V. Vapnik. Support vector regression machines. *Adv Neural Inform Process Syst*, 28:779–784, 01 1997.
- [5] Corey Dunn, Nour Moustafa, and Benjamin Turnbull. Robustness evaluations of sustainable machine learning models against data poisoning attacks in the internet of things. *Sustainability*, 12(16), 2020. ISSN 2071-1050. doi: 10.3390/su12166434. URL <https://www.mdpi.com/2071-1050/12/16/6434>.
- [6] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. Hotflip: White-box adversarial examples for text classification, 2018.
- [7] Michael Elhadad. SPAM Classifier using Scikit-Learn. URL https://www.cs.bgu.ac.il/~elhadad/nlp16/spam_classifier.html.
- [8] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2015.
- [9] Thomas Hofmann, Bernhard Schölkopf, and Alexander J. Smola. Kernel methods in machine learning. *The Annals of Statistics*, 36(3), Jun 2008. ISSN 0090-5364. doi: 10.1214/009053607000000677. URL <http://dx.doi.org/10.1214/009053607000000677>.

- [10] Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. Is bert really robust? a strong baseline for natural language attack on text classification and entailment, 2020.
- [11] Dan Jurafsky. *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition*. Prentice Hall, Upper Saddle River, N.J, 2000. ISBN 9780130950697.
- [12] Matthew Mayo KDnuggets. A framework for approaching textual data science tasks, Nov 2017. URL <https://www.kdnuggets.com/2017/11/framework-approaching-textual-data-tasks.html>.
- [13] Kalpesh Krishna and Nicolas Papernot. How to steal modern nlp systems with gibberish?, Apr 2020. URL <http://www.cleverhans.io/2020/04/06/stealing-bert.html>.
- [14] V. Kuleshov, Shantanu Thakoor, Tingfung Lau, and S. Ermon. Adversarial examples for natural language classification problems. 2018.
- [15] Tatyana Kulikova, Tatyana Shcherbakova, and Tatyana Sidorina. Spam and phishing in 2020. URL <https://securelist.com/spam-and-phishing-in-2020/100512/>.
- [16] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.
- [17] Bin Liang, Hongcheng Li, Miaoqiang Su, Pan Bian, Xirong Li, and Wenchang Shi. Deep text classification can be fooled. *CoRR*, abs/1704.08006, 2017. URL <http://arxiv.org/abs/1704.08006>.
- [18] Daniel Lowd and Christopher Meek. Good word attacks on statistical spam filters. 01 2005.
- [19] Matthew Mayo. A general approach to preprocessing text data. URL <https://www.kdnuggets.com/2017/12/general-approach-preprocessing-text-data.html>.
- [20] John McCarthy. Arthur samuel: Pioneer in machine learning. URL <http://infolab.stanford.edu/pub/voy/museum/samuel.html>.
- [21] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

- [22] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013. URL <http://arxiv.org/abs/1310.4546>.
- [23] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA, 1969.
- [24] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997. ISBN 978-0-07-042807-2.
- [25] John X. Morris, Eli Lifland, Jin Yong Yoo, Jake Grigsby, Di Jin, and Yanjun Qi. Textattack: A framework for adversarial attacks, data augmentation, and adversarial training in nlp, 2020.
- [26] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples, 2016.
- [27] David Reinsel, John Gantz, and John Rydning. Data age 2025: The evolution of data to life-critical, 2017. URL <http://infolab.stanford.edu/pub/voy/museum/samuel.html>.
- [28] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958. doi: 10.1037/h0042519.
- [29] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, 2014.
- [30] Michel Verleysen. Nonlinear regression with multi-layer perceptrons, from machine learning course. Accessed: 2021–26-05.
- [31] Yevgeniy Vorobeychik. *Adversarial machine learning*. Morgan & Claypool Publishers, San Rafael, California, 2018. ISBN 9781681733982.
- [32] Chenran Wang, Danyi Zhang, Suye Huang, Xiangyang Li, and Leah Ding. Crafting adversarial email content against machine learning based spam email detection. pages 23–28, 05 2021. doi: 10.1145/3457340.3458302.
- [33] Xinran Wang, Yu Xiang, Jun Gao, and Jie Ding. Information laundering for model privacy, 2020.

- [34] Xujing Yao, Xinyue Wang, Shuihua Wang, and Yu-Dong Zhang. A comprehensive survey on convolutional neural network in medical image analysis. *Multimedia Tools and Applications*, 08 2020. doi: 10.1007/s11042-020-09634-7.
- [35] Liping Yuan, Xiaoqing Zheng, Yi Zhou, Cho-Jui Hsieh, Kai wei Chang, and Xuanjing Huang. Generating universal language adversarial examples by understanding and enhancing the transferability across neural models, 2020.
- [36] Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification, 2016.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | www.uclouvain.be/epl