ΣΤΡΑΤΙΩΤΙΚΗ ΣΧΟΛΗ ΕΥΕΛΠΙΔΩΝ

Τμήμα Στρατιωτικών Επιστημών

ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ

ΔΙΙΔΡΥΜΑΤΙΚΟ ΔΙΑΤΜΗΜΑΤΙΚΟ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΑΚΑΔΗΜΑΪΚΟΥ ΕΤΟΥΣ 2017-18

ΕΦΑΡΜΟΣΜΕΝΗ

ΕΠΙΧΕΙΡΗΣΙΑΚΗ ΕΡΕΥΝΑ & ΑΝΑΛΥΣΗ

(ΠΔ 97 /2015/ΦΕΚ 163Α'/20.08.2014)

ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

Σχολή Μηχανικών Παραγωγής & Διοίκησης

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΑΤΡΙΒΗ

ΠΡΑΚΤΙΚΕΣ ΜΗΧΑΝΙΚΗΣ ΜΑΘΗΣΗΣ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗΝ ΚΥΒΕΡΝΟΑΣΦΑΛΕΙΑ

"MACHINE LEARNING PRACTICES FOR CYBERSECURITY"

Διατριβή που υπεβλήθη για την μερική ικανοποίηση των απαιτήσεων για την απόκτηση Μεταπτυχιακού Διπλώματος Ειδίκευσης

Υπό:

ΧΑΣΙΩΤΗ ΘΕΜΙΣΤΟΚΛΗ

Α.Μ.: 2017018020

2022

Η Μεταπτυχιακή Διατριβή του Χασιώτη Θεμιστοκλή εγκρίνεται:

ΤΡΙΜΕΛΗΣ ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ

Αναπλ. Καθηγητής Δούκας Νικόλαος (Επιβλέπων) ……..

Καθηγητής Μπάρδης Νικόλαος …….,……..

Καθηγητής Ματσατσίνης Νικόλαος …..……..

ΣΕΛΙΔΑ ΣΚΟΠΙΜΑ ΚΕΝΗ

Έτος 2022

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΕΡΙΕΧΟΜΕΝΑ ΕΙΚΟΝΩΝ

ΠΕΡΙΕΧΟΜΕΝΑ ΠΙΝΑΚΩΝ

Περίληψη

Η εργασία αυτή μελετά εργαλεία τεχνητής νοημοσύνης για την έγκαιρη και αποτελεσματική αναγνώριση, προτεραιοποίηση και αποσόβηση των απειλών, προκειμένου να αναπτυχθούν συστήματα λογισμικού για την υποστήριξη αποφάσεων εκτίμησης και αντιμετώπισης απειλών στον κυβερνοχώρο.

Τα θέματα που διερευνά η παρούσα εργασία περιλαμβάνουν την ανίχνευση παραβιάσεων δικτύου με βάση την ανίχνευση ανωμαλιών (anomaly-based network intrusion detection), την ανίχνευση εσωτερικών απειλών, την ανίχνευση ανεπιθύμητης αλληλογραφίας και phishing, την ανίχνευση κακόβουλου λογισμικού με βάση την συμπεριφορά και την ανίχνευση κακόβουλης δραστηριότητας με βάση την ανάλυση εντολών σε επίπεδο γλώσσας μηχανής κλπ.

Οι μέθοδοι της τεχνητής νοημοσύνης που προσφέρονται για το σκοπό της εργασίας περιλαμβάνουν μηχανική μάθηση για την ανάλυση λογισμικού, μηχανική μάθηση και εξόρυξη δεδομένων για την ασφάλεια βάσεων δεδομένων, μηχανική μάθηση για την ανίχνευση κακόβουλου λογισμικού και άλλα. Πηγή ιδιαιτέρως σημαντικών προβλημάτων είναι το γεγονός ότι αναγκαστικά, η εκπαίδευση των αλγορίθμων τεχνητής νοημοσύνης γίνεται με βάση ένα περιορισμένο σύνολο παραδειγμάτων, αφού η συμπεριφορά του κακόβουλου λογισμικού είναι άγνωστη και ταχύτατα μεταβαλλόμενη.

Στην εργασία παρουσιάζονται οι κύριες αρχές της ταξινόμησης και της ομαδοποίησης στην περίπτωση της εποπτευόμενης μάθησης. Στη συνέχεια διαφορετικά σύνολα δεδομένων υποβλήθηκαν σε επεξεργασία για την εκτέλεση εντοπισμού κακόβουλου λογισμικού και τέλος παρουσιάζεται η ανίχνευση και ταυτοποίηση διαφορετικών κυβερνοεπιθέσεων.

Abstract

This thesis studies artificial intelligence tools for early and effective threat identification, prioritization and deterrence in order to develop software systems to support cyber threat assessment and response decisions.

The topics explored in this thesis include anomaly-based network intrusion detection, insider threat detection, spam and phishing detection, behavior-based malware detection, and malware detection activity based on parsing machine language commands etc.

The AI methods offered for this purpose include machine learning for software analysis, machine learning and data mining for database security, machine learning for malware detection and more. A source of particularly significant problems is the fact that artificial intelligence algorithms are necessarily trained on a limited set of examples, since malware behavior is unknown and rapidly changing.

Finally presents the main classification and clustering principles in the case of supervised learning. Then different data sets were processed to perform malware detection and finally the detection and identification of different cyber attacks is presented.

ΚΕΦΑΛΑΙΟ 1

ΕΙΣΑΓΩΓΗ ΣΤΙΣ ΕΦΑΡΜΟΓΕΣ ΤΗΣ ΤΕΧΝΗΤΗΣ ΝΟΗΜΟΣΥΝΗΣ ΠΑΝΩ ΣΤΗΝ ΚΥΒΕΡΝΟΑΣΦΑΛΕΙΑ

Τα τελευταία χρόνια με την ανάγκη της ψηφιακής μεταρρύθμισης και του ψηφιακού μετασχηματισμού για τη ψηφιακή διακυβέρνηση, συμβάλλοντας σε αυτό και οι ανάγκες που προέκυψαν από τις επιπτώσεις του COVID19, ένα πολύ μεγάλο ποσοστό του πληθυσμού είδε την άμεση εξάρτηση της καθημερινότητας του και της εργασίας από την ύπαρξη του Διαδικτύου, την ταχύτητα σύνδεσης στο Διαδίκτυο αλλά και την εύρυθμη λειτουργία των υπολογιστών. Οι ανάγκες της 4^{ης} βιομηχανικής επανάστασης αλλά και οι επιπτώσεις του COVID19, αύξησαν δραματικά τη χρήση υπολογιστικών συστημάτων και των έξυπνων κινητών συσκευών για τις ηλεκτρονικές υπηρεσίες στο εμπόριο, την υγεία, την εκπαίδευση και φυσικά στην εξ αποστάσεως εργασία. Αυτή η ραγδαία αύξηση της χρήσης των συστημάτων που είναι συνδεδεμένα στο διαδίκτυο, από διαφορετικό επίπεδο χρηστών, αύξησαν τις ευπάθειες των συστημάτων καθώς και τις απειλές για την ασφάλεια στην κυβερνοχώρο. Ταυτόχρονα αυξήθηκαν οι απαιτήσεις της ασφάλειας και της αξιοπιστίας των συστημάτων και των πληροφοριών. Το ζήτημα της ασφάλειας από το κλασικό πληροφορικό σύστημα, ανάγεται πλέον σε έναν πολύ κρίσιμο παράγοντα για τις αδιάλειπτες λειτουργίες της ανθρωπότητας στο περίπλοκο περιβάλλον του κυβερνοχώρου [N1], [N2]. Μέχρι σήμερα η διατήρηση ενός πληροφοριακού συστήματος που δέχεται επιθέσεις σε λειτουργική κατάσταση απαιτεί κατά κύριο λόγο την προάσπιση της λεγόμενης τριάδας της CIA (CIA triad). Η τριάδα αυτή περιλαμβάνει τις τρεις θεμελιώδεις εγγυήσεις που πρέπει να παρέχει ένα πληροφοριακό σύστημα προκειμένου να θεωρείται λειτουργικό και είναι η εμπιστευτικότητα, η ακεραιότητα και η διαθεσιμότητα. Επιπλέον απαιτήσεις συμπεριλαμβάνουν την εξουσιοδότηση (authorization), την αυθεντικοποίηση (authentication) και την μη απάρνηση (non reputation) [3].

Σήμερα με τον συνεχώς αυξανόμενο αριθμό ανθρώπινων δραστηριοτήτων, έχει παρουσιαστεί η έννοια της επιβίωσης των πληροφοριακών συστημάτων του κυβερνοχώρου [2], [4], [5], [6]. Η βιωσιμότητα περιγράφει την ικανότητα ενός συστήματος να αποτρέπει τους επίδοξους εισβολείς στον κυβερνοχώρο, να αποφεύγει την πλήρη κατάρρευση, να διατηρεί έστω και μειωμένο το επίπεδο των υπηρεσιών κατά τη διάρκεια μιας συντονισμένης επίθεσης και ανακάμπτει όλες τις δυνατότητες μετά την παύση της επίθεσης. Η εξασφάλιση αυτών των νέων απαιτήσεων ως συνθήκη ασφάλειας στον κυβερνοχώρο στοχεύουν στην προώθηση της συνθήκης ασφάλειας της βιωσιμότητας (survivability) εστιάζοντας στα τρία R's, δηλαδή την ευρωστία (robustness), την απόκριση (response) και την ανθεκτικότητα (resilience) [7] που συμπληρώνουν την παραδοσιακή τριάδα CIA

Ως εκ τούτου, οι αλγόριθμοι τεχνητής νοημοσύνης χρησιμοποιούνται ως εργαλείο για τη συνεχή παρακολούθηση των συστημάτων πληροφοριών υπολογιστών και την παραγωγή προειδοποιήσεων για επικείμενες απειλές στον κυβερνοχώρο.

Έτσι σήμερα υπάρχει η απαίτηση μα αναπτυχθούν [8] σύγχρονα συστήματα άμυνας στον κυβερνοχώρο που βασίζονται στην Τεχνητή Νοημοσύνη (AI). Αυτά τα συστήματα της τεχνητής νοημοσύνης επιδιώκουν την αύξηση της επιβίωσης στοχεύοντας τα 3R. Οι τεχνικές τεχνητής νοημοσύνης θα προάγουν την ευρωστία ενισχύοντας την ικανότητα ενός συστήματος να διατηρεί την αναμενόμενη συμπεριφορά του, σε περίπτωση που επεξεργάζεται απροσδόκητα δεδομένα, αναπτύσσοντας και υλοποιώντας κάποιο λογισμικό αυτοελέγχου και αυτοίασης [N8]. Αυτά τα απροσδόκητα στοιχεία μπορεί να προκύψουν από σφάλματα, τυχαία συμβάντα ή από κακόβουλη δραστηριότητα προεχόμενη τόσο έξω όσο και από μέσα από το σύστημα. Στο πλαίσιο της απόκρισης, η τεχνητή νοημοσύνη επιτρέπει σε ένα σύστημα να νικήσει μια επίθεση χωρίς παρέμβαση και ταυτόχρονα να βελτιστοποιήσει τη στρατηγική απόκρισης και να προσαρμόσει την επιθετικότητά του με βάση προηγούμενες επιτυχίες [8]. Τέλος, η ανθεκτικότητα προωθείται από την τεχνητή νοημοσύνη ενισχύοντας την ικανότητα του συστήματος να ανιχνεύει απειλές και ανωμαλίες και ως εκ τούτου αυξάνοντας την ικανότητά τους να αντέχουν σε επιθέσεις [8].

Έτσι για την επιβίωση των συστημάτων σε αυτό το περίπλοκο περιβάλλον του κυβερνοχώρου με τις συνεχιζόμενες κυβερνοεπιθέσεις, απαιτείται συνεχής παρακολούθηση για την επιβίωση του συστήματος [8]. Η παρακολούθηση χρησιμοποιείται για την έγκαιρη ανίχνευση αποκλίσεων του πραγματικού συστήματος από την αναμενόμενη συμπεριφορά, προκειμένου να ενεργοποιηθεί η κατάλληλη απόκριση.

Πιο συγκεκριμένα, στη παρούσα έρευνα μελετάτε η μεθοδολογία ώστε η άμυνα έναντι των κυβερνοεπιθέσεων να εξασφαλίζεται με την κατανόηση της αλυσίδας της εκτέλεσης (*kill chain*), την αλληλουχία δηλαδή των ενεργειών ενός επιτιθέμενου παράγοντα προκειμένου να επιτύχει τους κακόβουλους σκοπούς του. Η αλυσίδα αυτή περιλαμβάνει (i) την αναγνώριση, όπου συγκεντρώνονται πληροφορίες για συγκεκριμένους στόχους μέσα στο υπό επίθεση πληροφοριακό σύστημα καθώς και αδυναμίες του, (ii) την ανάπτυξη των κατάλληλων όπλων (*weaponization*), η σχεδίαση δηλαδή κακόβουλου λογισμικού ή άλλων μέσων για την συγκεκριμένη επίθεση, (iii) την παράδοση, δηλαδή τη διαδικασία μέσω της οποίας το κακόβουλο λογισμικό μεταφέρεται στο πληροφοριακό σύστημα στόχο, (iv) την εκμετάλλευση (*exploitation*), οπότε οι αδυναμίες που ανιχνεύθηκαν χρησιμοποιούνται για να εκτελεστεί το κακόβουλο λογισμικό, (v) την εγκατάσταση του κακόβουλου λογισμικού, (vi) την απόκτηση από τον επιτιθέμενο του πλήρους ελέγχου και χειρισμού του στόχου και τέλος (vii) την χρήση του στόχου από τον επιτιθέμενο για την επίτευξη των σκοπών του (υποκλοπή δεδομένων, παραποίηση κλπ) [9]. Με την υπόθεση ότι διαφορετικές φάσεις της επίθεσης ήταν αποτελεσματικές, η άμυνα περιλαμβάνει τη σχεδίαση τεχνικών και

αντιμέτρων που θα περιορίσουν τη ζημιά και θα εμποδίσουν την περεταίρω πρόοδο της επίθεσης.

Η διαδικασία αντιμετώπισης μίας κυβερνοεπίθεσης είναι διαδραστική και απαιτεί την συγκέντρωση και ανάλυση δεδομένων του συστήματος ώστε οι χειριστές του να κάνουν τις κατάλληλες ενέργειες σε πραγματικό χρόνο, για την αντιμετώπιση του κινδύνου. Η χρήση τεχνητής νοημοσύνης μπορεί να υποστηρίξει τη διαδικασία λήψης των αναγκαίων αποφάσεων από τους χειριστές. Η εργασία αυτή μελετά εργαλεία τεχνητής νοημοσύνης για την έγκαιρη και αποτελεσματική αναγνώριση, προτεραιοποίηση και αποσόβηση των απειλών, προκειμένου να αναπτυχθούν συστήματα λογισμικού για την υποστήριξη αποφάσεων εκτίμησης και αντιμετώπισης απειλών στον κυβερνοχώρο.

1.1 Η εξέλιξη της τεχνητής νοημοσύνης

1.1.1 Η σημαντικότητα της τεχνητής νοημοσύνης στην κυβερνοασφάλεια

Στις μέρες μας οι οργανισμοί ξοδεύουν δισεκατομμύρια χρημάτων παγκοσμίως πάνω στην κυβερνοασφάλεια. Η τεχνητή νοημοσύνη εμφανίστηκε σαν μια πολύ καλή λύση για τη δημιουργία εξυπνότερων και ασφαλέστερων συστημάτων ασφαλείας τα οποία επιτρέπουν στους χρήστες να προβλέπουν και να εντοπίζουν ύποπτες δραστηριότητες στο δίκτυο όπως την εισβολή μη εξουσιοδοτημένων χρηστών. Προστατεύοντας με αυτόν τον τρόπο ευαίσθητα δεδομένα, προσωπικές πληροφορίες, πνευματικά δικαιώματα, κυβερνητικά και βιομηχανικά πληροφοριακά συστήματα από την κλοπή και τη καταστροφή.

Οι περισσότερες απειλές χρησιμοποιούν τα μηνύματα του ηλεκτρονικού ταχυδρομείου ως μέσο διείσδυσης στους υπολογιστές. Εφόσον το αριθμός των απειλών που διεισδύει με αυτό τον τρόπο στο σύστημα είναι αρκετά μεγάλο, είναι απαραίτητη η χρησιμοποίηση αυτόματων διαδικασιών αναγνώρισης οι οποίες εκμεταλλεύονται αλγορίθμους μηχανικής εκμάθησης.

Στη συγκεκριμένη εργασία παρουσιάζονται δημοφιλής και επιτυχημένες πρακτικές τεχνητής νοημοσύνης και μοντέλων που μπορούν να υιοθετηθούν για τον εντοπισμό πιθανών επιθέσεων με σκοπό την προστασία των παραπάνω δεδομένων εστιάζοντας περισσότερο στις απειλές που εισέρχονται σε κάποιο σύστημα μέσω των μηνυμάτων του ηλεκτρονικού ταχυδρομείου.

1.1.2 Μια σύντομη εισαγωγή στα ειδικά συστήματα

Μια από τις πρώτες προσπάθειες της αυτόματης εκμάθησης αποτελούσε ο καθορισμός του συστήματος αποφάσεων βασισμένο σε κανόνες (*ruled-based decision system*), το οποίο κάλυπτε όλες τις πιθανές διακλαδώσεις και συγκεκριμένες περιπτώσεις οι οποίες μπορούν να βρεθούν στον πραγματικό κόσμο. Με αυτό τον τρόπο, όλες οι πιθανές περιπτώσεις ήταν

κωδικοποιημένες μέσα στις αυτοματοποιημένες λύσεις εκμάθησης, και ήταν επαληθευμένες από ειδικούς του τομέα.

Ο βασικός περιορισμός τέτοιων συστημάτων ήταν ότι μείωναν τις αποφάσεις των λογικών μεταβλητών αλλά και επίσης περιόριζαν την ικανότητά τους να προσαρμόζονται στις διαφορετικές περιπτώσεις του πραγματικού κόσμου.

1.1.3 Εκφράζοντας την απειριοριστική φύση της πραγματικότητας

Ως γνωστό, οι βασικές καταστάσεις στις οποίες εμφανίζονται στον πραγματικό κόσμο δεν μπορούν να αναπαριστούν χρησιμοποιώντας μόνο το Αληθές/Ψευδές μοντέλο κατηγοριοποίησης, είναι επομένως απαραίτητο να γίνεται η όσο το δυνατόν καλύτερη χρήση των δεδομένων έτσι ώστε να εμφανιστούν νέες τάσεις και περιπτώσεις, χρησιμοποιώντας στατιστικές και πιθανά μοντέλα τα οποία μπορούν πιο κατάλληλα να αναπαραστήσουν την απειριοριστική φύση της πραγματικότητας.

1.1.4 Από τη στατιστική προς τη μηχανική μάθηση

Αν και η εισαγωγή των στατιστικών μοντέλων ξεπέρασε τα όρια των ειδικών συστημάτων, η σημαντική ακαμψία της συγκεκριμένης μεθόδου παρέμεινε, καθώς τα στατιστικά μοντέλα, όπως τα συστήματα κρίσεων και λήψης αποφάσεων βασιζόμενα σε κανόνες «*rule-based decision systems*», ήταν εγκατεστημένα εκ των προτέρων και δεν μπορούσαν να μορφοποιηθούν περαιτέρω για να προσαρμοστούν σε νέα δεδομένα και γνώση. Για να ξεπεραστούν αυτοί οι περιορισμοί, ήταν αναγκαίο να υιοθετηθεί μια επαναληπτική προσέγγιση η οποία θα επέτρεπε την εισαγωγή αλγορίθμων μηχανικής εκμάθησης ικανούς να γενικεύουν τα περιγραφικά μοντέλα ξεκινώντας από τα διαθέσιμα δεδομένα, γενικεύοντας αυτόνομα τα δικά τους χαρακτηριστικά, χωρίς να περιορίζουν τον εαυτό τους με προκαθορισμένες συναρτήσεις, αλλά προσαρμόζοντάς τες σε μία συνεχή εξέλιξη της διαδικασίας εκμάθησης του αλγορίθμου [10].

1.2 Τύποι μηχανικής εκμάθησης

Η διαδικασία της μηχανικής μάθησης δεδομένων μπορεί να πάρει διαφορετικές μορφές, με διαφορετικά χαρακτηριστικά και ικανότητες πρόβλεψης. Οι κυριότερες μορφές της μηχανικής μάθησης είναι οι παρακάτω:

- Επιτηρούμενη μάθηση ή εκμάθηση με επίβλεψη «*Supervised learning*»
- Μη επιτηρούμενη μάθηση ή Εκμάθηση χωρίς επίβλεψη «*Unsupervised learning*»
- Ενισχυτική μάθηση «*Reinforcement learning*»

Οι διαφορές μεταξύ των παραπάνω μεθόδων εκμάθησης βρίσκονται στον τύπο του επιθυμητού αποτελέσματος ή στον τύπο της εξόδου «output», βασισμένο στη φύση των εισόδων «input» που χρειάζονται για να το παράξουν [11].

1.2.1 Επιτηρούμενη μάθηση

Στην περίπτωση της επιτηρούμενης μάθησης, η εκπαίδευση του αλγορίθμου επιτυγχάνεται χρησιμοποιώντας εισόδους δεδομένων, από τις οποίες ο τύπος του επιθυμητού αποτελέσματος είναι ήδη γνωστός.

Στην πράξη οι αλγόριθμοι πρέπει να εκπαιδευτούν ώστε να αναγνωρίσουν τις σχέσεις μεταξύ των μεταβλητών που εκπαιδεύονται, προσπαθώντας να βελτιστοποιήσουν τις παραμέτρους μάθησης με βάση τις μεταβλητές-στόχους οι οποίες είναι ήδη γνωστές.

Ένα παράδειγμα επιτηρούμενης μάθησης αλγορίθμου είναι οι αλγόριθμοι κατηγοριοποίησης, οι οποίοι χρησιμοποιούνται στο πεδίο της κυβερνοασφάλειας για την κατηγοριοποίηση της ανεπιθύμητης αλληλογραφίας.

Ένα φίλτρο ανεπιθύμητης αλληλογραφίας εκπαιδεύεται δίνοντας στον αλγόριθμο δεδομένα εισόδου τα οποία εμπεριέχουν πολλά παραδείγματα email τα οποία είχαν κατηγοριοποιηθεί πιο πριν είτε ως ανεπιθύμητα ή κακόβουλα «*spam*» είτε ως γνήσια και αβλαβή «*ham*».

Ο αλγόριθμος κατηγοριοποίησης των κακόβουλων φίλτρων πρέπει να μάθει να κατηγοριοποιεί τα καινούρια emails τα οποία θα ληφθούν στο μέλλον, βάζοντάς τα στην κατηγορία *spam* ή *ham* βασισμένος στις προηγούμενες εκπαιδεύσεις του με τα δεδομένα εισόδου των ήδη κατηγοριοποιημένων email [12].

Άλλα παραδείγματα επιτηρούμενη μάθηση αλγορίθμων είναι τα παρακάτω:

- Παλινδρόμηση (*Regression (linear and logistic)*)
- Κ- κοντινών γειτόνων (*K- Nearest Neighbors (k-NNs)*)
- Υποστήριξη διανυσματικών μηχανών (*Support vector machines (SVMs)*)
- Δέντρα απόφασης και τυχαία δάση (*Decision trees and random forests*)
- Νευρωνικά Δίκτυα (*Neural networks (NNs) [13]*)

1.2.2 Μη επιτηρούμενη μάθηση

Στην περίπτωση της μη επιτηρούμενης μάθησης, οι αλγόριθμοι πρέπει να προσπαθήσουν να κατηγοριοποιήσουν τα δεδομένα ανεξάρτητα, χωρίς τη βοήθεια προηγούμενων κατηγοριοποιήσεων οι οποίες είναι δοσμένες από τον αναλυτή. Στον τομέα της κυβερνοασφάλειας, η χρήση αλγορίθμων μη επιτηρούμενης μάθησης είναι ιδιαίτερα

σημαντική για την αναγνώριση νέων μορφών κακόβουλων επιθέσεων, ανεπιθύμητης αλληλογραφίας και μορφές απάτης οι οποίες στο παρελθόν δεν είχαν ανιχνευτεί [14].

Παραδείγματα αλγορίθμων μη επιτηρούμενης μάθησης είναι τα παρακάτω:

- Μείωσης της διατομής (*Dimensionality reduction*)
    (1) Ανάλυση κύριας συνιστώσας (*Principal component analysis (PCA)*)
    (2) Ανάλυση Πυρήνα (*PCA Kernel*)
- Ομαδοποίησης (Clustering)
    (1) K-means
    (2) Hierarchical cluster analysis (HCA) [15]

1.2.3 Ενισχυτική μάθηση

Στην περίπτωση της ενισχυτικής μάθησης (RL), ακολουθείται μια διαφορετική στρατηγική εκμάθησης, η οποία εξομοιώνει την προσέγγιση δοκιμής και σφάλματος. Με αυτό τον τρόπο, παίρνοντας πληροφορίες από την ανατροφοδότηση κατά τη διάρκεια της διαδικασίας μάθησης, με σκοπό τη μεγιστοποίηση της ανταμοιβής, το τελικό αποτέλεσμα επιτυγχάνεται βάσει του αριθμού των σωστών αποφάσεων που ο αλγόριθμος έχει επιλέξει.

Στην πράξη, η διαδικασία εκμάθησης λαμβάνει χώρα χωρίς επίβλεψη, με τη βασική διαφορά ότι μια θετική ανταμοιβή καταλογίζεται σε κάθε σωστή απόφαση, και μια αρνητική σε κάθε λανθασμένη. Στο τέλος της διαδικασίας εκμάθησης, οι αποφάσεις του αλγορίθμου επαναξιολογούνται ανάλογα την τελική τιμή της ανταμοιβής [16].

Τα παρακάτω είναι παραδείγματα αλγορίθμων ενισχυτικής εκμάθησης:

- Μαρκοβιανές διαδικασίες (*Markov process*)
- Q μάθηση (*Q-learning*)
- Μέθοδος χρονικής διαφοράς (*Temporal difference (TD) methods*)
- Μέθοδος Monte Carlo [16]

1.3 Εκπαίδευση αλγορίθμου και βελτιστοποίηση

Κατά τη διάρκεια της προετοιμασίας αυτομάτων διαδικασιών εκμάθησης, αντιμετωπίζονται αρκετά προβλήματα. Η αντιμετώπιση αυτών των προβλημάτων είναι αναγκαία για να αναγνωριστούν και να αποφευχθεί η αναξιοπιστία των διαδικασιών αυτών, ελαττώνοντας την πιθανότητα ενός μεγάλου και βασικού λάθους, το οποίο στον τομέα της κυβερνοασφάλειας μπορεί να έχει καταστρεπτικές συνέπειες.

Ένα από τα κυριότερα προβλήματα που αντιμετωπίζονται, ειδικά στις περιπτώσεις διαμόρφωσης διαδικασιών αναγνώρισης απειλών, είναι η διαχείριση των περιστατικών που έχουν χαρακτηριστεί ως πιθανές απειλές από τον αλγόριθμο ενώ στην πραγματικότητα δεν είναι. Η διαχείριση αυτών των περιστατικών είναι αρκετά δύσκολη και χρονοβόρα καθώς ο αριθμός των απειλών που αναγνωρίζεται από ένα σύστημα είναι τόσο μεγάλος ώστε να απορροφά όλο το ανθρώπινο δυναμικό που ασχολείται με τη διαδικασία της αναγνώρισης [17].

Από την άλλη πλευρά, ακόμα και οι σωστές αναφορές, σε αρκετά μεγάλους αριθμούς, σταδιακά υπερφορτώνουν τους αναλυτές, αποσπώντας τους έτσι από τις πιο κρίσιμες και βασικές εργασίες τους. Έτσι λοιπόν προκύπτει η ανάγκη της βελτιστοποίησης των διαδικασιών εκμάθησης με σκοπό να μειωθεί ο αριθμός των περιπτώσεων που χρήζουν ανάλυση σε βάθος από τους αναλυτές τους ίδιους.

1.4 Τεχνητή νοημοσύνη στο πλαίσιο της κυβερνοασφάλειας

Με την εκθετική αύξηση της διασποράς των απειλών σε συνδυασμό με την καθημερινή διάδοση νέου κακόβουλου λογισμικού, είναι πρακτικά αδύνατο να σκεφτεί κανείς ότι η αντιμετώπιση όλων αυτών των απειλών μπορεί να γίνει μόνο από αναλύσεις οι οποίες έχουν γίνει από το ανθρώπινο δυναμικό. Ήταν απαραίτητη η εισαγωγή αλγορίθμων, οι οποίοι θα επέτρεπαν την αυτοματοποίηση της εισαγωγικής φάσης της ανάλυσης, ώστε να γίνεται ένα προκαταρκτικό φιλτράρισμα των απειλών που θα μελετηθούν στη συνέχεια από τους ειδικούς κυβερνοασφάλειας, κερδίζοντας έτσι πολύτιμο χρόνο για τη αποτελεσματική αντιμετώπιση των υποκείμενων επιθέσεων. Η δυναμική αντιμετώπιση των απειλών είναι αναγκαία, για την προσαρμογή στις αλλαγές που παρουσιάζουν οι πρωτοφανείς απειλές. Αυτό σημαίνει ότι οι αναλυτές κυβερνοασφάλειας δνη διαχειρίζονται μόνο τα εργαλεία και τις μεθόδους της κυβερνοασφάλειας, αλλά και επίσης μπορούν να ερμηνεύσουν και να αξιολογήσουν τα αποτελέσματα που προκύπτουν από αλγορίθμους μηχανικής εκμάθησης και τεχνητής νοημοσύνης [18].

Εργασίες οι οποίες χρησιμοποιούν τεχνητή νοημοσύνη είναι οι ακόλουθες:

- Κατηγοριοποίηση (Classification): Η συγκεκριμένη είναι από τις πιο βασικές εργασίες στα πλαίσια της κυβερνοασφάλειας. Χρησιμοποιείται για τη σωστή αναγνώριση τύπων παρόμοιων επιθέσεων, όπως για παράδειγμα διαφορετικά κομμάτια κακόβουλου λογισμικού τα οποία ανήκουν στην ίδια οικογένεια, που έχουν κοινά χαρακτηριστικά και συμπεριφορά ακόμα και αν τα ίχνη τους είναι εξαφανισμένα. Με τον ίδιο τρόπο κατηγοριοποιούνται και τα μηνύματα του ηλεκτρονικού ταχυδρομείου είτε ως ανεπιθύμητα είτε ως κανονικά [19].
- Ομαδοποίηση (Clustering): Η ομαδοποίηση διαφέρει από την κατηγοριοποίησης από την ικανότητά της να αναγνωρίζει αυτόματα τις κατηγορίες στις οποίες τα δείγματα

ανήκουν όταν οι πληροφορίες σχετικά με αυτές δεν είναι διαθέσιμες. Αυτή η διαδικασία επίσης έχει τεράστια σημασία στην ανάλυση κακόβουλου λογισμικού [20] .

- Προγνωστική ανάλυση (Predictive analysis): Με την εφαρμογή των νευρωνικών δικτύων, είναι δυνατό να εντοπιστούν οι απειλές όταν συμβαίνουν. Για το σκοπό αυτό, ήταν αναγκαία η υιοθέτηση μιας ιδιαίτερα δυναμικής προσέγγισης, η οποία θα επιτρέπει στους αλγόριθμους να βελτιστοποιούν αυτόματα τις μαθησιακές δυνατότητές τους [21].

Παρακάτω παρουσιάζονται πιθανές χρήσεις της τεχνητής νοημοσύνης στην κυβερνοασφάλεια:

- Προστασία δικτύου (*Network protection*): Η χρήσης της μηχανικής μάθησης επιτρέπει την εκτέλεση αρκετά εκλεπτυσμένων συστημάτων αναγνώρισης εισβολών (*Intrusion Detection Systems* ή *IDS*), τα οποία χρησιμοποιούνται στην περιοχή προστασίας της περιμέτρου του δικτύου [22].

- Προστασία τελικού σημείου (*Endpoint protection*): Απειλές όπως τα *ransomware*, (τύπος κακόβουλου λογισμικού το οποίο είναι σχεδιασμένο με τέτοιο τρόπο ώστε να μπλοκάρει την πρόσβαση σε ένα υπολογιστικό σύστημα μέχρι να πληρωθεί ένα συγκεκριμένο ποσό χρημάτων) μπορούν να εντοπιστούν κατά ένα πολύ μεγάλο βαθμό από αλγορίθμους που μαθαίνουν τη συμπεριφορά τέτοιων απειλών, έτσι ξεπερνάνε τα όρια των παραδοσιακών προγραμμάτων αναγνώρισης και καταστροφής ιών (*anti-virus*) [23].

- Ασφάλεια εφαρμογών (*Application security*): Μερικές από τις κυριότερες και πιο επικίνδυνες μορφές απειλών σε εφαρμογές είναι οι εξής:

    (1) Παραχάραξη αιτήματος από την πλευρά του διακομιστή (*Server-Side Request Forgery (SSRF)*)

    (2) SQL injection

    (3) Διασταυρούμενη δέσμη ενεργειών (*Cross-Site Scripting (XSS)*)

    (4) Κατανεμημένη άρνηση υπηρεσίας (*Distributed Denial of Service (DDoS)*) [24]

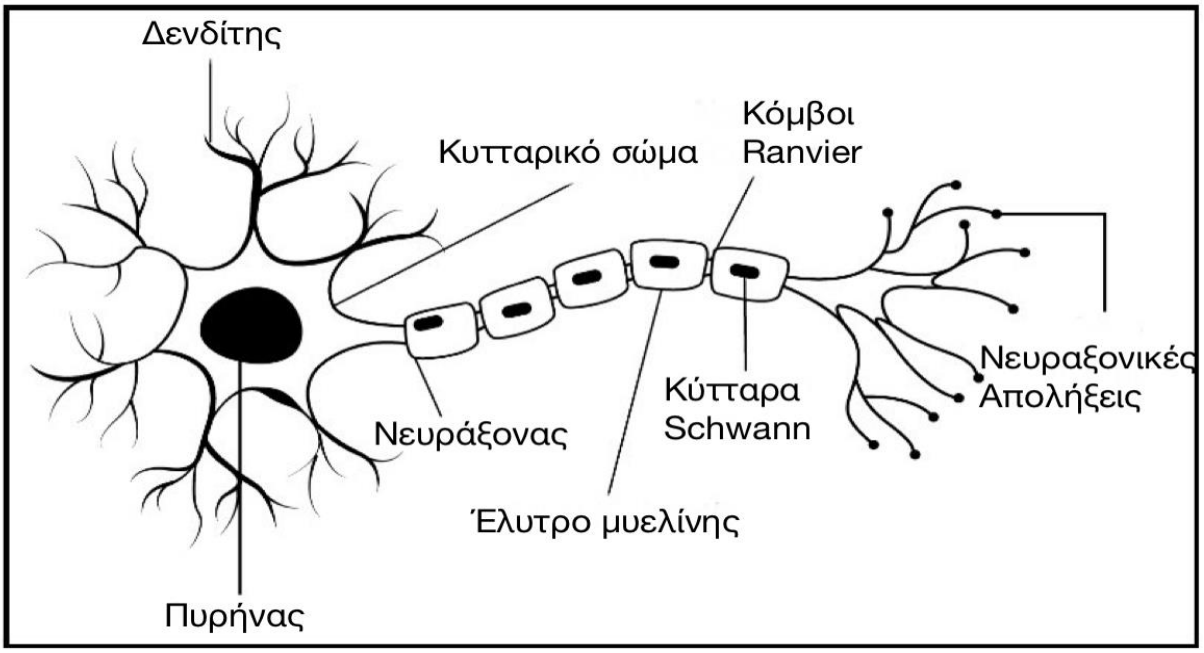
Όλες αυτές οι απειλές μπορούν να αντιμετωπιστούν αποτελεσματικότερα χρησιμοποιώντας αλγορίθμους και εργαλεία τεχνητής νοημοσύνης και μηχανικής μάθησης [25].

Μια από τις πρώτες και πιο πετυχημένες εφαρμογές της τεχνητής νοημοσύνης στο πεδίο της κυβερνοασφάλειας είναι ο εντοπισμός της ανεπιθύμητης αλληλογραφίας. Οι στρατηγικές που εφαρμόζονται για την επιτυχημένη αναγνώρισή της είναι αρκετές και διαφέρουν μεταξύ τους, η πιο κοινή και απλή από αυτές είναι η χρησιμοποίηση των Νευρωνικών

Δικτύων (*Neural Networks NNs*) στην πιο βασική τους μορφή, το νευρώνας Perceptron, (ένα υπολογιστικό μοντέλο ή μηχανή που αναπαριστά την ικανότητα του εγκεφάλου να αναγνωρίζει και να διακρίνει)[26].

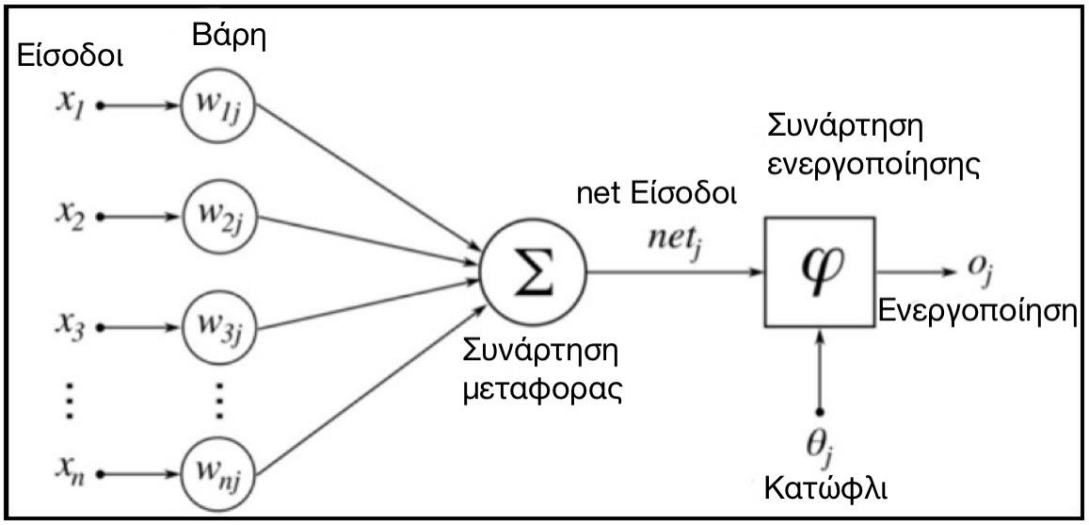
1.5 Εισαγωγή στον νευρώνα Perceptron

Το κοινό χαρακτηριστικό όλων των νευρωνικών δικτύων, ανεξάρτητα από την εκτέλεση και την πολυπλοκότητά τους, είναι ότι προσπαθούν να μιμηθούν τη συμπεριφορά του ανθρώπινου εγκεφάλου. Η πιο βασική δομή που συναντάται κατά την ανάλυση της συμπεριφοράς του εγκεφάλου είναι αναμφισβήτητα ο νευρώνας. Το perceptron είναι μια από τις πρώτες επιτυχημένες εκφράσεις ενός νευρώνα στο πλαίσιο της τεχνητής νοημοσύνης Εικόνα 1. Όπως ακριβώς ένας νευρώνας στον ανθρώπινο εγκέφαλο, προσπαθεί να συνδέσει ένα αποτέλεσμα εξόδου με συγκεκριμένες εισόδους, με τον ίδιο τρόπο, η τεχνητή αναπαράσταση του νευρώνα Perceptron [27], είναι δομημένο με τέτοιο τρόπο ώστε να συνδέει μια δοσμένη τιμή εξόδου με μια ή περισσότερες τιμές εισόδων, Εικόνα 2.



Πηγή: Perceptron. What is a Perceptron https://deepai.org/machine-learning-glossary-and-terms/perceptron#:~:text=A%20Perceptron%20is%20an%20algorithm,a%20single%2Dlayer%20neural%20network.

Εικόνα 1. Νευρώνας Percepton



Πηγή: Davies, Melissa (2002-04-09). "The Neuron: size comparison". Neuroscience: A journey through the brain. Retrieved 2009-06-20. Sweet Spot. Saniya Parveez. Underfitting & Overfitting — The Thwarts of Machine Learning Models'Accuracy https://towardsai.net/p/machine-learning/underfitting-overfitting%E2%80%8A-%E2%80%8Athe-thwarts-of-machine-learning-models%E2%80%8Baccuracy

Εικόνα 2. Νευρωνικό Δίκτυο εισόδου εξόδου με συνάρτηση ενεργοποίησης

Ο μηχανισμός ο οποίος μετασχηματίζει τα δεδομένα εισόδου σε μια τιμή εξόδου εκφράζεται χρησιμοποιώντας το κατάλληλο ''βάρος'' στις τιμές εισόδου, οι οποίες συντίθενται και προωθούνται σε έναν αλγόριθμο ενεργοποίησης, ο οποίος όταν ξεπεραστεί ένα συγκεκριμένο κατώφλι τιμής, παράγει μια τιμή εξόδου η οποία προωθείται στα υπόλοιπα μέρη του νευρωνικού δικτύου (Εικόνα 2).

1.6 Φίλτρα ανεπιθύμητης αλληλογραφίας

Τα φίλτρα ανεπιθύμητης αλληλογραφίας λειτουργούν ως εξής: Για το διαχωρισμό των μηνυμάτων που λαμβάνονται, η κατηγοριοποίησή τους γίνεται ανάλογα με την απουσία ή την παρουσία συγκεκριμένων λέξεων κλειδιών που εμφανίζονται στο κείμενό τους με συγκεκριμένη συχνότητα [28]. Αποδίδεται λοιπόν, μια τιμή σε κάθε ξεχωριστό μήνυμα που χαρακτηρίζεται ως ανεπιθύμητο, βασισμένη στο αριθμό των εμφανίσεων των αναγνωρισμένων λέξεων κλειδιών. Αυτή η τιμή αποτελεί επίσης σημείο αναφοράς για την κατηγοριοποίηση μελλοντικών μηνυμάτων ηλεκτρονικού ταχυδρομείου. Επομένως η χρήση ενός κατωφλίου τιμής για τον διαχωρισμό της ανεπιθύμητης αλληλογραφίας είναι απαραίτητη. Αν δηλαδή η τιμή που έχει αποδοθεί σε ένα μήνυμα, ξεπερνάει την τιμή του κατωφλίου, το μήνυμα κατηγοριοποιείται αυτόματα ως ανεπιθύμητο, αλλιώς ως κανονικό. Η τιμή του κατωφλίου πρέπει συνεχώς να επαναπροσδιορίζεται ώστε να λαμβάνει υπ' όψη νέους τύπους ανεπιθύμητων μηνυμάτων που θα συναντηθούν στο μέλλον. Μια εξίσου σημαντική εργασία για την σωστή λειτουργία του αλγορίθμου είναι ο ορθός ορισμός του βάρους κάθε λέξης κλειδί που παρουσιάζεται στο κείμενο του μηνύματος, έτσι ώστε να είναι επαρκώς αντιπροσωπευτική η πιθανότητα τα μήνυμα αυτό να είναι ανεπιθύμητο [29].

Πρέπει επίσης να ληφθεί υπ’ όψη το γεγονός ότι οι δημιουργοί κακόβουλων μηνυμάτων ηλεκτρονικού ταχυδρομείου και λογισμικού γνωρίζουν την προσπάθεια που πραγματοποιείται για το φιλτράρισμα των μηνυμάτων, οπότε θα προσπαθήσουν και εκείνοι να προσαρμοστούν σε καινούριες στρατηγικές και τεχνικές με σκοπό να εξαπατήσουν τους ανθρώπους αλλά και τα φίλτρα της ανεπιθύμητης αλληλογραφίας.

ΚΕΦΑΛΑΙΟ 2

ΕΠΙΤΗΡΟΥΜΕΝΗ ΜΑΘΗΣΗ

Η επιτηρούμενη μάθηση είναι από τους πιο πετυχημένος τύπους μηχανικής μάθησης. Σε αυτό το κεφάλαιο θα περιγράψουμε την επιτηρούμενη μάθηση με περισσότερη λεπτομέρεια και θα εξηγήσουμε αρκετούς δημοφιλείς αλγόριθμους επιτηρούμενης μάθησης. επιτηρούμενη μάθηση χρησιμοποιείται όταν θέλουμε να προβλέψουμε ένα συγκεκριμένο αποτέλεσμα από μία δοσμένη είσοδο και έχουμε παραδείγματα η ζευγαριών εισόδων-εξόδων. Ένα μοντέλο μηχανικής εκμάθησης το φτιάχνουμε από αυτά τα ζευγάρια εισόδων και εξόδων τα οποία αποτελούν το εκπαιδευτικό μας κομμάτι. Ο στόχος μας είναι να φτιάξουμε έγκυρες προβλέψεις σε νέα δεδομένα τα οποία ο αλγόριθμος δεν έχει ξαναδεί. Η εκμάθηση με επίβλεψη συχνά απαιτεί την βοήθεια του ανθρώπου για να φτιάξει το εκπαιδευτικό κομμάτι αλλά ύστερα αυτοματοποιεί και συχνά επιταχύνει την όλη αυτή χρονοβόρα διαδικασία [30].

2.1 Κατηγοριοποίηση και παλινδρόμηση

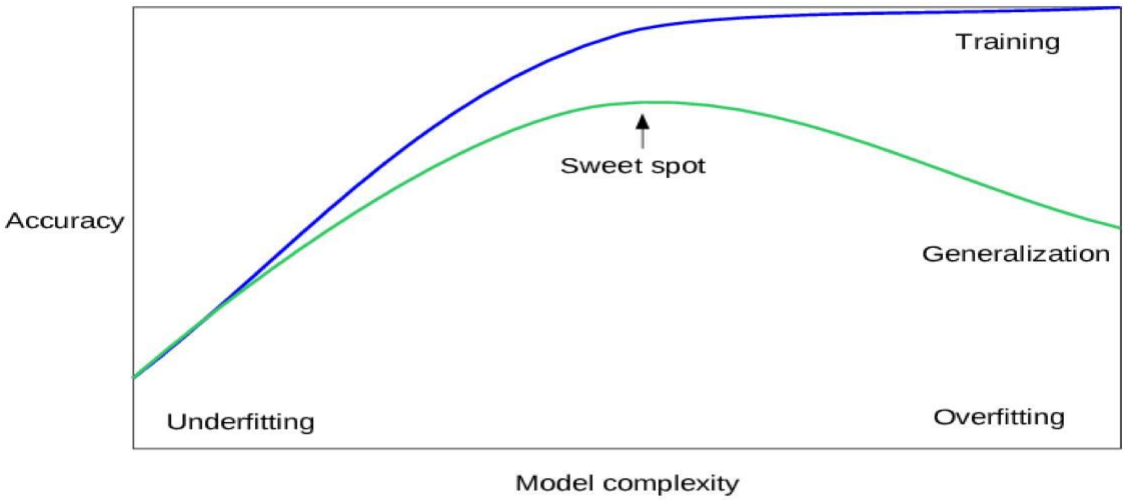
Υπάρχουν δύο βασικοί τύποι αλγορίθμων μηχανικής επιτηρούμενης μάθησης: η κατηγοριοποίηση και η παλινδρόμηση. Στην κατηγοριοποίηση ο σκοπός είναι να προβλέψουμε την ετικέτα τάξης η οποία είναι μία επιλογή από μία προκαθορισμένη λίστα πιθανοτήτων. Η κατηγοριοποίηση συχνά χωρίζεται σε δυαδική κατηγοριοποίηση η οποία είναι μια ειδική περίπτωση διάκρισης μεταξύ δύο τάξεων και η πολύ-ταξική κατηγοριοποίηση η οποία είναι κατηγοριοποίηση με πάνω από δύο τάξεις. Σε κάθε πρόβλημα δυαδικής κατηγοριοποίησης πρέπει να απαντήσουμε μια ερώτηση τύπου «ΝΑΙ ή ΟΧΙι» [25].

Η κατηγοριοποίηση της αλληλογραφίας σε ανεπιθύμητη είτε σε επιθυμητή είναι ένα παράδειγμα προβλήματος δυαδικής κατηγοριοποίησης. Σε αυτό το πρόβλημα δυαδικής κατηγοριοποίησης το ερώτημα που πρέπει να απαντηθεί με ΝΑΙ ή ΟΧΙ είναι «*Είναι αυτό το μήνυμα ανεπιθύμητο*». Στην δυαδική κατηγοριοποίηση συχνά αναφέρουμε την μία τάξη σαν θετική και την άλλη σαν αρνητική τάξη. Βέβαια η θετική τάξη δε σημαίνει ότι έχει πλεονέκτημα η μεγαλύτερη αξία αλλά επεξηγεί τι είναι το αντικείμενο μελέτης μας. Ένα παράδειγμα ταξικής κατηγοριοποίησης είναι η πρόβλεψη της γλώσσας στην οποία είναι γραμμένη μία ιστοσελίδα. Οι τάξεις εδώ είναι μία λίστα από προκαθορισμένες πιθανές γλώσσες [31].

Στην περίπτωση της παλινδρόμησης ο στόχος είναι να προβλεφθεί ένας συνεχής αριθμός. Η πρόβλεψη του ετήσιου εισοδήματος ενός ανθρώπου από την μόρφωσή του, την ηλικία του

και το που κατοικεί είναι ένα παράδειγμα προβλήματος παλινδρόμησης. Ένας εύκολος τρόπος για το διαχωρισμό μεταξύ της κατηγοριοποίησης και των προβλημάτων παλινδρόμησης είναι να θέσουμε το ερώτημα εάν υπάρχει κάποιος τύπος διάταξης ή συνέχειας του αποτελέσματος. Αν υπάρχει διάταξη ή συνέχεια μεταξύ των πιθανών αποτελεσμάτων τότε το πρόβλημα είναι πρόβλημα παλινδρόμησης [32].

Στην επιτηρούμενη μάθηση θέλουμε να φτιάξουμε μοντέλα πάνω στα δεδομένα εκπαίδευσης και μετά να μπορούμε να κάνουμε ακριβής προβλέψεις, σε νέα δεδομένα τα οποία έχουν τα ίδια χαρακτηριστικά με αυτά που χρησιμοποιήσαμε στην εκπαίδευση. Αν ένα μοντέλο είναι ικανό να κάνει ακριβείς προβλέψεις σε δεδομένα τα οποία δεν έχει ξαναδεί τότε θεωρούμε ότι είναι ικανό να γενικεύσει από τα εκπαιδευτικά δεδομένα στα δεδομένα δοκιμών. Θέλουμε να φτιάξουμε ένα μοντέλο το οποίο είναι ικανό να γενικεύει όσο το δυνατόν καλύτερα. Συνήθως φτιάχνουμε το μοντέλο κατά τέτοιον τρόπο ώστε να μπορεί να κάνει ακριβείς προβλέψεις στα εκπαιδευτικά δεδομένα. Αν τα εκπαιδευτικά δεδομένα και τα δεδομένα δοκιμών έχουνε αρκετά κοινά τότε περιμένουμε και το μοντέλο επίσης να είναι ακριβές στα δεδομένα δοκιμών [33]. Η δημιουργία ενός περίπλοκου μοντέλου το οποίο τα πάει καλά με τα εκπαιδευτικά δεδομένα αλλά δεν γενικεύει τα καινούργια δεδομένα είναι το πιο σύνηθες πρόβλημα που μπορούμε να αντιμετωπίσουμε, το οποίο είναι γνωστό ως *overfitting*. Η αποφυγή αυτού του κρίσιμου προβλήματος είναι αρκετά σημαντική στην δημιουργία ενός επιτυχημένου μοντέλου μηχανικής εκμάθησης. Ο καλύτερος τρόπος για να το αντιμετωπίσουμε είναι να περιοριστούμε στην δημιουργία πολύ απλών μοντέλων. Από την άλλη πλευρά, *underfitting*, Εικόνα 3, είναι το πρόβλημα κατά το οποίο δεν γίνεται επαρκής επεξήγηση για τον στόχο εξόδου από τα εκπαιδευτικά δεδομένα [34].



Πήγη: Sweet Spot. Saniya Parveez. Underfitting & Overfitting — The Thwarts of Machine Learning Models'Accuracy https://towardsai.net/p/machine-learning/underfitting-overfitting%E2%80%8A-%E2%80%8Athe-thwarts-of-machine-learning-models%E2%80%8Baccuracy

Εικόνα 3. Underfitting & Overfitting του μοντέλου της μηχανικής μάθησης

Αν επιλέξουμε να χρησιμοποιήσουμε ένα μοντέλο το οποίο είναι πολύ απλό, η απόδοση με τα εκπαιδευτικά δεδομένα δεν θα είναι καλή και επίσης στο ίδιο φάσμα θα βρίσκεται και η απόδοση με τα δεδομένα δοκιμών. Όσο περιπλοκότερο επιτρέπουμε στο μοντέλο μας να είναι τόσο καλύτερα και με μεγαλύτερη ακρίβεια θα μπορούμε να προβλέπουμε τα εκπαιδευτικά δεδομένα. Ωστόσο αν το μοντέλο μας είναι αρκετά πολύπλοκο ξεκινάμε να εστιάζουμε περισσότερο από ότι θα έπρεπε στις ιδιαιτερότητες των εκπαιδευτικών δεδομένων και έτσι το μοντέλο δεν θα γενικεύει με τον καλύτερο τρόπο τα νέα δεδομένα. Παρόλα αυτά υπάρχει η χρυσή τομή στην πολυπλοκότητα, η οποία μας παρέχει την καλύτερη απόδοση γενίκευσης. Αυτό ακριβώς είναι το μοντέλο που ψάχνουμε να βρούμε [35].

2.2 Αλγόριθμοι μηχανικής επιτηρούμενης μάθησης

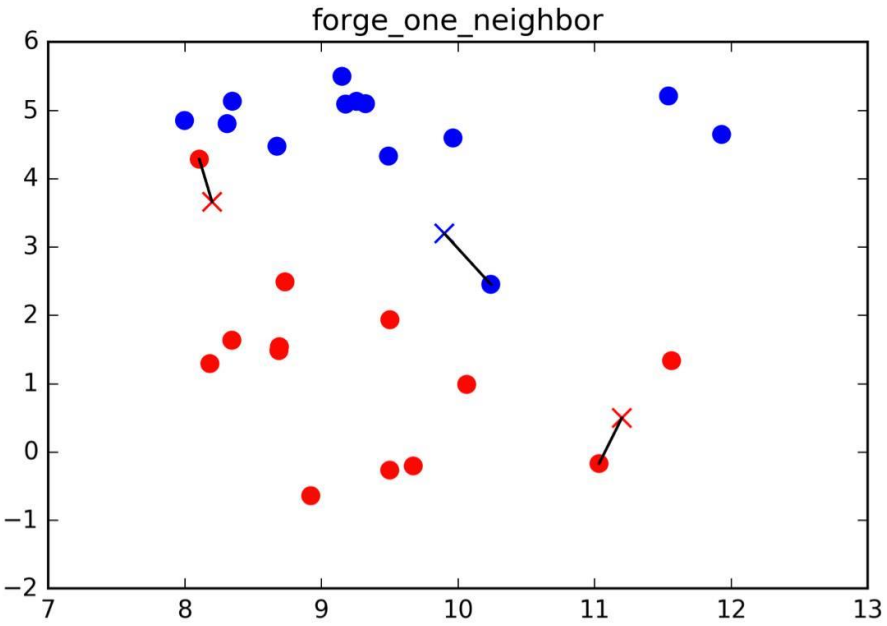
Στη συνέχεια θα δούμε τους πιο δημοφιλείς αλγόριθμους μηχανικής εκμάθησης και θα εξηγήσουμε πως μαθαίνουν από τα δεδομένα και πώς κάνουν τις προβλέψεις τους.

2.3 k-Nearest Neighbor

Ο αλγόριθμος *K-Nearest Neighbor* είναι ο πιο απλός αλγόριθμος μηχανικής μάθησης. Η δημιουργία ενός μοντέλου αποτελείται μόνο από την αποθήκευση των εκπαιδευτικών δεδομένων. Για να κάνει μία πρόβλεψη σε ένα καινούργιο δεδομένο ο αλγόριθμος βρίσκει τα κοντινότερα δεδομένα προς το καινούριο από τα εκπαιδευτικά δεδομένα δηλαδή τους κοντινότερους γείτονές του (*nearest neighbors*) [36].

2.3.1 k-Neighbors κατηγοριοποίηση

Είναι η απλούστερη μορφή, ο αλγόριθμος λαμβάνει υπόψη μόνο έναν κοντινότερο γείτονα ο οποίος είναι το κοντινότερο εκπαιδευτικό δεδομένο στο σημείο που επιθυμούμε να κάνουμε την πρόβλεψη. Η πρόβλεψη είναι απλά η έξοδος που θα προκύψει από το σημείο εκπαίδευσης, Εικόνα 4.

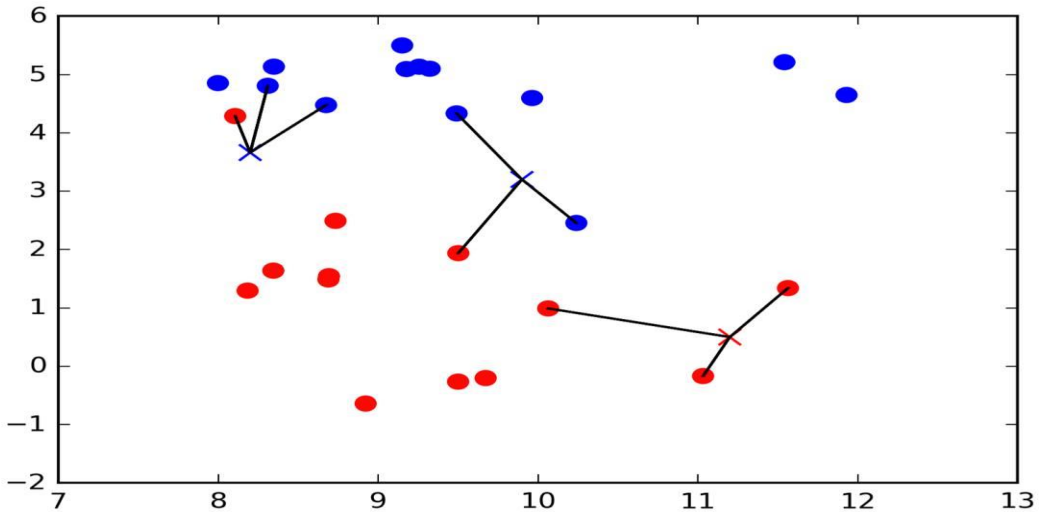


Πηγή: Parisi, A. (n.d.). Artificial Intelligence for Cybesecurity.

Εικόνα 4. k-Neighbors κατηγοριοποίηση για k=1

Στο παραπάνω παράδειγμα προσθέσαμε τρία καινούργια σημεία δεδομένων τα οποία φαίνονται στον παραπάνω πίνακα σαν σταυροί. Για κάθε έναν από αυτούς σημειώθηκε με το κοντινότερο σημείο των εκπαιδευτικών δεδομένων. Η πρόβλεψη του αλγορίθμου με έναν κοντινότερο γείτονα είναι η ταμπέλα του σημείου που φαίνεται από το χρώμα του σταυρού [38]. Μπορούμε επίσης αντί να συμπεριλάβουμε μόνο τον κοντινότερο γείτονα να συμπεριλάβουμε έναν τυχαίο αριθμό γειτόνων (k) από που προκύπτει και το όνομα του αλγορίθμου k nearest neighbor. Όταν λαμβάνουμε υπόψιν περισσότερους από έναν γείτονες τότε στην ουσία είναι σαν να ψηφίζουμε για το την ταμπέλα θα πάρει κάθε σημείο. Αυτό σημαίνει ότι για κάθε σημείο μετράμε πόσοι γείτονες είναι κόκκινοι και πόσοι είναι μπλε και στο τέλος το σημείο θα πάρει την κλάση η οποία είναι πιο συχνή, με άλλα λόγια η κλάση που έχει την πλειοψηφία ανάμεσα στους κ γείτονες [37].

Παρακάτω είναι μία απεικόνιση χρησιμοποιώντας τη μέθοδο των τριών πλησιέστερων γειτόνων όπου ξανά η πρόβλεψη φαίνεται από το χρώμα το σταυρού.



Πηγή: A. (n.d.). Artificial Intelligence for Cybesecurity.

Εικόνα 5. k-Neighbors κατηγοριοποίηση για k=3

Ενώ αυτή η απεικόνιση είναι για πρόβλημα δυαδικής κατηγοριοποίησης, λειτουργεί ακριβώς με τον ίδιο τρόπο για οποιονδήποτε αριθμό κλάσεων. Για περισσότερες κλάσεις μετρούνται πόσοι γείτονες ανήκουν σε κάθε κλάση και έπειτα προβλέπεται η πιο συχνή από αυτές. Παρακάτω θα δούμε πως μπορούμε να εφαρμόσουμε στην πράξη αλγορίθμους κοντινότερων γειτόνων χρησιμοποιώντας την βιβλιοθήκη *scikit – learn*.

Αρχικά, γίνεται ο διαχωρισμός των δεδομένων σε εκπαιδευτικά και σε δεδομένα δοκιμών ώστε να μπορεί να γίνει αξιολόγηση της γενίκευσης του αλγορίθμου. Έπειτα, εισάγεται η κλάση. Σε αυτό το σημείο θέτονται οι παράμετροι, για παράδειγμα πόσοι γείτονες θα χρησιμοποιηθούν. Στη συνέχεια προσαρμόζουμε τον κατηγοριοποιητή χρησιμοποιώντας τα εκπαιδευτικά δεδομένα. Για τον *KNeighborsClassifier* αυτό σημαίνει ότι θα πρέπει να αποθηκεύσει τα δεδομένα ώστε να είναι σε θέση να υπολογίσει τους γείτονες κα τα τη διάρκεια της πρόβλεψης. Για να γίνουν οι προβλέψεις πάνω στα δεδομένα δοκιμών, είναι απαραίτητο το κάλεσμα της μεθόδου πρόβλεψης *clf.predict*, η οποία υπολογίζει τους κοντινότερους γείτονες στα εκπαιδευτικά δεδομένα και βρίσκει την πιο κοινή κλάση ανάμεσά τους. Για να υπολογιστεί πόσο καλά το μοντέλο γενικεύει χρησιμοποιείται η συνάρτηση *clf.score(X_test, y_test)*. Το συγκεκριμένο μοντέλο έχει περίπου 86% ευστοχία, που σημαίνει ότι το μοντέλο προέβλεψε σωστά για το 86% των δειγμάτων στα δεδομένα δοκιμών τα οποία δεν είχε ξαναδεί όπως φαίνεται παρακάτω στο *Κομμάτι Κώδικα 1* .

```
>>from sklearn.model_selection import train_test_split
    X, y = mglear.datasets.make_forge()
```

```
    X_train , X_test, y_train, y_test = train_test_split(X , y
, random_state=0)

>>from sklearn.neighbors import KNeighborsClassifier
    clf = KNeighborsClassifier( n_neighbors = 3)
>>clf.fit(X_trtain , y_train)

     KNeighborsClassifier(algorithm='auto' , leaf_size = 30,
metric = 'minkowski'

     metric_params = None , n_jobs = 1, n_neighbors = 3 , p= 2

     weights = 'uniform'

>>clf.predict(X_test)

>>Array([1, 0 , 1 , 0 , 1 , 0 , 0 ])

>>clf.score(X_test, y_test)

>> 0.8571428571428571
```

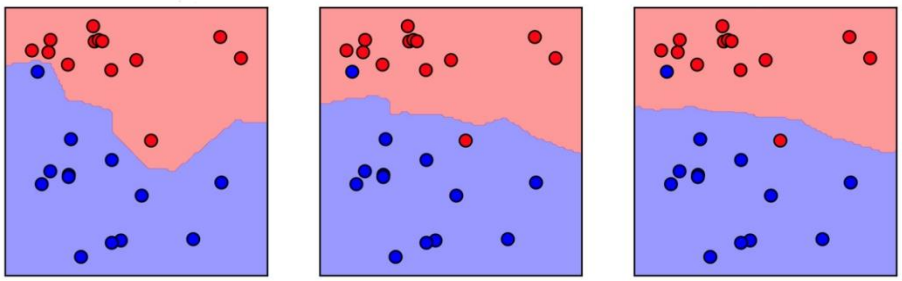
Πηγή: Parisi, A. (n.d.). Artificial Intelligence for Cybesecurity

Κομμάτι Κώδικα 1.

2.3.2 Ανάλυση της k-Neighbors κατηγοριοποίησης

Για τα δεδομένα δύο διαστάσεων, η πρόβλεψη μπορεί να εικονογραφηθεί για όλα τα πιθανά σημεία δοκιμών σε ένα επίπεδο xy. Με κόκκινο χρώμα καλύπτεται το επίπεδο όπου τα σημεία θα έπρεπε να ανήκουν στην κόκκινη κλάση και με το μπλε χρώμα τα αντίστοιχα.

Παρακάτω ακολουθεί μια παρουσίαση του ορίου απόφασης για έναν, τρεις και πέντε γείτονες.



Πηγή: Parisi, A. (n.d.). Artificial Intelligence for Cybesecurity

Εικόνα 1. *K= 1 γείτονας, K= 3 γείτονες, K= 5 γείτονες*

Όπως φαίνεται στην αριστερή εικόνα, η χρησιμοποίηση ενός γείτονα έχει ως αποτέλεσμα ένα όριο απόφασης το οποίο ακολουθεί τα δεδομένα εκπαίδευσης προσεκτικά. Καθώς συμπεριλαμβάνονται όλο και περισσότεροι γείτονες το όριο απόφασης τείνει να γίνει ομαλότερο. Ένα ομαλό όριο αντιστοιχεί σε ένα απλό μοντέλο. Με άλλα λόγια, όσο λιγότεροι

γείτονες χρησιμοποιούνται τόσο πιο πολύπλοκο είναι το μοντέλο ενώ με περισσότερους γείτονες το μοντέλο γίνεται απλούστερο [38].

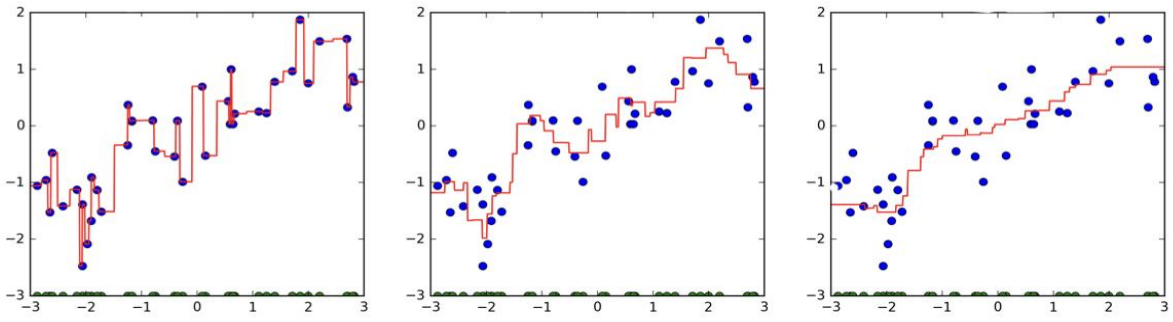
2.3.3 Ανάλυση της k nearest neighbors παλινδρόμησης

Για τα δεδομένα μίας διάστασης, μπορούμε να δούμε ποιες θα είναι οι προβλέψεις για όλες τις πιθανές τιμές. Έτσι δημιουργούμε ένα σύνολο δεδομένων εκπαίδευσης που περιλαμβάνει πολλά σημεία σε μία γραμμή χρησιμοποιώντας τον παρακάτω κώδικα.

```
>>fig, axes = plt.subplots(1, 3, figsize=(15, 4))
# Δημιουργία 1000 σημείων δεδομένων ισοκατανεμημένα μεταξύ #-3
και 3
line = np.linspace(-3, 3, 1000).reshape(-1, 1)
plt.suptitle("nearest_neighbor_regression")
for n_neighbors, ax in zip[1, 3, 9], axes):
# προβλέψεις χρησιμοποιώντας 1, 3 και 9 γείτονες
reg = KNeighborsRegressor (n_neighbors = n_neighbors). fit(X,
y)
ax.plot(X, y, 'o')
ax.plot(X, -3 * np.ones(len(X)), 'o')
ax.plot(line, reg.predict(line))
ax.set_title("%d neighbor(s)" % n_neighbors)
```

Parisi, A. (n.d.). Artificial Intelligence for Cybesecurity.

Κομμάτι Κώδικα 2.



Parisi, A. (n.d.). Artificial Intelligence for Cybesecurity.

Εικόνα 2. Κ=1 γείτονας, Κ=3 γείτονες, Κ=9 γείτονες.

Στα παραπάνω γραφήματα, τα μπλε σημεία είναι οι απαντήσεις για τα δεδομένα εκπαίδευσης, ενώ η κόκκινη γραμμή είναι η πρόβλεψη που έκανε το μοντέλο για όλα τα σημεία πάνω στη γραμμή .

Χρησιμοποιώντας μόνο έναν γείτονα, κάθε σημείο από τα δεδομένα εκπαίδευσης έχει εμφανή επίδραση πάνω στις προβλέψεις, και οι τιμές που προβλέφθηκαν περνάνε πάνω από όλα τα σημεία δεδομένων. Αυτό οδηγεί σε μία αρκετά ασταθή πρόβλεψη. Όπως είδαμε

και προηγουμένως, όσο περισσότερους γείτονες λάβουμε υπ' όψη τόσο πιο ομαλή θα είναι η πρόβλεψη αλλά δεν θα περνάει πάνω από τα σημεία των δεδομένων εκπαίδευσης [39].

2.3.4 Πλεονεκτήματα, μειονεκτήματα και παράμετροι

Αρχικά, υπάρχουν δύο βασικοί παράμετροι όσο αφορά την *k-Neighbors* κατηγοριοποίηση: ο αριθμός των γειτόνων και ο τρόπος μέτρησης της απόστασης μεταξύ των σημείων των δεδομένων. Στην πράξη, η χρησιμοποίηση ενός μικρού αριθμού γειτόνων 3 ή 5 συνήθως δουλεύει αρκετά καλά, όμως αυτός ο αριθμός πρέπει να επαναπροσδιορίζεται ανάλογα την περίσταση [40].

Ένα από τα πλεονεκτήματα του *nearest neighbors* μοντέλου είναι η ευκολία του στην κατανόηση και συχνά δίνει λογικά αποτελέσματα χωρίς πολλές διευθετήσεις. Το χτίσιμο του συγκεκριμένου μοντέλου είναι τις περισσότερες φορές αρκετά γρήγορο όμως, όταν το σύνολο των δεδομένων εκπαίδευσης είναι αρκετά μεγάλο, η πρόβλεψη μπορεί να είναι αργή. Ένα ακόμα μειονέκτημα είναι ότι το μοντέλο αυτό συχνά δεν αποδίδει καλά σε σύνολα δεδομένων με πολλά χαρακτηριστικά. Οπότε ενώ ο αλγόριθμος είναι αρκετά εύκολος στην κατανόηση, δεν χρησιμοποιείται συχνά στην πράξη, λόγω των αργών προβλέψεων και της ανικανότητάς του να χειρίζεται αρκετά χαρακτηριστικά [41].

2.4 Γραμμικά μοντέλα

Τα γραμμικά μοντέλα είναι μια κλάση μοντέλων τα οποία χρησιμοποιούνται αρκετά στην πράξη, έχουν μελετηθεί αρκετά τις τελευταίες δεκαετίες και οι ρίζες τους βρίσκονται πάνω από εκεί χρόνια πριν. Τα γραμμικά μοντέλα είναι μοντέλα τα οποία κάνουν προβλέψεις και χρησιμοποιούν μια γραμμική συνάρτηση των χαρακτηριστικών εισόδου [42].

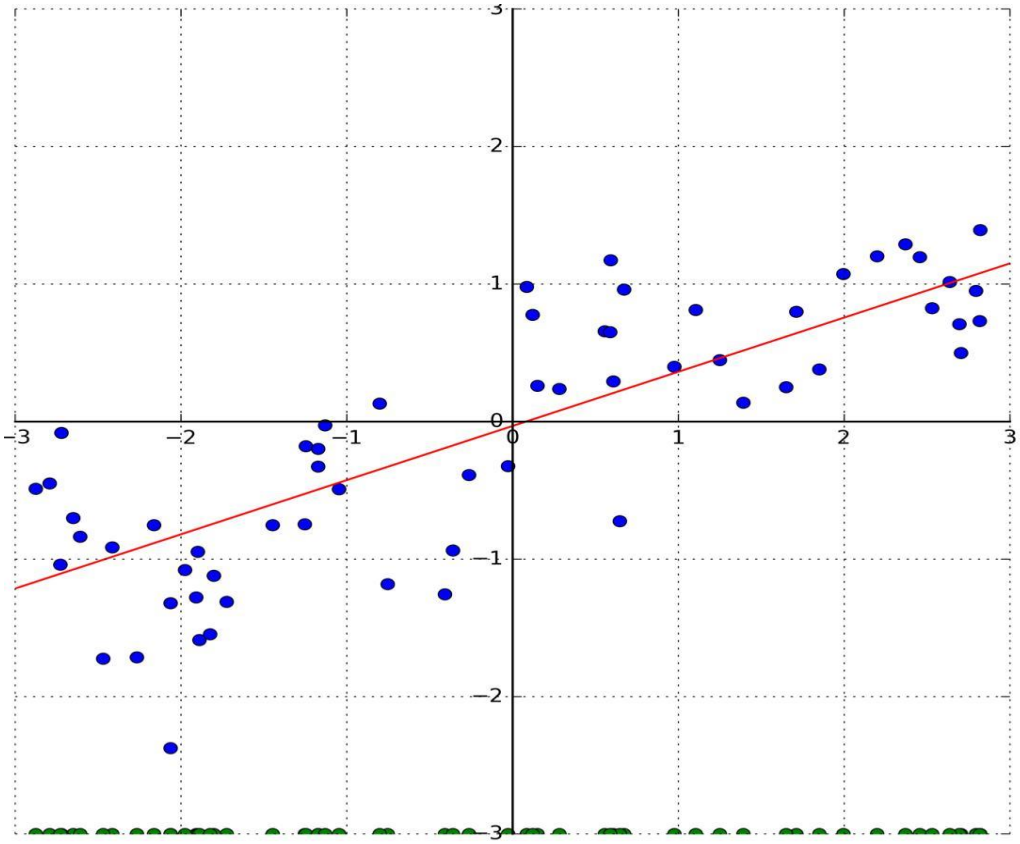
2.4.1 Γραμμικά μοντέλα παλινδρόμησης

Για την παλινδρόμηση, ο γενικός τύπος της πρόβλεψης ενός γραμμικού μοντέλου είναι ο ακόλουθος:

$$\mathbf{y = w[0]x[0] + w[1]x[1] + \ldots + w[p]x[p] + b}$$

Από το **x[0]** έως το **x[p]** δηλώνονται τα χαρακτηριστικά ενός σημείου δεδομένων, στην προκειμένη περίπτωση ο αριθμός των χαρακτηριστικών είναι p. Ως **w** και **b** ορίζονται οι παράμετροι του μοντέλου οι οποίοι έχουν αφομοιωθεί και ως **y** ορίζεται η πρόβλεψη που θα κάνει το μοντέλο. Για ένα σύνολο δεδομένων με ένα μόνο χαρακτηριστικό, η συνάρτηση είναι μια ευθεία γραμμή με κλίση **w[0]** και μετατόπιση του άξονα των **y** με τιμή **b**. Για περισσότερα χαρακτηριστικά, η **w** μεταβλητή αποτελεί τις επιμέρους κλίσεις κάθε χαρακτηριστικού. Με αλλά λόγια η πρόβλεψη αποτελεί ένα σύνολο βαρών των

χαρακτηριστικών εισόδου, με βάρη δοσμένα από τις τιμές του **w** [43]. Η βάση δεδομένων μιας διάστασης οδηγεί στην παρακάτω γραμμή :



Soma Halder, S. O. (2018). Machine Learning for Cybersecurity . Birmingham: Packt.

Εικόνα 3. Ευθεία από τη βάση δεδομένων μιας διάστασης

Σε περίπτωση σύγκρισης των προβλέψεων που προέκυψαν από την κόκκινη γραμμή με τις αντίστοιχες της μεθόδου *KNeighbors Regression* καταλήγουμε στο συμπέρασμα ότι η χρησιμοποίηση μιας ευθείας γραμμής για να κάνουμε προβλέψεις είναι αρκετά περιοριστικό καθώς όλες οι λεπτομέρειες των δεδομένων χάνονται. Στην πραγματικότητα οι βάσεις δεδομένων με ένα χαρακτηριστικό δίνουν μια πιο λοξή προοπτική από την κανονική, ενώ αντίστοιχα οι βάσεις δεδομένων με πολλά χαρακτηριστικά μπορούν να εκμεταλλευτούν πολύ περισσότερο τα γραμμικά μοντέλα. Ειδικότερα, αν τα χαρακτηριστικά είναι περισσότερα από σημεία εκπαιδευτικών δεδομένων τότε κάθε **y** μπορεί να μοντελοποιηθεί τέλεια σαν μια γραμμική συνάρτηση [43].

2.4.2 Γραμμική παλινδρόμηση

Η γραμμική παλινδρόμηση είναι η απλούστερη και πιο κλασική γραμμική μέθοδος παλινδρόμησης. Βρίσκει τις παραμέτρους **w** και **b** οι οποίες ελαχιστοποιούν το μέσο

τετραγωνικό σφάλμα μεταξύ των προβλέψεων και των πραγματικών στόχων της παλινδρόμησης των εκπαιδευτικών δεδομένων. Το μέσο τετραγωνικό σφάλμα είναι το άθροισμα της διαφοράς των τετραγώνων μεταξύ των προβλέψεων και των πραγματικών τιμών. Η γραμμική παλινδρόμηση δεν έχει παραμέτρους, το οποίο είναι ένα μεγάλο πλεονέκτημα, αλλά όμως δεν έχει τρόπο να ελέγξει την πολυπλοκότητα των μοντέλων [42].

2.4.3 Γραμμικά μοντέλα για κατηγοριοποίηση

Τα γραμμικά μοντέλα χρησιμοποιούνται εκτενώς για κατηγοριοποίηση. Ξεκινώντας λοιπόν με τη δυαδική κατηγοριοποίηση, η πρόβλεψή της προκύπτει από τον παρακάτω τύπο :

y = w[0]x[0] + w[1]x[1] + ... + w[p]x[p] + b > 0.

Ο παραπάνω τύπος μοιάζει με αυτόν της γραμμικής παλινδρόμησης, αλλά αντί να επιστρέφει το σταθμισμένο άθροισμα των χαρακτηριστικών, έχει κατώφλι της προβλεπόμενης τιμής το μηδέν. Οπότε αν η συνάρτηση είναι μικρότερη του μηδέν, η πρόβλεψη είναι η κλάση -1 , ενώ αν είναι μεγαλύτερη του μηδέν, η πρόβλεψη είναι η κλάση +1. Αυτός ο κανόνας πρόβλεψης είναι συνήθης σε όλα τα γραμμικά μοντέλα κατηγοριοποίησης. Για τα γραμμικά μοντέλα παλινδρόμησης, η έξοδος **y** ήταν μια γραμμική συνάρτηση των χαρακτηριστικών. Για τα γραμμικά μοντέλα κατηγοριοποίησης, το όριο απόφασης είναι μια γραμμική συνάρτηση των εισόδων.

Υπάρχουν αρκετοί αλγόριθμοι για την εκμάθηση γραμμικών μοντέλων. Οι αλγόριθμοι αυτοί διαφέρουν σε 2 σημαντικά σημεία :

1. Ως προς τον τρόπο που μετράνε ποσό καλά ένας συγκεκριμένος συνδυασμός συντελεστών ταιριάζει στα δεδομένα εκπαίδευσης .
2. Αν χρησιμοποιηθεί και ποιον τρόπο συστηματοποίησης

Οι δυο πιο βασικοί αλγόριθμοι γραμμικής κατηγοριοποίησης είναι η *logistic regression* που εκφράζεται ως *linear_model.LogisticRegression* και η *linear support vector machines (linear SVMs)* που εκφράζεται ως *svm.LinearSVC*. Παρά το όνομά της η *logistic regression* είναι αλγόριθμος κατηγοριοποίησης και όχι παλινδρόμησης [44].

2.4.4 Πλεονεκτήματα , μειονεκτήματα και παράμετροι

Τα γραμμικά μοντέλα είναι πολύ γρήγορα στην εκπαίδευση και επίσης μπορούν να προβλέψουν ταχύτατα. Αφορούν πολύ μεγάλα σύνολα δεδομένων και δουλεύουν εξίσου καλά και με αραιά δεδομένα. Ένα ακόμα πλεονέκτημα των γραμμικών μοντέλων είναι ότι κάνουν πολύ εύκολο στην κατανόηση πως γίνεται μια πρόβλεψη. Από την άλλη πλευρά, δεν είναι αρκετά συχνά ξεκάθαρο γιατί οι συντελεστές είναι αυτοί που είναι. Αν δηλαδή το σύνολο δεδομένων αποτελείται από αρκετά παρόμοια χαρακτηριστικά, είναι δύσκολο να ερμηνευτούν οι συντελεστές. Τα γραμμικά μοντέλα συνήθως αποδίδουν καλά όταν ο

αριθμός των χαρακτηριστικών είναι μεγαλύτερος από τον αριθμό των δειγμάτων. Χρησιμοποιούνται συχνά επίσης σε πολύ μεγάλα σύνολα δεδομένων, μόνο και μόνο επειδή τα αλλά μοντέλα δεν είναι εφικτό να εκπαιδεύσουν τον αλγόριθμο sciee [45]. Βέβαια, σε μικρότερα σύνολα δεδομένων τα αλλά μοντέλα μπορεί να αποδίδουν καλύτερα.

2.5 Ταξινομητές Naive Bayes

Οι ταξινομητές *Naive Bayes* είναι μια οικογένεια ταξινομητών που μοιάζουν αρκετά με τα γραμμικά μοντέλα που αναφέρθηκαν παραπάνω. Ωστόσο, τείνουν να είναι ακόμη γρηγορότεροι στη διαδικασία της εκπαίδευσης. Το κόστος για αυτή την αποτελεσματικότητά τους είναι ότι τα μοντέλα *Naïve Bayes* παρέχουν συχνά απόδοση γενικοποίησης η οποία είναι ελαφρώς χειρότερη από αυτή των γραμμικών ταξινομητών.

Ο λόγος που τα μοντέλα *Naïve Bayes* είναι τόσο αποτελεσματικά είναι ότι μαθαίνουν παραμέτρους κοιτάζοντας κάθε χαρακτηριστικό ξεχωριστά και συλλέγουν απλές στατιστικές ανά κατηγορία από κάθε χαρακτηριστικό. Υπάρχουν τρία είδη ταξινομητών *Naïve Bayes* που εφαρμόζονται στο *scikit-learn* οι οποίοι είναι: *GaussianNB*, *BernoulliNB* και *MultinomialNB*. Ο ταξινομιτής *GaussianNB* μπορεί να εφαρμοστεί σε οποιαδήποτε συνεχή δεδομένα, ενώ ο *BernoulliNB* υποθέτει δυαδικά δεδομένα και ο *MultinomialNB* υποθέτει δεδομένα καταμέτρησης (που κάθε χαρακτηριστικό αντιπροσωπεύει έναν ακέραιο αριθμό, όπως το πόσο συχνά εμφανίζεται μια λέξη σε μια πρόταση). Τα μοντέλα *BernoulliNB* και *MultinomialNB* χρησιμοποιούνται ως επί το πλείστον στην ταξινόμηση δεδομένων κειμένου. Ο ταξινομητής *BernoulliNB* μετρά πόσο συχνά κάθε χαρακτηριστικό κάθε κλάσης δεν είναι μηδέν.

Τα άλλα δύο μοντέλα *Bayes*, *MultinomialNB* και *GaussianNB* είναι ελαφρώς διαφορετικά όσο αφορά τα είδη των στατιστικών στοιχείων που υπολογίζουν. Το *MultinomialNB* λαμβάνει υπόψη τη μέση τιμή του κάθε χαρακτηριστικού για κάθε κλάση, ενώ το *GaussianNB* αποθηκεύει τη μέση τιμή καθώς και την τυπική απόκλιση κάθε χαρακτηριστικού για κάθε κλάση. Για να γίνει μια πρόβλεψη, ένα σημείο δεδομένων συγκρίνεται με τα στατιστικά στοιχεία για καθεμία από τις κλάσεις, και προβλέπεται η καλύτερη αντιστοίχηση . Είναι ενδιαφέρον το γεγονός ότι για τα μοντέλα *MultinomialNB* και *BernoulliNB*, αυτό οδηγεί σε έναν τύπο πρόβλεψης που έχει την ίδια μορφή με τα γραμμικά μοντέλα [46].

2.5.1 Πλεονεκτήματα, μειονεκτήματα και παράμετροι

Τα μοντέλα *MultinomialNB* και *BernoulliNB* έχουν μία μόνο παράμετρο την άλφα, η οποία ελέγχει την πολυπλοκότητα του μοντέλου. Ο τρόπος λειτουργίας της *άλφα* είναι ότι ο αλγόριθμος προσθέτει στην *άλφα* πολλά εικονικά σημεία δεδομένων στα δεδομένα που έχουν θετικές τιμές για όλα τα χαρακτηριστικά. Αυτό έχει ως αποτέλεσμα σε μια

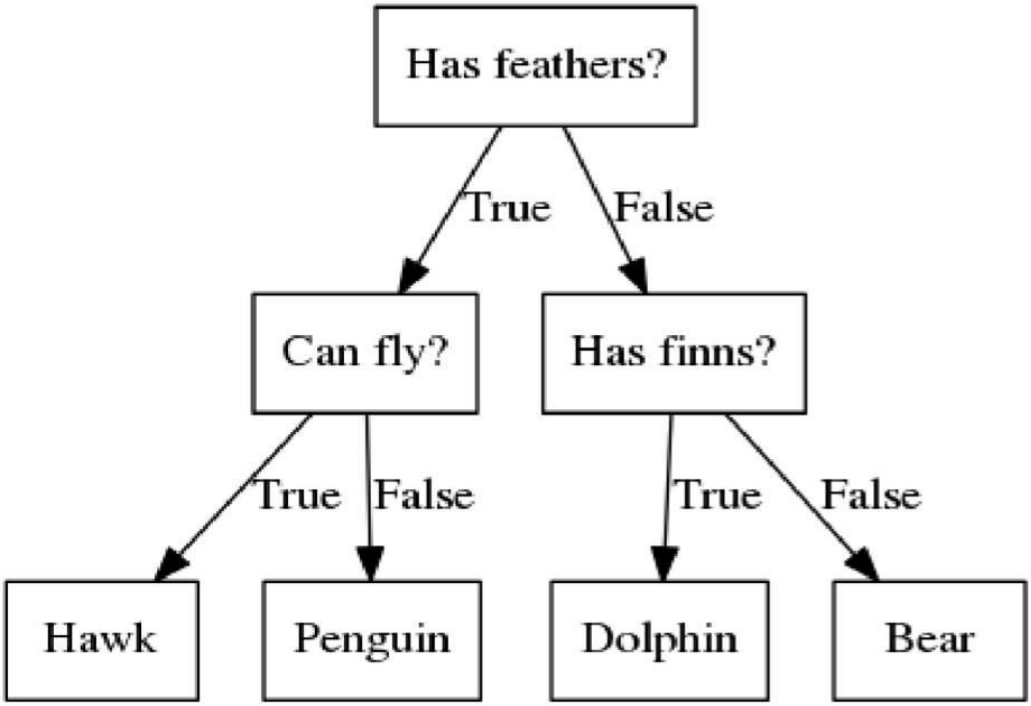
«εξομάλυνση» των στατιστικών. Ένα μεγάλο *άλφα* σημαίνει περισσότερη εξομάλυνση, με αποτέλεσμα λιγότερο περίπλοκα μοντέλα.

Το μοντέλο *GaussianNB* φαίνεται να χρησιμοποιείται σπάνια από του ειδικούς κυβερνοασφάλειας, ενώ οι άλλες δύο παραλλαγές των *Naive Bayes* μοντέλων χρησιμοποιούνται ευρέως για αραιά δεδομένα όπως σε ένα κείμενο. Το *MultinomialNB* συνήθως αποδίδει καλύτερα από το *BinaryNB*, ιδίως σε σύνολα δεδομένων με σχετικά μεγάλο αριθμό μη μηδενικών χαρακτηριστικών (δηλαδή μεγάλα έγγραφα).

Τα μοντέλα *Naïve Bayes* μοιράζονται πολλά από τα πλεονεκτήματα και τα μειονεκτήματα των γραμμικών μοντέλων. Είναι πολύ γρήγορα για εκπαίδευση και πρόβλεψη και η διαδικασία εκπαίδευσης είναι εύκολη στην κατανόηση. Τα μοντέλα λειτουργούν πολύ καλά με αραιά δεδομένα υψηλών διαστάσεων και είναι σχετικά ανθεκτικά στις παραμέτρους. Τα μοντέλα *Naive Bayes* είναι εξαιρετικά βασικά μοντέλα και χρησιμοποιούνται συχνά σε πολύ μεγάλα σύνολα δεδομένων, όπου η εκπαίδευση ενός γραμμικού μοντέλου μπορεί να πάρει αρκετό χρόνο [47].

2.6 Δέντρα απόφασης

Τα δέντρα απόφασης είναι ευρέως χρησιμοποιούμενα μοντέλα για εργασίες κατηγοριοποίησης και παλινδρόμησης. Ουσιαστικά, μαθαίνουν μια ιεραρχία ερωτήσεων «αν-αλλιώς», που οδηγούν σε μια απόφαση. Αυτές οι ερωτήσεις είναι παρόμοιες με τις ερωτήσεις που μπορεί να γίνουν σε ένα παιχνίδι ερωτήσεων. Για παράδειγμα, στην περίπτωση που πρέπει να ξεχωριστούν τα ακόλουθα τέσσερα ζώα: αρκούδες, γεράκια, πιγκουΐνοι και δελφίνια. Ο στόχος είναι να φτάσουμε στη σωστή απάντηση ζητώντας όσο το δυνατόν λιγότερες ερωτήσεις «αν-αλλιώς». Μπορούμε να ξεκινήσουμε ρωτώντας εάν το ζώο έχει φτερά, μια ερώτηση που περιορίζει τα πιθανά ζώα σε δύο μόνο. Εάν η απάντηση είναι ναι, μπορούμε να κάνουμε μια άλλη ερώτηση που θα μπορούσε να βοηθήσει να διακρίνουμε μεταξύ γερακιών και πιγκουΐνων. Για παράδειγμα, θα μπορούσαμε να ρωτήσουμε αν το ζώο μπορεί ή όχι να πετάξει. Εάν το ζώο δεν έχει φτερά, οι πιθανές επιλογές ζώων μας είναι τα δελφίνια και οι αρκούδες, και θα πρέπει να κάνουμε μια ερώτηση για να ξεχωρίσουμε αυτά τα δύο ζώα, για παράδειγμα, ρωτώντας αν το ζώο έχει πτερύγια. Αυτή η σειρά ερωτήσεων μπορεί να εκφραστεί ως δέντρο αποφάσεων, όπως φαίνεται στο παρακάτω σχήμα. Σε αυτήν την απεικόνιση, κάθε κόμβος στο δέντρο αντιπροσωπεύει είτε μια ερώτηση είτε έναν τερματικό κόμβο (ονομάζεται επίσης φύλλο) που περιέχει την απάντηση. Οι άκρες συνδέουν τις απαντήσεις με μια ερώτηση με την επόμενη ερώτηση που θα θελήσουμε [48].



Πηγή: Guido, A. C. (2016). Introduction to Machine Learning with Python. O'REILLY

Εικόνα 4. Δέντρο απόφασης..

Στη γλώσσα της μηχανικής μάθησης, δημιουργήσαμε ένα μοντέλο για τη διάκριση μεταξύ τεσσάρων κατηγοριών ζώων (γεράκια, πιγκουΐνους, δελφίνια και αρκούδες) που χρησιμοποιούν τα τρία χαρακτηριστικά «έχει φτερά»,«Μπορεί να πετάξει» και «έχει πτερύγια». Αντί να κατασκευάσουμε αυτά τα μοντέλα με το χέρι, μπορούμε να τα μάθουμε από δεδομένα χρησιμοποιώντας την εκμάθηση με επίβλεψη [49].

2.6.1 Φτιάχνοντας ένα δέντρο απόφασης

Η εκμάθηση ενός δέντρου αποφάσεων σημαίνει την εκμάθηση μιας ακολουθίας «*αν-αλλιώς*» ερωτήσεων που μας οδηγούν στην αληθινή απάντηση πιο γρήγορα. Στη διαδικασία της μηχανικής μάθησης, αυτές οι ερωτήσεις ονομάζονται δοκιμές (δεν πρέπει να συγχέονται με το σετ δοκιμών, που είναι τα δεδομένα που χρησιμοποιούμε για να ελέγξουμε και να δούμε πόσο γενικεύσιμο είναι το μοντέλο μας). Συνήθως τα δεδομένα δεν έχουν τη μορφή δυαδικών χαρακτηριστικών ναι ή όχι όπως στο παράδειγμα με τα ζώα, αλλά αναπαρίσταται ως μια συνέχεια χαρακτηριστικών όπως φαίνεται στο σύνολο δεδομένων στο σχήμα. Οι δοκιμές που χρησιμοποιούνται σε συνεχή δεδομένα είναι της μορφής "*είναι το χαρακτηριστικό i μεγαλύτερο από την τιμή a* ". Για τη δημιουργία ενός δέντρο, ο αλγόριθμος αναζητά όλες τις πιθανές δοκιμές και βρίσκει αυτή που είναι πιο ενημερωτική για τη μεταβλητή στόχου [50].

2.6.2 Έλεγχος της πολυπλοκότητας των δέντρων απόφασης

Συνήθως, η κατασκευή ενός δέντρου όπως περιγράφεται παραπάνω και η συνέχισή του έως ότου όλα τα φύλλα είναι καθαρά οδηγεί σε μοντέλα που είναι πολύ περίπλοκα και υπερβάλλουν τα δεδομένα εκπαίδευσης, έχουμε δηλαδή το φαινόμενο *overfit*. Η παρουσία καθαρών φύλλων σημαίνει ότι ένα δέντρο είναι 100% ακριβές στο σετ προπόνησης. Υπάρχουν δύο κύριες στρατηγικές για την αποφυγή της υπερβολικής τοποθέτησης ή αλλιώς *overfit* : 1) η διακοπή της δημιουργίας του δέντρο νωρίς, που ονομάζεται επίσης προ-κλάδεμα, ή 2) η κατασκευή του δέντρου, αλλά στη συνέχεια να γίνει αφαίρεση των κόμβων που περιέχουν λίγες πληροφορίες. Πιθανά κριτήρια για το προ-κλάδεμα περιλαμβάνουν τον περιορισμό του μέγιστου βάθους του δέντρου, περιορίζοντας έτσι τον μέγιστο αριθμό φύλλων ή απαιτώντας έναν ελάχιστο αριθμό πόντων σε έναν κόμβο για να συνεχίσει να το χωρίζει [51].

2.6.3 Πλεονεκτήματα, μειονεκτήματα και παράμετροι

Όπως αναφέρθηκε και παραπάνω, οι παράμετροι που ελέγχουν την πολυπλοκότητα του μοντέλου στα δέντρα αποφάσεων είναι οι παράμετροι προ-κλάδευσης που σταματούν την κατασκευή του δέντρου πριν ολοκληρωθεί. Συνήθως επιλέγοντας μία από τις στρατηγικές προ-κλαδέματος, κάθε ρύθμιση *min_depth*, *max_leaf_nodes* ή *min_samples_leaf* είναι να αποφευχθεί η υπερβολική τοποθέτηση. Τα δέντρα αποφάσεων έχουν δύο πλεονεκτήματα έναντι πολλών από τους αλγορίθμους που συζητήσαμε μέχρι τώρα: Το μοντέλο που προκύπτει μπορεί εύκολα να οπτικοποιηθεί και να γίνει κατανοητό από μη ειδικούς (τουλάχιστον για τα μικρότερα δέντρα), και οι αλγόριθμοι είναι απολύτως αναλλοίωτοι στην κλιμάκωση των δεδομένων: καθώς κάθε χαρακτηριστικό επεξεργάζεται ξεχωριστά και οι πιθανές διαχωρίσεις των δεδομένων δεν εξαρτώνται κατά την κλιμάκωση, δεν υπάρχει προ επεξεργασία όπως κανονικοποίηση ή τυποποίηση των χαρακτηριστικών. Συγκεκριμένα, τα δέντρα αποφάσεων λειτουργούν καλά όταν έχουμε χαρακτηριστικά που είναι εντελώς σε διαφορετικές κλίμακες ή είναι ένας συνδυασμός δυαδικών και συνεχών χαρακτηριστικών. Το κύριο μειονέκτημα των δέντρων απόφασης είναι ότι ακόμη και με τη χρήση της προ-κλάδευσης, η τα δέντρα τείνουν να υπερφορτώνουν και παρέχουν χαμηλή απόδοση γενίκευσης. Επομένως, στις περισσότερες εφαρμογές, οι μέθοδοι που έχουν αναφερθεί χρησιμοποιούνται συνήθως στη θέση ενός δέντρου αποφάσεων [52].

ΚΕΦΑΛΑΙΟ 3

ΑΝΙΧΝΕΥΣΗ ΚΑΚΟΒΟΥΛΟΥ ΛΟΓΙΣΜΙΚΟΥ

Η υψηλή διάδοση κακόβουλου λογισμικού (*malware*) και *ransomware* (τύπος κακόβουλου λογισμικού σχεδιασμένο να μην επιτρέπει την είσοδο σε ένα υπολογιστικό σύστημα μέχρι να πληρωθεί ένα συγκεκριμένο ποσό χρημάτων), σε συνδυασμό με τις γρήγορες και πολυμορφικές μεταλλάξεις των διάφορων παραλλαγών (πολυμορφικού και μεταμορφικού κακόβουλου λογισμικού) των ίδιων των απειλών, έχει κάνει παραδοσιακές λύσεις ανίχνευσης βάσει υπογραφών και των κατακερματισμένων αρχείων εικόνας ξεπερασμένα, στα οποία βασίζεται το πιο κοινό λογισμικό προστασίας από ιούς. Είναι επομένως όλο και πιο απαραίτητο να καταφύγουμε σε λύσεις μηχανικής μάθησης (*ML*) που επιτρέπει την ταχεία διαλογή απειλών, εστιάζοντας την προσοχή στη μη σπατάλη σπάνιων πόρων όπως οι δεξιότητες και οι προσπάθειες ενός αναλυτή κακόβουλου λογισμικού[53].

3.1 Ανάλυση κακόβουλου λογισμικού

Μια από τις πιο ενδιαφέρουσες πτυχές για τους ειδικούς που ασχολούνται με την ανάλυση κακόβουλου λογισμικού είναι η διάκρισή του, για παράδειγμα, δυαδικά αρχεία που είναι δυνητικά επικίνδυνα για την ακεραιότητα των μονάδων των συστημάτων και των δεδομένων που περιέχουν. Αναφερόμαστε γενικά στα δυαδικά αρχεία και όχι σε εκτελέσιμα αρχεία (δηλαδή, αρχεία με τύπους όπως .exe ή .dll), δεδομένου ότι το κακόβουλο λογισμικό μπορεί να κρύβεται ακόμη και σε φαινομενικά αβλαβείς αρχεία, όπως αρχεία εικόνας (αρχεία τύπου όπως .jpg και .png)

Με τον ίδιο τρόπο, ακόμη και έγγραφα κειμένου (όπως docx ή .pdf) μπορεί να αποδειχτούν υγιείς αρχεία ή μέσα μολυσμένου λογισμικού, παρά τη μη εκτελέσιμη μορφή αρχείου τους. Επιπλέον, το πρώτο στάδιο της εξάπλωσης ενός κακόβουλου λογισμικού (και στις δύο περιπτώσεις προσωπικού υπολογιστή και σε δίκτυο LAN εταιρείας) συμβαίνει συχνά με κίνδυνο της ακεραιότητας των αρχείων που βρίσκονται μέσα τα μηχανήματα που δέχονται επίθεση.

Ως εκ τούτου, είναι θεμελιώδους σημασίας να μπορούμε να αναγνωρίζουμε αποτελεσματικά την παρουσία του κακόβουλου λογισμικού, προκειμένου να αποφευχθεί, ή τουλάχιστον να περιοριστεί, η διάδοσή του εντός ενός οργανισμού.

Παρακάτω ακολουθούν οι στρατηγικές ανάλυσης (και τα σχετικά εργαλεία) που συνήθως χρησιμοποιούνται για την διεξαγωγή μιας προκαταρκτικής έρευνας για αρχεία και λογισμικό που διαδίδονται στην άγρια φύση (μέσω παραπλανητικών συνδέσμων, ανεπιθύμητα

μηνύματα ηλεκτρονικού ταχυδρομείου, ηλεκτρονικό ψάρεμα (phishing) και άλλα), προκειμένου να εντοπιστούν αυτά που είναι δυνητικά επικίνδυνα. Για να επιτύχουμε αυτόν τον στόχο, θα πρέπει να εξετάσουμε τις παραδοσιακές μεθόδους στατικής και δυναμικής ανάλυσης *malware* πιο στενά [54].

3.2 Τεχνητή νοημοσύνη για τον εντοπισμό κακόβουλου λογισμικού

Με την σχεδόν εκθετική αύξηση του αριθμού των απειλών που σχετίζονται με την καθημερινή εξάπλωση νέων κακόβουλων λογισμικών, είναι σχεδόν αδύνατο να σκεφτεί κανείς ότι η αντιμετώπιση αυτών των απειλών μπορεί να γίνει αποτελεσματικά χρησιμοποιώντας μόνο την ανάλυση από το ανθρώπινο δυναμικό. Επομένως, είναι απαραίτητη η εισαγωγή αλγορίθμων που επιτρέπουν τουλάχιστον την αυτοματοποιήση της προπαρασκευαστικής φάσης ανάλυσης κακόβουλου λογισμικού, γνωστή ως triage, που απορρέει από την ίδια πρακτική που υιοθετήθηκε από τους γιατρούς κατά τον Πρώτο Παγκόσμιο Πόλεμο, και συνίσταται στην επιλογή για θεραπεία των τραυματιών που είναι πιο πιθανό να επιβιώσουν. Δηλαδή, διεξαγωγή προκαταρκτικού ελέγχου των κακόβουλων λογισμικών που αναλύονται από τον αυτόματο αναλυτή κακόβουλου λογισμικού που τους επιτρέπει να ανταποκρίνονται εγκαίρως και με αποτελεσματικό τρόπο για πραγματικές απειλές στον κυβερνοχώρο. Αυτοί οι αλγόριθμοι έχουν τη μορφή υιοθέτησης εργαλείων τεχνητής νοημοσύνης, δεδομένου του δυναμισμού που εξ’ ορισμού χαρακτηρίζει την κυβερνοασφάλεια. Στην πραγματικότητα, είναι απαραίτητο τα συστήματα να μπορούν να ανταποκριθούν αποτελεσματικά, ενώ προσαρμόζονται στις αλλαγές που σχετίζονται με την εξάπλωση πρωτοφανών απειλών. Αυτό δεν σημαίνει μόνο ότι ο αναλυτής χειρίζεται τα εργαλεία και τις μεθόδους ανάλυσης κακόβουλου λογισμικού (η οποία είναι προφανής), αλλά ότι μπορούν επίσης να ερμηνεύσουν τη συμπεριφορά των αλγορίθμων, έχοντας επίγνωση των επιλογών που έχει υιοθετήσει η μηχανή. Ο αναλυτής κακόβουλου λογισμικού, επομένως, καλείται να κατανοήσει τη λογική που ακολουθείται από το μοντέλο μηχανικής μάθησης, παρεμβαίνοντας (άμεσα ή έμμεσα) στην τελειοποίηση της σχετικής διαδικασίας εκμάθησης, με βάση τα αποτελέσματα που προέκυψαν από την αυτοματοποιημένη ανάλυση [55].

3.3 Ονοματολογία κακόβουλου λογισμικού

Υπάρχουν πολλοί τύποι κακόβουλου λογισμικού και καθημερινά δημιουργούνται νέες μορφές απειλής που δημιουργικά επαναχρησιμοποιούν τις προηγούμενες μορφές επίθεσης ή υιοθετούν ριζικά νέες συμβιβαστικές στρατηγικές που εκμεταλλεύονται συγκεκριμένα χαρακτηριστικά του οργανισμού-στόχου (στην περίπτωση των *Advanced Persistent Threats (APTs),* αυτές είναι προσαρμοσμένες μορφές επίθεσης που προσαρμόζονται τέλεια στο θύμα τους). Αυτό περιορίζεται μόνο στη φαντασία του εισβολέα. Ωστόσο, είναι δυνατό να

καταρτιστεί μια ταξινόμηση των πιο κοινών τύπων κακόβουλου λογισμικού, προκειμένου να κατανοήσουμε ποια είναι τα πιο αποτελεσματικά μέτρα πρόληψης ώστε να είμαστε σε θέση να αντιμετωπίσουμε αποτελεσματικά κάθε είδους κακόβουλου λογισμικού:

- **Trojans**: Εκτελέσιμα αρχεία που εμφανίζονται ως νόμιμα και ακίνδυνα, αλλά όταν ανοίγονται, εκτελούν κακόβουλες οδηγίες στο παρασκήνιο.
- **Botnets**: κακόβουλο λογισμικό που έχει ως στόχο να θέσει σε κίνδυνο όσο το δυνατόν περισσότερους κεντρικούς υπολογιστές του δικτύου (hosts), προκειμένου να τεθεί η υπολογιστική τους ικανότητα στην υπηρεσία του επιτιθέμενου.
- **Downloads**: κακόβουλο λογισμικό που κατεβάζει κακόβουλες βιβλιοθήκες ή τμήματα κώδικα από το διαδίκτυο και τα εκτελεί σε κεντρικούς υπολογιστές θυμάτων.
- **Rootkits**: κακόβουλο λογισμικό που θέτει σε κίνδυνο τους κεντρικούς υπολογιστές σε επίπεδο λειτουργικού συστήματος, συχνά έρχονται με τη μορφή οδηγών προγραμμάτων συσκευών, καθιστώντας τα διάφορα αντίμετρα (όπως τα antivirus που είναι εγκατεστημένα στα τελικά σημεία) μη αποτελεσματικά.
- **Ransomwares**: κακόβουλο λογισμικό που προχωρά στην κρυπτογράφηση αρχείων που είναι αποθηκευμένα στον κεντρικό υπολογιστή, ζητώντας λύτρα από το θύμα (συχνά πληρώνεται σε Bitcoin) για να λάβει το κλειδί αποκρυπτογράφησης που χρησιμοποιείται για την ανάκτηση των αρχικών αρχείων.
- **APTs**: Τα APT είναι μορφές προσαρμοσμένων επιθέσεων που εκμεταλλεύονται συγκεκριμένες αδυναμίες στον υπολογιστή-θύμα.
- **Zero days (0 days)**: κακόβουλο λογισμικό που εκμεταλλεύεται αδυναμίες που δεν έχουν ακόμη αποκαλυφθεί στο κοινότητα ερευνητών και αναλυτών, των οποίων τα χαρακτηριστικά και οι επιπτώσεις δεν είναι ακόμη γνωστοί με αποτέλεσμα να μην εντοπίζονται από antivirus λογισμικό.

Προφανώς, αυτοί οι διαφορετικοί τύποι απειλών μπορούν να ενισχυθούν από το γεγονός ότι μπορούν να συνδυαστούν μαζί στο ίδιο κακόβουλο αρχείο για παράδειγμα, ένας φαινομενικά ακίνδυνος *Trojan* γίνεται πραγματική απειλή, καθώς συμπεριφέρεται σαν πρόγραμμα λήψης όταν εκτελείται, συνδέεται στο δίκτυο και κατεβάζει κακόβουλο λογισμικό, όπως *rootkits*, το οποίο θέτει σε κίνδυνο το τοπικό δίκτυο και το μετατρέπει σε *botnet* (ένα δίκτυο προσωπικών υπολογιστών το οποίο είναι μολυσμένο με κακόβουλο λογισμικό και ελέγχεται από τρίτους χωρίς την επίγνωση του ιδιοκτήτη) [56].

3.4 Εργαλεία ανάλυσης κακόβουλου λογισμικού

Τα περισσότερα από τα εργαλεία που χρησιμοποιούνται συνήθως για τη διεξαγωγή ανάλυσης κακόβουλου λογισμικού κατηγοριοποιούνται όπως παρακάτω:

- Disassembles (όπως Disasm και IDA)
- Debuggers (όπως OllyDbg, WinDbg και IDA)
- System monitors (όπως Process Monitor και Process Explorer)
- Network monitors (όπως TCP View, Wireshark και tcpdump)
- Unpacking tools και Packer Identifiers (όπως PEiD)
- Binary and code analysis tools (όπως PEView, PE Explorer, LordPE και ImpREC) [56]

3.5 Στρατηγικές εντοπισμού κακόβουλου λογισμικού

Προφανώς, κάθε είδος απειλής απαιτεί μια συγκεκριμένη στρατηγική ανίχνευσης. Σε αυτήν την ενότητα, θα παρουσιαστούν οι μέθοδοι ανάλυσης που χρησιμοποιούνται, παραδοσιακά στην ανίχνευση κακόβουλων προγραμμάτων, χειροκίνητα από αναλυτές κακόβουλου λογισμικού. Παρέχουν μια πιο λεπτομερή κατανόηση των φάσεων της ανάλυσης που μπορεί να βελτιωθεί και να γίνει πιο αποτελεσματική με την εισαγωγή αλγορίθμων τεχνητής νοημοσύνης, απελευθερώνοντας έτσι τον ανθρώπινο αναλυτή από τις πιο επαναλαμβανόμενες και κουραστικές εργασίες, επιτρέποντάς του να επικεντρωθεί στις πιο περίεργες ή ασυνήθιστες πτυχές της ανάλυσης. Αξίζει να σημειωθεί ότι η ανάπτυξη λογισμικού κακόβουλου λογισμικού είναι το αποτέλεσμα μιας δημιουργικής δραστηριότητας που πραγματοποιείται από τον εισβολέα και με αποτέλεσμα να μην μπορεί εύκολα να κατηγοριοποιηθεί σε προκαθορισμένα πρότυπα ή προκαθορισμένους τρόπους επίθεσης. Με τον ίδιο τρόπο, ο αναλυτής κακόβουλου λογισμικού πρέπει να χρησιμοποιεί σε όλους τους ευφάνταστους πόρους τους, καθώς και στην ανάπτυξη μη συμβατικών διαδικασιών, για να είναι σε θέση να βρίσκεται μπροστά από τον εισβολέα σε ένα είδος παιχνιδιού γάτας και ποντικιού. Επομένως, η ανάλυση κακόβουλου λογισμικού πρέπει να θεωρείται περισσότερο ως τέχνη παρά επιστήμη, καθώς απαιτεί την ικανότητα του αναλυτή να φαντάζεται πάντα νέους τρόπους ανίχνευσης για τον προσδιορισμό μελλοντικών απειλών εγκαίρως. Κατά συνέπεια, ο αναλυτής κακόβουλου λογισμικού καλείται να ενημερώνει συνεχώς όχι μόνο τις τεχνικές τους δεξιότητες, αλλά και τις μεθόδους που εφαρμόζει για την διερεύνηση των απειλών.

Είναι γεγονός ότι δύναται να αρχίσει μια δραστηριότητα ανίχνευσης κακόβουλου λογισμικού καταφεύγοντας μόνο σε κοινές πρακτικές ανάλυσης, ειδικά για τον εντοπισμό των γνωστών απειλών. Μεταξύ των πιο κοινών διαδικασιών εντοπισμού κακόβουλου λογισμικού, μπορούν να συμπεριληφθούν οι ακόλουθες :

- *Hashes file calculation*: Για τον εντοπισμό γνωστών απειλών που είναι ήδη γνωστοί.
- *System monitoring*: Για τον εντοπισμό μη κανονικής συμπεριφοράς τόσο στο υλικό όσο και στο λειτουργικό σύστημα (όπως μια ασυνήθιστη αύξηση των κύκλων της κεντρικής μονάδας επεξεργασίας (CPU), αυξημένη δραστηριότητα εγγραφής των σκληρών δίσκων, αλλαγές στα κλειδιά μητρώου και δημιουργία νέων και ανεπιθύμητων διαδικασιών στο σύστημα).
- *Network monitoring*: Για τον εντοπισμό μη κανονικών συνδέσεων που δημιουργούνται από κεντρικούς υπολογιστές σε απομακρυσμένους και άγνωστους προορισμούς.

Αυτές οι δραστηριότητες ανίχνευσης μπορούν να αυτοματοποιηθούν εύκολα χρησιμοποιώντας συγκεκριμένους αλγόριθμους, όπως θα γίνει παρακάτω μετά την εξέταση μεθοδολογιών ανάλυσης κακόβουλου λογισμικού [56].

3.6 Στατική ανάλυση κακόβουλου λογισμικού

Το πρώτο βήμα στην ανάλυση κακόβουλου λογισμικού ξεκινά με την αξιολόγηση της παρουσίας ύποπτων αντικειμένων σε δυαδικά αρχεία, χωρίς να εκτελείται ο κώδικας.

Η ανάλυση στατικού κακόβουλου λογισμικού αποτελείται από τις ακόλουθες διεργασίες:

- Προσδιορισμός των αντικειμένων που χρήζουν ανάλυση.
- Κατανόηση της ροής των εκτελέσιμων οδηγιών.
- Προσδιορισμός γνωστών μοτίβων και συσχέτισή τους με πιθανά κακόβουλά λογισμικά.

Επιπλέον, χρησιμοποιούνται εργαλεία και διαδικασίες ανάλυσης για την εκτέλεση των παρακάτω απαραίτητων λειτουργιών:

- Προσδιορισμός κλήσεων στα APIs του συστήματος.
- Αποκωδικοποίηση και χειρισμός δεδομένων συμβολοσειράς για τη λήψη ευαίσθητων πληροφοριών (για παράδειγμα, ονόματα τομέων και διευθύνσεις IP).
- Εντοπισμός παρουσίας με τη λήψη κώδικα από άλλα κακόβουλα λογισμικά [54].

3.6.1 Μεθοδολογία στατικής ανάλυσης

Η μεθοδολογία που χρησιμοποιείται από την στατική ανάλυση κακόβουλου λογισμικού συνίσταται στην εξέταση των οδηγιών των μηχανών (οδηγίες συναρμολόγησης) που υπάρχουν στην αποσυναρμολογημένη δυαδική εικόνα του κακόβουλου λογισμικού (αποσυναρμολόγηση κακόβουλου λογισμικού), προκειμένου να εντοπιστούν οι επιβλαβείς

δυνατότητές του και να γίνει αξιολόγηση των εξωτερικών χαρακτηριστικών του δυαδικού κώδικα, προτού γίνει η εκτέλεσή του [56].

3.6.2 Δυσκολίες στατικής ανάλυσης κακόβουλου λογισμικού

Μεταξύ των πιο ύπουλων πτυχών της στατικής ανάλυσης κακόβουλου λογισμικού είναι οι δυσκολίες προσδιορισμού της ορθότητας της αποσυναρμολόγησης του κακόβουλου λογισμικού. Δεδομένης της όλο και μεγαλύτερης παρουσίας τεχνικών κατά της ανάλυσης, δεν είναι πάντα δυνατό να το υποθέσουμε ότι η αποσυναρμολογημένη δυαδική εικόνα που παράγεται από τον αποσυναρμολογητή είναι αξιόπιστη. Επομένως, ο αναλυτής πρέπει να πραγματοποιήσει μια προκαταρκτική ανάλυση, προκειμένου να εντοπίσει, για παράδειγμα, την παρουσία συσκευαστών (packers) που κρυπτογραφούν τμήματα του εκτελέσιμου κώδικα.

Τέτοιες προκαταρκτικές διαδικασίες ανάλυσης παραλείπονται συχνά από τους αναλυτές επειδή είναι αρκετά χρονοβόρα διαδικασία, παρόλα αυτά, είναι απαραίτητες για την οριοθέτηση των στόχων που πρέπει να επιτευχθούν. Επιπλέον, εάν η παρουσία τμημάτων εκτελέσιμου κώδικα δεν ανιχνεύεται σωστά (ίσως επειδή κρύβονται μέσα σε δεδομένα που θεωρούνται ακίνδυνα, όπως εικόνες), αυτή η ανεπάρκεια, μπορεί να υπονομεύσει τις επόμενες φάσεις της δυναμικής ανάλυσης, καθιστώντας αδύνατο τον προσδιορισμό του ακριβούς τύπου κακόβουλου λογισμικού που ερευνάται [54].

3.6.3 Πως πραγματοποιείται η στατική ανάλυση

Μόλις γίνει επαλήθευση ότι το αποσυναρμολογημένο κακόβουλο λογισμικό είναι αξιόπιστο, έπειτα συνεχίζεται η διαδικασία με διαφορετικούς τρόπους: κάθε αναλυτής, στην πραγματικότητα, ακολουθεί τη δική του προτιμώμενη στρατηγική, η οποία βασίζεται στην εμπειρία αλλά και τους στόχους που σκοπεύουν να επιδιωχθούν.

Οι υιοθετούμενες στρατηγικές είναι οι εξής:

- Ανάλυση των δυαδικών οδηγιών με συστηματικό τρόπο, χωρίς την εκτέλεσή τους. Είναι μια αποτελεσματική τεχνική για περιορισμένα τμήματα κώδικα που γίνονται περίπλοκα σε περιπτώσεις μεγάλου κακόβουλου λογισμικού, καθώς ο αναλυτής πρέπει να παρακολουθεί την κατάσταση των δεδομένων για κάθε οδηγία που αναλύεται.

- Σάρωση των οδηγιών με σκοπό την αναζήτηση ακολουθιών που θεωρούνται ενδιαφέρουσες, ρύθμιση σημείων διακοπής και εκτέλεση του προγράμματος έως το σημείο διακοπής, και έπειτα εξέταση την κατάσταση του προγράμματος ως αυτό το σημείο. Αυτή η προσέγγιση χρησιμοποιείται συχνά για τον προσδιορισμό της παρουσίας κλήσεων του συστήματος που θεωρούνται επικίνδυνες, βασισμένες στην ακολουθία με την οποία πραγματοποιούνται αυτές οι κλήσεις.

- Με τον ίδιο τρόπο, είναι δυνατό να εντοπιστεί η απουσία ορισμένων κλήσεων *API*. Ένας κώδικας που δεν παρουσιάζει προσκλήσεις στις κλήσεις συστήματος (για παράδειγμα, κλήσεις σχετικές με το δίκτυο), οι οποίες είναι απαραίτητες για την έκδοση συνδέσεων δικτύου, δεν μπορεί προφανώς αντιπροσωπεύει μια πίσω πόρτα (*backdoor*) αλλά θα μπορούσε να λειτουργήσει, για παράδειγμα, ως *keylogger*, επειδή καλεί την ακολουθία των *API* του συστήματος για να εντοπίσει τα πλήκτρα που πατήθηκαν στο πληκτρολόγιο για την εγγραφή στο δίσκο.

- Αναζήτηση ευαίσθητων πληροφοριών (όπως ονόματα τομέα και διευθύνσεις *IP*) με τη μορφή συμβολοσειράς μέσα στην αποσυναρμολογημένη εικόνα. Επίσης, σε αυτήν την περίπτωση, είναι δυνατό να ρυθμιστούν σημεία σφαλμάτων και διακοπής σε αντιστοιχία με τις κλήσεις δικτύου για την ανίχνευση τυχόν ονομάτων τομέα ή απομακρυσμένες διευθύνσεις *IP* που έρχονται σε επαφή με το κακόβουλο λογισμικό όταν υπάρχει σύνδεση στο Διαδίκτυο [56].

3.6.4 Απαιτήσεις υλικού για στατική ανάλυση

Σε αντίθεση με τη δυναμική ανάλυση, η στατική ανάλυση συνήθως απαιτεί λιγότερους συγκεκριμένους από την άποψη υλικού, αφού, ο αναλυτής δεν εκτελεί τον κακόβουλο κώδικα. Αντίθετα, στην περίπτωση δυναμικής ανάλυσης κακόβουλου λογισμικού, σημαντικές απαιτήσεις υλικού μπορεί να απαιτούνται, και σε ορισμένες περιπτώσεις δεν αρκεί η χρήση εικονικών μηχανών. Αυτό οφείλεται στην παρουσία αντιμέτρων (τεχνάσματα κατά της ανάλυσης) που εφαρμόζονται από το κακόβουλο λογισμικό, το οποίο αποτρέπει την εκτέλεση του κώδικα εάν εντοπιστεί η παρουσία μιας εικονικής μηχανής [54].

3.7 Δυναμική ανάλυση κακόβουλου λογισμικού

Όπως προαναφέρθηκε, τα χαρακτηριστικά της στατικής ανάλυσης κακόβουλου λογισμικού είναι τα ακόλουθα:

- Βεβαιότητα ότι ένα δεδομένο δυαδικό αρχείο είναι πραγματικά κακόβουλο.

- Αναγνώριση όσο το δυνατόν περισσότερων πληροφοριών σχετικά με το δυαδικό αρχείο, χωρίς την εκτέλεσή του και διεξαγωγή της ανάλυσης με βάση τα χαρακτηριστικά του, που μπορεί να αντιστοιχούν σε προηγούμενα αρχεία κακόβουλου λογισμικού, όπως χαρακτηριστικά από τη μορφή του αρχείου ή από τους πόρους που είναι αποθηκευμένοι σε αυτό.

- Καταγραφή του ύποπτου δυαδικού αρχείου με τον υπολογισμό του κατακερματισμού του, το οποίο αποτελεί την υπογραφή του (αυτή η υπογραφή μπορεί επίσης να κοινοποιηθεί στην κοινότητα των αναλυτών κακόβουλου

λογισμικού, προκειμένου να ενημερώνεται η βάση δεδομένων για τις απειλές από κακόβουλα προγράμματα).

- Χωρίς καμία αμφιβολία, η στατική ανάλυση κακόβουλου λογισμικού, αν και είναι γρήγορη στη διεξαγωγή, παρουσιάζει μια σειρά μεθοδολογικών περιορισμών, ειδικά όταν πρόκειται για την ανάλυση εξελιγμένου τύπου κακόβουλου λογισμικού (όπως το *APT* και το πολυμορφικό κακόβουλο λογισμικό). Ένα από τα διορθωτικά μέτρα σε αυτά τα μεθοδολογικά όρια αποτελεί ο συνδυασμός της στατικής με τη δυναμική ανάλυση κακόβουλου λογισμικού, σε μια προσπάθεια να κατανόησης της φύσης και του τύπο του κακόβουλου λογισμικού.

Το χαρακτηριστικό που διακρίνει τη δυναμική ανάλυση κακόβουλου λογισμικού είναι το γεγονός ότι, σε αντίθεση με τη στατική, εκτελείται το δυαδικό αρχείο (συχνά σε μεμονωμένο και προστατευμένο περιβάλλον, γνωστό ως εργαστήριο ανάλυσης κακόβουλου λογισμικού, το οποίο κάνει χρήση των *sandbox* και διάφορων εικονικών μηχανημάτων (*virtual machines*) για την αποτροπή της ευρείας εξάπλωσης κακόβουλου λογισμικού στο εταιρικό δίκτυο).

Επομένως, αυτή η στρατηγική συνεπάγεται της ανάλυσης της δυναμικής συμπεριφοράς, δηλαδή της επαλήθευσης, για παράδειγμα, ότι το κακόβουλο εκτελέσιμο αρχείο δεν κάνει λήψη κακόβουλων βιβλιοθηκών ή τμημάτων του κώδικα από το διαδίκτυο ή προχωρά στην τροποποίηση των δικών του εκτελέσιμων οδηγιών σε κάθε εκτέλεση του αρχείου, κάνοντας έτσι τις διαδικασίες ανίχνευσης βάσει υπογραφής (που χρησιμοποιούνται από το *antivirus*) αναποτελεσματικές [54].

3.7.1 Τεχνάσματα κατά της ανάλυσης

Τα αντίμετρα που συνήθως υιοθετούνται από προγραμματιστές κακόβουλου λογισμικού, αποτρέπουν ή κάνουν δυσκολότερη την ανάλυση του κακόβουλου λογισμικού, βασίζονται στην κρυπτογράφηση των φορτίων (*payloads*), στη χρήση συσκευαστών (*packers*), στα προγράμματα λήψεων και άλλων.

Αυτά τα τεχνάσματα είναι συνήθως ανιχνεύσιμα με τη δυναμική ανάλυση κακόβουλου λογισμικού, ωστόσο, ακόμη η δυναμική ανάλυση κακόβουλου λογισμικού εμπεριέχει περιορισμούς που σχετίζονται με τη χρήση εικονικών μηχανημάτων, για παράδειγμα, των οποίων η παρουσία μπορεί να εντοπιστεί εύκολα από το κακόβουλο λογισμικό, χρησιμοποιώντας μερικά από τα παρακάτω τεχνάσματα εκτέλεσης:

- Εκτέλεση οδηγιών που αναμένουν μια προεπιλεγμένη συμπεριφορά: το κακόβουλο λογισμικό μπορεί να υπολογίσει το χρόνο που έχει παρέλθει κατά την εκτέλεση ορισμένων λειτουργιών και εάν αυτές εκτελέστηκαν πιο αργά από το αναμενόμενο, μπορεί να βγάλει το συμπέρασμα ότι η εκτέλεση πραγματοποιείται σε μια εικονική μηχανή.

- Ανίχνευση της εικονικής μηχανής με βάση το υλικό: μέσω της εκτέλεσης συγκεκριμένων οδηγιών σε επίπεδο υλικού (για παράδειγμα, οι οδηγίες που έχουν πρόσβαση σε περιοχές προστατευμένες από την κεντρική μονάδα επεξεργασίας, όπως *sldt*, *sgdt* και *sidt*).
- Πρόσβαση σε ορισμένα κλειδιά μητρώου.

Όταν το κακόβουλο λογισμικό εντοπίζει την παρουσία μιας εικονικής μηχανής, σταματά να λειτουργεί με τον αναμενόμενο τρόπο, αποφεύγοντας έτσι τις προσπάθειες εντοπισμού του από τους αναλυτές [57].

3.8 Τα αρχεία μορφής PE ως πιθανοί φορείς μόλυνσης

Τα εκτελέσιμα αρχεία *PE* έχουν πολλά τμήματα που περιλαμβάνονται στην εικόνα του δυαδικού αρχείου και αυτό το χαρακτηριστικό τους μπορεί να αξιοποιηθεί για την απόκρυψη του κακόβουλου λογισμικού. Στην πραγματικότητα, κάθε ένα από αυτά τα τμήματα των *PE* αρχείων μπορεί να θεωρηθεί ως φάκελος, που φιλοξενεί διάφορα δυαδικά αντικείμενα (από αρχεία γραφικών έως κρυπτογραφημένες βιβλιοθήκες), τα οποία εκτελούνται ή και αποκρυπτογραφούνται στο χρόνο εκτέλεσης, μολύνοντας ενδεχομένως άλλα εκτελέσιμα αρχεία στο ίδιο μηχάνημα ή σε απομακρυσμένα μηχανήματα που βρίσκονται στο ίδιο δίκτυο. Για παράδειγμα, ένα τμήμα *PE* μπορεί να περιέχει ένα αρχείο .sys (κακόβουλο πρόγραμμα οδήγησης) που στοχεύει στην παραβίαση του πυρήνα, μαζί με ένα άλλο αρχείο εκκίνησης που περιέχει παραμέτρους διαμόρφωσης ή απομακρυσμένους συνδέσμους με τους οποίους μπορεί να συνδεθεί το δυαδικό, για να τη λήψη μέσω διαδικτύου άλλων αντικείμενων ενεργοποίησης [58].

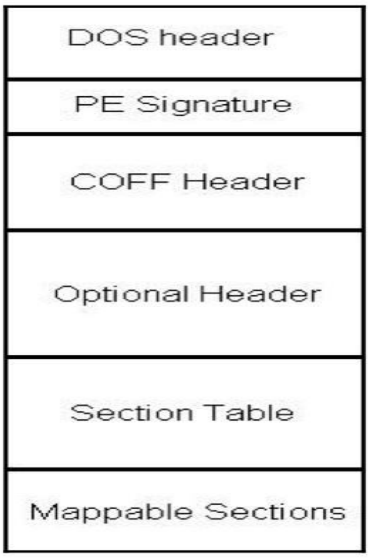
3.8.1 Επισκόπηση της μορφής αρχείου PE

Οι προδιαγραφές των αρχείων *PE* προέρχονται από το *Unix Common Object File Format* (*COFF*) και είναι βασικά μια δομή δεδομένων που καλύπτει τις απαραίτητες πληροφορίες για το λειτουργικό σύστημα των *Windows* με σκοπό τη διαχείριση της εκτελέσιμης εικόνας, δηλαδή όταν οι δομές της χαρτογραφούνται στη *runtime memory* πριν εκτελεστεί το αρχείο από το λειτουργικό σύστημα. Με απλά λόγια, ένα αρχείο *PE* αποτελείται από την κεφαλίδα του αρχείου *PE* και έναν πίνακα τμημάτων, ακολουθούμενο από τα δεδομένα των τμημάτων. Η κεφαλίδα αρχείου *PE* ενσωματώνεται στη δομή κεφαλίδας *NT* των *Windows* (ορίζεται στο αρχείο κεφαλίδας *winnt.h*, μαζί με άλλες δομές *C*) και αποτελείται από τα ακόλουθα:

- Κεφαλίδα MS DOS (MS DOS header)
- Την υπογραφή PE (The PE signature)
- Την κεφαλίδα του αρχείου εικόνας (The image file header)

- Μια προεραιτική κεφαλίδα (An optional header)

Οι κεφαλίδες των αρχείων ακολουθούνται από τις παρακάτω κεφαλίδες τμημάτων:



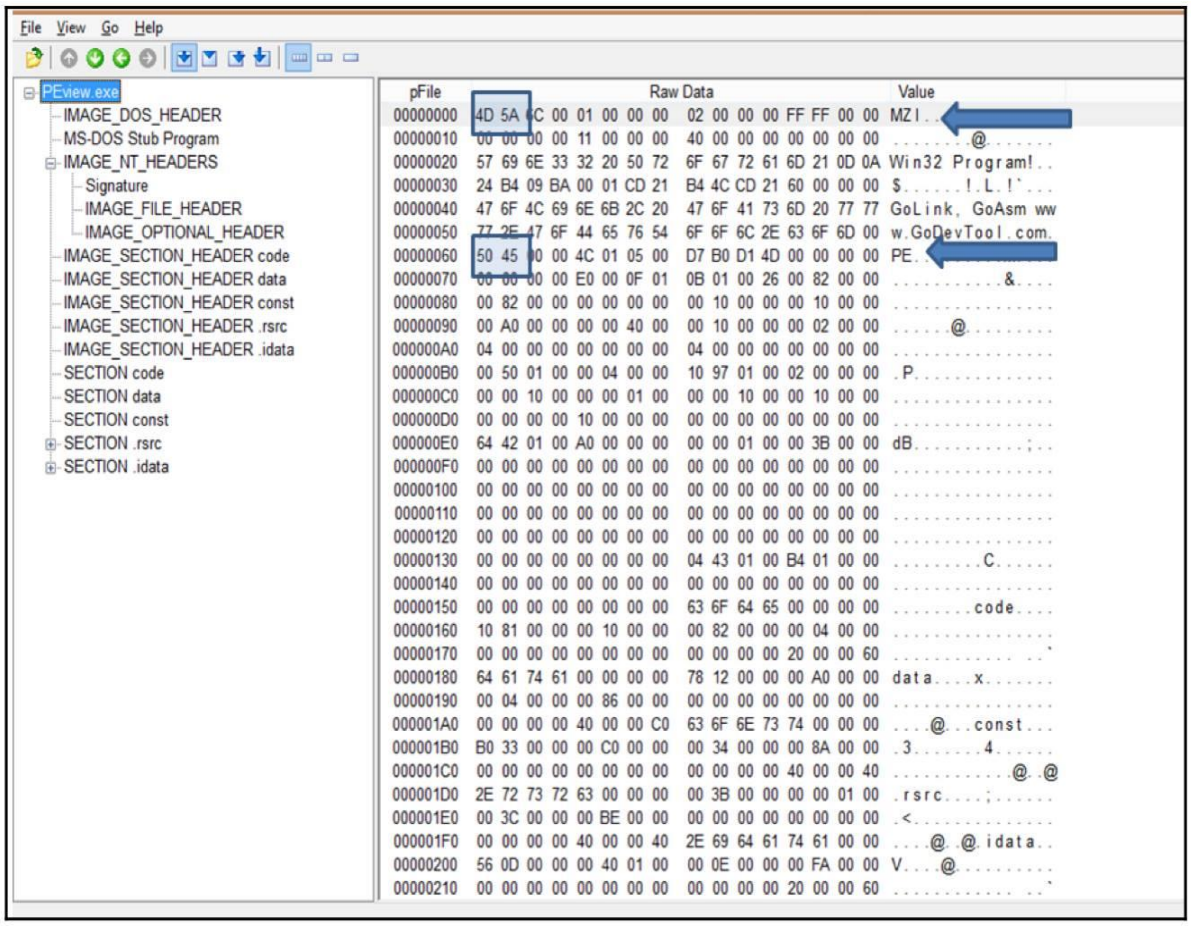
Πηγή: Soma Halder, S. O. (2018). Machine Learning for Cybersecurity . Birmingham: Packt.

Εικόνα 5 Τμήματα της κεφαλίδας των εκτελέσιμων αρχείων.

Η κεφαλίδα του τμήματος παρέχει πληροφορίες συμπεριλαμβανομένης της τοποθεσίας, του μήκους και τα χαρακτηριστικά του αρχείου. Ένα τμήμα είναι η βασική μονάδα του κώδικα ή των δεδομένων σε ένα αρχείο *PE*. Διαφορετικές λειτουργικές περιοχές, όπως περιοχές του κώδικα και των δεδομένων, διαχωρίζονται λογικά σε τμήματα. Επιπλέον, ένα αρχείο εικόνας μπορεί να περιέχει έναν αριθμό τμημάτων, όπως *.tls* και *.reloc*, τα οποία έχουν ειδικούς σκοπούς. Η κεφαλίδα του τμήματος παρέχει πληροφορίες που αφορούν το αντίστοιχο τμήμα. Τα πιο κοινά τμήματα στα εκτελέσιμα αρχεία είναι κείμενα, δεδομένα, *RSRC*, *RData* και *RELOC*. Τα περισσότερα εκτελέσιμα αρχεία των Windows περιέχουν πόρους: έναν γενικό όρο που αναφέρεται σε αντικείμενα όπως κέρσορες, εικονίδια, χάρτες *bit*, μενού και γραμματοσειρές. Ένα αρχείο *PE* μπορεί να περιέχει έναν κατάλογο πόρων για όλους τους πόρους που θα χρησιμοποιήσει ο κώδικας προγράμματος σε αυτό το αρχείο. Το κακόβουλο λογισμικό σπάνια χρησιμοποιεί γραφικούς πόρους, επομένως ο συνολικός αριθμός των πόρων τους είναι σχετικά μικρότερος από τον αντίστοιχο αριθμό του μη κακόβουλου λογισμικού [25].

Πολλά πεδία των αρχείων *PE* δεν έχουν κάποιον υποχρεωτικό περιορισμό. Υπάρχει ένας αριθμός περιττών πεδίων και διαστημάτων στα αρχεία *PE* που θα μπορούσαν να δημιουργήσουν ευκαιρίες απόκρυψης κακόβουλου λογισμικού. Στο παρακάτω στιγμιότυπο οθόνης, εκτελείται το *PEView* και φορτώνεται η εικόνα του *.exe* στη μνήμη. Η ενότητα Εργαλεία (*Tools*) δείχνει τα διάφορα τμήματα της μορφής *PE*. Περιγράφεται επίσης το ειδικό

πεδίο *e_magic* της κεφαλίδας *DOS*, το οποίο συνήθως περιέχει την ακολουθία χαρακτήρων *MZ* (που αντιστοιχεί στην ακολουθία *byte* "*0x4D 0x5A*") και το ειδικό πεδίο υπογραφής της κεφαλίδας *PE* (ορίζεται ως η δομή *IMAGE_NT_HEADERS*), που περιέχει την ακολουθία χαρακτήρων *PE* και δηλώνει ότι το δυαδικό αρχείο είναι εγγενές.



Πηγή: Soma Halder, S. O. (2018). Machine Learning for Cybersecurity . Birmingham: Packt.

Εικόνα 6. Εκτελέσιμο αρχείο των Windows.

3.9 DOS header και DOS stub

Η *DOS header* χρησιμοποιείται μόνο για συμβατότητα προς τα πίσω και προηγείται του *DOS stub*, το οποίο εμφανίζει ένα μήνυμα σφάλματος που δηλώνει ότι το πρόγραμμα ενδέχεται να μην εκτελείται σε λειτουργία *DOS*. Σύμφωνα με την επίσημη τεκμηρίωση των *PE* (που βρίσκεται στη διεύθυνση *https:/ / docs. microsoft. com/ en- us/windows/ desktop/ debug/ pe- format#ms- dos- stub- image- only*), το *MS-DOS stub* επιτρέπει στα *Windows* να εκτελέσουν σωστά το αρχείο εικόνας, παρόλο που έχει ένα *MS-DOS stub*. Τοποθετείται στο μπροστινό μέρος της εικόνας exe και εκτυπώνει το μήνυμα, αυτό το πρόγραμμα δεν μπορεί να εκτελεστεί σε λειτουργία DOS, όταν η εικόνα εκτελείται σε *MS-DOS [25]*.

3.9.1 Η δομή της PE header

Μετά την *DOS header* και το *DOS stub*, βρίσκεται η *PE header*. Η *PE header* περιέχει πληροφορίες σχετικά με τα διάφορα τμήματα που χρησιμοποιούνται για την αποθήκευση του κώδικα και των δεδομένων, μαζί με τις αιτούμενες εισαγωγές από άλλες βιβλιοθήκες (*DLL*) ή τις παρεχόμενες εξαγωγές, στην περίπτωση που η ενότητα είναι στην πραγματικότητα μια βιβλιοθήκη.

3.10 Διαχωρίζοντας διαφορετικές οικογένειες κακόβουλου λογισμικού

Παραπάνω έχουν αναφερθεί τα πλεονεκτήματα και οι περιορισμοί που σχετίζονται με την παραδοσιακή μεθοδολογία ανάλυσης κακόβουλου λογισμικού και έτσι γίνεται κατανοητό το γεγονός ότι, είναι απαραίτητο να εισαχθούν αλγοριθμικές μέθοδοι αυτοματισμού για την ανίχνευση κακόβουλων προγραμμάτων. Ειδικότερα, είναι όλο και πιο σημαντικό οι ομοιότητες στη συμπεριφορά του κακόβουλου λογισμικού να είναι σωστά αναγνωρισμένες, που σημαίνει ότι τα δείγματα του κακόβουλου λογισμικού πρέπει να κατανέμονται σε τάξεις ή οικογένειες του ίδιου τύπου. Η ανάλυση των ομοιοτήτων μπορεί να πραγματοποιηθεί σε αυτοματοποιημένη μορφή, χρησιμοποιώντας αλγορίθμους ομαδοποίησης (*clustering algorithms*).

3.10.1 Κατανοώντας τους αλγορίθμους ομαδοποίησης

Η διαίσθηση που βασίζονται οι αλγόριθμοι ομαδοποίησης συνίστανται στον εντοπισμό και την εκμετάλλευση των ομοιοτήτων που χαρακτηρίζουν συγκεκριμένους τύπους φαινόμενων. Από τεχνικής άποψης, είναι θέμα διάκρισης και αναγνώρισης, μέσα σε ένα σύνολο δεδομένων, των χαρακτηριστικών των οποίων οι τιμές αλλάζουν με υψηλή συχνότητα, από εκείνων των οποίων οι τιμές είναι φαίνονται να παραμένουν σταθερές συστηματικά. Λαμβάνονται υπόψη μόνο αυτά τα τελευταία χαρακτηριστικά για την ανίχνευση φαινόμενων που χαρακτηρίζονται από ομοιότητες.

Για τον εντοπισμό των ομοιοτήτων χρησιμοποιούνται οι παρακάτω προσεγγίσεις που έχουν αναλυθεί σε προηγούμενη ενότητα:

- Με επιτήρηση: Οι ομοιότητες εντοπίζονται με βάση προηγούμενες κατηγοριοποιημένα δείγματα (για παράδειγμα, οι αλγόριθμοι *k-Nearest Neighbours* (*k-NNs*)).
- Χωρίς επιτήρηση: Οι ομοιότητες αναγνωρίζονται ανεξάρτητα από τον ίδιο τον αλγόριθμο (για παράδειγμα, ο αλγόριθμος *K-Means*).

Η εκτίμηση της ομοιότητας μεταξύ των χαρακτηριστικών πραγματοποιείται συσχετίζοντάς τα με έναν ορισμό απόστασης. Εάν θεωρούμε τα μεμονωμένα χαρακτηριστικά ως σημεία σε

έναν χώρο με n διαστάσεις (σε συνδυασμό με τον αριθμό των χαρακτηριστικών που αναλύθηκαν), μπορούμε να επιλέξουμε ένα κατάλληλο μαθηματικό κριτήριο για τον υπολογισμό της απόσταση που υπάρχει μεταξύ των μεμονωμένων σημείων. Μερικοί τρόποι μέτρησης που μπορούν να επιλεγούν για τον προσδιορισμό των αποστάσεων μεταξύ αριθμητικών διανυσμάτων είναι οι εξής:

- Ευκλείδεια απόσταση: Αυτή το χαρακτηριστικό προσδιορίζει τη συντομότερη διαδρομή (την ευθεία γραμμή) που ενώνει δύο σημεία στο Καρτεσιανό σύστημα συντεταγμένων και υπολογίζεται με τον ακόλουθο μαθηματικό τύπο:

$$\mathrm{E}(X,Y) = \sqrt{\sum (x-y)^2}$$

Τύπος 1. Soma Halder, S. O. (2018). Machine Learning for Cybersecurity . Birmingham: Packt.

- Απόσταση του Μανχάταν: Αυτό το χαρακτηριστικό λαμβάνεται από το άθροισμα των απόλυτων τιμών των διαφορών υπολογισμένο σε διανύσματα. Σε αντίθεση με την Ευκλείδεια απόσταση, η απόσταση του Μανχάταν προσδιορίζει τη μεγαλύτερη διαδρομή που ενώνει τα δύο σημεία, σύμφωνα με τον παρακάτω τύπο:

$$M(x,y) = \sum \left(\left| x-y \right| \right)$$

Τύπος 2. Soma Halder, S. O. (2018). Machine Learning for Cybersecurity . Birmingham: Packt.

- Απόσταση Chebyshev: Αυτό το χαρακτηριστικό προσδιορίζεται με τον υπολογισμό της μέγιστης τιμής της απόλυτης διαφοράς μεταξύ των στοιχείων των διανυσμάτων όπως ο ακόλουθος τύπος :

$$C(x,y) = \max \left| x-y \right|$$

Τύπος 3. Soma Halder, S. O. (2018). Machine Learning for Cybersecurity . Birmingham: Packt.

Η χρήση της απόστασης Chebyshev είναι ιδιαίτερα χρήσιμη εάν ο αριθμός των διαστάσεων που πρέπει να ληφθούν υπόψη είναι ιδιαίτερα υψηλός [60].

3.10.2 Από τις αποστάσεις στις συστάδες

Η διαδικασία ομαδοποίησης συνίσταται στην κατηγοριοποίηση στοιχείων που δείχνουν συγκεκριμένες ομοιότητες μεταξύ τους. Έχοντας ορίσει την έννοια της ομοιότητας χρησιμοποιώντας διαφορετικούς μαθηματικούς ορισμούς της απόστασης, η διαδικασία ομαδοποίησης μετατρέπεται στην εξερεύνηση των διαφόρων διαστάσεων ενός

συγκεκριμένου χώρου δεδομένων προς κάθε κατεύθυνση, ξεκινώντας από ένα δεδομένο σημείο και μετά συγκεντρώνοντας μαζί τα δείγματα που απέχουν μια συγκεκριμένη απόσταση.

3.10.3 Αλγόριθμοι ομαδοποίησης

Υπάρχουν αρκετοί διαφορετικοί τύποι αλγορίθμων ομαδοποίησης, οι οποίοι είναι κατανοητοί, από τους απλούστερους ως τους πιο περίπλοκους και αφηρημένους. Μερικοί από τους πιο συχνά χρησιμοποιούμενους αλγόριθμους, όπως προαναφέρθηκαν σε προηγούμενη ενότητα, είναι οι εξής:

- *K-Means*: Ένας από τους πιο διαδεδομένους μεταξύ των μη εποπτευόμενων αλγορίθμων ομαδοποίησης. Τα πλεονεκτήματα του *K-Means* είναι ότι μπορεί να κατατάξει με τη χρήση της απλότητάς του στην εφαρμογή και επίσης έχει τη δυνατότητα αποκάλυψης κρυφών μοτίβων στα δεδομένα. Αυτό μπορεί να επιτευχθεί προχωρώντας στον ανεξάρτητο προσδιορισμό των πιθανών ετικετών.
- *K-NNs*: Αυτός είναι ένα παράδειγμα ενός τεμπέλη αλγορίθμου μάθησης. Ο αλγόριθμος *K-NN* αρχίζει να εργάζεται μόνο στη φάση αξιολόγησης, ενώ στη φάση της εκπαίδευσης απλά περιορίζεται στην απομνημόνευση των δεδομένων παρατήρησης. Λόγω αυτών των χαρακτηριστικών του, η χρήση του *k-NN* είναι αναποτελεσματική εξ' αιτίας της παρουσίας μεγάλων συνόλων δεδομένων στις μέρες μας.
- *Density-Based Spatial Clustering of Applications with Noise(DBSCAN*): Σε αντίθεση με τον *K-Means*, που είναι ένας αλγόριθμος που βασίζεται στην απόσταση, ο *DBSCAN* είναι ένας αλγόριθμος με βάση την πυκνότητα. Για παράδειγμα, ο αλγόριθμος προσπαθεί να ταξινομήσει τα δεδομένα προσδιορίζοντας περιοχές υψηλής πυκνότητας [61].

3.10.4 Αξιολόγηση ομαδοποίησης με τον συντελεστή Silhouette

Ένα από τα επαναλαμβανόμενα προβλήματα με τους αλγόριθμους ομαδοποίησης είναι η αξιολόγηση των αποτελεσμάτων. Ενώ στην περίπτωση των εποπτευόμενων αλγορίθμων, γνωρίζοντας ήδη τις ετικέτες ταξινόμησης, οι αναλυτές είναι σε θέση να αξιολογήσουν τα αποτελέσματα που λαμβάνονται από τον αλγόριθμο απλά μετρώντας τον αριθμό δειγμάτων που ταξινομήθηκαν λανθασμένα και συγκρίνοντάς τα με τα σωστά. Στην περίπτωση όμως των αλγορίθμων χωρίς επιτήρηση, η αξιολόγηση των αποτελεσμάτων είναι λιγότερο διαισθητική. Χωρίς οι αναλυτές να διαθέτουν στα χέρια τους τις ετικέτες ταξινόμησης εκ των προτέρων, θα πρέπει να αξιολογήσουν το αποτελέσματα, αναλύοντας τη συμπεριφορά του ίδιου του αλγορίθμου, λαμβάνοντας μόνο ως σωστή τη διαδικασία ομαδοποίησης εάν και μόνο αν τα δείγματα που ταξινομούνται στο ίδιο σύμπλεγμα είναι όλα όμοια. Για τους

αλγόριθμους ομαδοποίησης με βάση την απόσταση, μπορούμε να χρησιμοποιήσουμε ως συντελεστή αξιολόγησης τον συντελεστή Silhouette, ο οποίος παίρνει τον ακόλουθο μαθηματικό τύπο:

$$Sc = \frac{m - n}{max(m, n)}$$

Τύπος 4. Soma Halder, S. O. (2018). Machine Learning for Cybersecurity . Birmingham: Packt.

Εδώ, το *m* αντιπροσωπεύει τη μέση απόσταση που υπάρχει μεταξύ κάθε δείγματος και όλων των άλλων δειγμάτων του πλησιέστερου συμπλέγματος, ενώ το n αντιπροσωπεύει τη μέση απόσταση που υπάρχει μεταξύ κάθε δείγματος και όλων των άλλων δειγμάτων του ίδιου συμπλέγματος. Ο συντελεστής *Silhouette* υπολογίζεται για κάθε μεμονωμένο δείγμα (και είναι επόμενο η διαδικασία του υπολογισμού να γίνεται ιδιαίτερα αργή όταν έχουμε να κάνουμε με μεγάλα σύνολα δεδομένων), και η εκτίμηση της απόσταση γίνεται από τη συγκεκριμένη μέτρηση που έχει επιλεγεί (όπως η ευκλείδεια απόσταση ή η απόσταση Μανχάταν).

Τα κύρια χαρακτηριστικά του συντελεστή *Silhouette* είναι τα ακόλουθα:

- Η τιμή του *Sc* μπορεί να κυμαίνεται μεταξύ -1 και +1, ανάλογα με το πόσο καλή είναι η διαδικασία ομαδοποίησης.
- Η τιμή του *Sc* θα τείνει προς +1 στην περίπτωση της βέλτιστης ομαδοποίησης, ενώ τείνει προς -1 στην αντίθετη περίπτωση της μη βέλτιστης ομαδοποίησης.
- Εάν η τιμή του *Sc* είναι κοντά στο 0, θα υπάρχει παρουσία συμπλεγμάτων που επικαλύπτουν το ένα το άλλο [62].

3.11 Ανάλυση του αλγορίθμου K-Means

Όπως αναφέρθηκε προηγουμένως, ο *K-Means* είναι ένας αλγόριθμος χωρίς επιτήρηση, δηλαδή δεν προϋποθέτει την προηγούμενη γνώση των ετικετών που σχετίζονται με τα δεδομένα. Ο αλγόριθμος παίρνει το όνομά του από το γεγονός ότι ο τελικός σκοπός του είναι να χωρίσει τα δεδομένα σε *k* διαφορετικές υποομάδες. Όντας ένας αλγόριθμος ομαδοποίησης, προχωρά στην υποδιαίρεση των δεδομένων σε διαφορετικές υποομάδες με βάση έναν επιλεγμένο τρόπο μέτρησης απόστασης των μεμονωμένων δειγμάτων (συνήθως, αυτό το μέτρο είναι η Ευκλείδεια απόσταση) από το κέντρο του αντίστοιχου συμπλέγματος. Με άλλα λόγια, ο αλγόριθμος *K-Means* προχωρά στην ομαδοποίηση των δεδομένων σε ξεχωριστές ομάδες, ελαχιστοποιώντας μια συνάρτηση κόστους που αντιπροσωπεύεται από την Ευκλείδεια απόσταση που υπολογίζεται μεταξύ των δεδομένων και τα αντίστοιχα κέντρα τους. Στο τέλος της επεξεργασίας του, ο αλγόριθμος επιστρέφει τα μεμονωμένα

δείγματα που ομαδοποιούνται αντίστοιχα για κάθε σύμπλεγμα, των οποίων τα κέντρα αποτελούν το σύνολο διακριτικών χαρακτηριστικών που αναγνωρίζονται από τον αλγόριθμο ως αντιπροσωπευτικά για κάθε διαφορετική κατηγορία που μπορεί να προσδιοριστεί μέσα στο σύνολο δεδομένων [62].

3.11.1 Βήματα K-Means

Ο αλγόριθμος *K-Means* χαρακτηρίζεται από τα ακόλουθα βήματα:

1. *Αρχικοποίηση*: Αυτή είναι η φάση στην οποία τα κέντρα εντοπίζονται στη βάση από τον αριθμό των συστάδων που ορίζονται από τον αναλυτή (συνήθως, δεν μπορούμε να γνωρίζουμε τον αριθμό των πραγματικών συστάδων εκ των προτέρων, επομένως είναι συχνά απαραίτητο να προχωρήσουμε στη διαδικασία της δοκιμής και σφάλματος κατά τον καθορισμό του αριθμού των συστάδων).

2. *Αντιστοίχηση δεδομένων στις συστάδες*: Με βάση τον ορισμό των κέντρων που μεταφέρονται στη φάση προετοιμασίας, τα δεδομένα αντιστοιχίζονται στο πλησιέστερο σύμπλεγμα, με βάση την ελάχιστη Ευκλείδεια απόσταση που υπολογίζεται μεταξύ των δεδομένων και των αντίστοιχων κέντρων τους.

3. *Ενημέρωση Κέντρων*: Όντας μια επαναληπτική διαδικασία, ο αλγόριθμος *K-Means* προχωρά και πάλι στην εκτίμηση των κέντρων με την εκτίμηση του μέσου όρου των δεδομένων που περιλαμβάνονται στις μεμονωμένες συστάδες. Στη συνέχεια, ο αλγόριθμος προχωρά στην εκ νέου ανάθεση του μέσου όρου, μέχρις ότου η Ευκλείδεια απόσταση μεταξύ των δεδομένων και των αντίστοιχων κέντρων δεν ελαχιστοποιείται ή ο αριθμός των επαναλήψεων που ορίζονται από τον αναλυτή ως παράμετρος εισόδου δεν έχει υπερβληθεί [62].

Για να την χρησιμοποίηση του αλγορίθμου *K-Means* που συνοδεύεται από την *scikit-learn* βιβλιοθήκη, πρέπει να γίνει η κατάλληλη επιλογή μιας σειράς παραμέτρων εισόδου για να καθοριστούν οι φάσεις της επαναληπτικής διαδικασίας του αλγορίθμου, όπως προσδιορίστηκε προηγουμένως. Ειδικότερα, θα είναι απαραίτητο να προσδιοριστεί ο αριθμός των συστάδων k και ο τρόπος αρχικοποίησης των κέντρων. Η επιλογή του αριθμού των συστάδων από τον αναλυτή έχει συνέπειες στο αποτέλεσμα που θα επιτευχθεί από τον αλγόριθμο: εάν ο αριθμός των συστάδων που έχει οριστεί ως παράμετρος αρχικοποίησης είναι πολύ υψηλός, τότε αγνοείται ο σκοπός της ομαδοποίησης (η συμπεριφορά του αλγορίθμου στο όριο τείνει να προσδιορίζει ένα εντελώς διαφορετικό σύμπλεγμα για κάθε δεδομένο). Για το σκοπό αυτό, μπορεί να είναι χρήσιμο να διεξαχθεί μια προκαταρκτική φάση διερευνητικής ανάλυσης δεδομένων (*EDA*) – η οποία πραγματοποιείται με τη βοήθεια της σχεδίασης δεδομένων - προσδιορίζοντας οπτικά τον αριθμό των πιθανών ξεχωριστών υποομάδων στις οποίες μπορούν να διανεμηθούν τα δεδομένα [62].

3.11.2 Πλεονεκτήματα και μειονεκτήματα του αλγορίθμου K - Means

Μεταξύ των πλεονεκτημάτων του αλγορίθμου *K-Means*, εκτός από την απλότητα στη χρήση του, είναι και η υψηλή επεκτασιμότητά του που τον καθιστά προτιμότερο στην παρουσία μεγάλων συνόλων δεδομένων. Τα μειονεκτήματα αντ 'αυτού οφείλονται ουσιαστικά στην ακατάλληλη επιλογή της παραμέτρου *k* , που αντιπροσωπεύει τον αριθμό των συστάδων, η οποία, όπως έχει αναφερθεί, απαιτεί ιδιαίτερη προσοχή εκ μέρους του αναλυτή, ο οποίος θα κληθεί να το αξιολογήσει προσεκτικά με βάση την *EDA* ή προχωρώντας στη διαδικασία των δοκιμών και των σφαλμάτων. Ένα άλλο μειονέκτημα που σχετίζεται με τη χρήση του αλγορίθμου *K-Means* καθορίζεται από το γεγονός ότι παρέχει ελάχιστα αντιπροσωπευτικά αποτελέσματα στην παρουσία συνόλων δεδομένων το οποίο χαρακτηρίζεται από υψηλές διαστάσεις [63].

3.12 Ανίχνευση κακόβουλου λογισμικού με δέντρα απόφασης

Εκτός από τους αλγόριθμους ομαδοποίησης, είναι δυνατόν να χρησιμοποιηθούν αλγόριθμοι ταξινόμησης για την ανίχνευση απειλών κακόβουλου λογισμικού. Ιδιαίτερη σημασία έχει η ταξινόμηση του κακόβουλου λογισμικού που πραγματοποιείται χρησιμοποιώντας τα δέντρα αποφάσεων. Το διακριτικό χαρακτηριστικό των δέντρων απόφασης είναι ότι αυτοί οι αλγόριθμοι επιτυγχάνουν τον στόχο της ταξινόμησης των δεδομένων σε ορισμένες κλάσεις με τη μοντελοποίηση της μαθησιακής διαδικασίας βάσει μιας ακολουθίας εάν-τότε-αλλιώς αποφάσεων. Για αυτό τους το χαρακτηριστικό, τα δέντρα αποφάσεων αντιπροσωπεύουν έναν τύπο μη γραμμικού ταξινομητή.

3.12.1 Στρατηγική ταξινόμησης των δέντρων απόφασης

Τα δέντρα αποφάσεων, διαμορφώνουν τη μαθησιακή τους διαδικασία βάσει μιας δομής δέντρου. Ξεκινώντας από έναν αρχικό κόμβο και οι επόμενες αποφάσεις διακλαδίζονται σε διάφορους κλάδους διαφορετικών βάσεων. Στην ουσία, το σύνολο δεδομένων των δειγμάτων διαιρείται από τον αλγόριθμο με επαναληπτικό τρόπο, με βάση τις αποφάσεις που λαμβάνονται σε κάθε κόμβο, δημιουργώντας έτσι τους διάφορους κλάδους. Οι διακλαδώσεις, από την άλλη πλευρά, δεν αντιπροσωπεύουν τίποτα περισσότερο από τους διάφορους τρόπους με τους οποίους τα δεδομένα μπορούν να είναι ταξινομημένα, με βάση τις πιθανές επιλογές που έγιναν στους διάφορους κόμβους απόφασης. Αυτή η επαναληπτική διαδικασία υποδιαίρεσης του συνόλου δεδομένων καθορίζεται από ένα προκαθορισμένο μέτρο της ποιότητας των συνθηκών υποδιαίρεσης. Οι μετρήσεις που χρησιμοποιούνται πιο συχνά για τη μέτρηση της ποιότητα της υποδιαίρεσης είναι οι εξής:

- Σκουπίδια Gini
- Μείωση διακύμανσης

- Κέρδος πληροφοριών

Παρά την υψηλή επεξηγηματική τους ικανότητα, τα δέντρα αποφάσεων, ωστόσο, υποφέρουν από κάποιους σημαντικούς περιορισμούς:

- Καθώς αυξάνεται ο αριθμός των χαρακτηριστικών, η πολυπλοκότητα της δομής που αναπαριστά το σχετικό δέντρο αποφάσεων μεγαλώνει ανάλογα, μεταφράζοντας αυτήν την πολυπλοκότητα στο φαινόμενο που είναι γνωστό ως *overfitting* (δηλαδή, ο αλγόριθμος τείνει να μοντελοποιεί τον θόρυβο στα δεδομένα, παρά το σήμα, οδηγώντας σε λιγότερο ακριβείς προβλέψεις στα δεδομένα δοκιμής)
- Τα δέντρα απόφασης είναι ιδιαίτερα ευαίσθητα ακόμη και σε μικρές παραλλαγές στα δείγματα δεδομένων, κάνοντας τις προβλέψεις ασταθείς

Ένας τρόπος για να ξεπεραστούν οι παραπάνω περιορισμοί είναι η δημιουργία συνόλων δέντρων, δίνοντας μια ψήφο σε κάθε δέντρο. Ο μηχανισμός για την εκχώρηση δειγμάτων στις αντίστοιχες τάξεις είναι επομένως ελαφρυμένος στη μέτρηση των ψήφων που δόθηκαν από τα διάφορα δέντρα, ένα παράδειγμα συνόλου δέντρων είναι ο αλγόριθμος τυχαίων δασών [64].

3.13 Τυχαία δάση

Τα δέντρα αποφάσεων υποφέρουν από ορισμένους σημαντικούς περιορισμούς, οι οποίοι μπορούν να οδηγήσουν σε ασταθή αποτελέσματα που προκαλούνται ακόμη και από μικρές διαφορές στα δεδομένα εκπαίδευσης. Για τη βελτίωση των προβλέψεων, μπορεί να γίνει χρήση αλγορίθμων συνόλου, όπως είναι το τυχαίο δάσος. Το τυχαίο δάσος δεν είναι τίποτα άλλο από ένα σύνολο δέντρων αποφάσεων στο οποίο κάθε δέντρο έχει μια ψήφο. Η βελτίωση των προβλέψεών τους καθορίζεται από τον αριθμό των ψήφων που αποδίδεται σε αυτές: οι προβλέψεις που λαμβάνουν τον υψηλότερο αριθμό ψήφων είναι αυτές που έχουν επιλεχθεί για να επιτευχθεί το τελικό αποτέλεσμα του αλγορίθμου. Ο δημιουργός του αλγορίθμου *Random Forest*, *Leo Breiman*, σημείωσε ότι τα αποτελέσματα που ελήφθησαν από ένα σύνολο δέντρων βελτιωνόταν εάν τα δέντρα δεν ήταν συσχετισμένα στατιστικά και ήταν ανεξάρτητα το ένα από το άλλο [25].

3.14 Εντοπισμός μεταμορφικού κακόβουλου λογισμικού με HMMs (Hidden Markov Models)

Τα παραδείγματα αλγορίθμων που έχουν εφαρμογή στην ανίχνευση κακόβουλου λογισμικού που έχουν παρουσιαστεί μέχρι στιγμής προορίζονταν να αυτοματοποιήσουν ορισμένες από τις δραστηριότητες ρουτίνας που εκτελούνται από αναλυτές κακόβουλου λογισμικού. Ωστόσο, η μεθοδολογία ανάλυσης στην οποία βασίζονται είναι ουσιαστικά το στατικό κακόβουλο λογισμικό. Πολλές από τις συγκεκριμένες περιπτώσεις απειλών

κακόβουλου λογισμικού δεν είναι εύκολα αναγνωρίσιμες με τη συγκεκριμένη μέθοδο ανάλυσης, καθώς οι προγραμματιστές κακόβουλου λογισμικού έχουν μάθει πώς να εργάζονται γύρω από τις τεχνικές ανίχνευσης με βάση τις υπογραφές. Είναι επομένως απαραίτητο να υιοθετηθεί μια διαφορετική μεθοδολογία για τον εντοπισμό της κακόβουλης συμπεριφοράς πιο προηγμένου κακόβουλου λογισμικού, και για το σκοπό αυτό, θα πρέπει να υιοθετηθεί μια προσέγγιση η οποία βασίζεται στη δυναμική ανάλυση κακόβουλου λογισμικού, συνδυάζοντάς το με τους κατάλληλους αλγόριθμους. Αλλά για να αντιμετωπιστεί επαρκώς το πρόβλημα, είναι απαραίτητη η κατανόηση των ορίων των παραδοσιακών στρατηγικών ανίχνευσης με βάση τις υπογραφές [65].

3.15 Πώς το κακόβουλο πρόγραμμα παρακάμπτει τον εντοπισμό;

Η πιο συχνά χρησιμοποιούμενη στρατηγική ανίχνευσης είναι αυτή που χρησιμοποιεί υπογραφές που σχετίζονται με τα εκτελέσιμα αρχεία που αναγνωρίζονται ως κακόβουλα. Αυτή η στρατηγική προσφέρει αναμφισβήτητα πλεονεκτήματα και εφαρμόζεται ευρέως από τα προγράμματα προστασίας ιών. Βασίζεται στην αναζήτηση συγκεκριμένων μοτίβων, σε καθένα από τα αρχεία που είναι αποθηκευμένα στο σύστημα και πραγματοποιούν τη συστηματική σάρωση των πόρων (συμπεριλαμβανομένης της μνήμης χρόνου εκτέλεσης) του συστήματος. Η αναζήτηση μοτίβων πραγματοποιείται με βάση μια βάση δεδομένων, η οποία περιέχει τις υπογραφές των κακόβουλων αρχείων. Αυτά πρέπει να ενημερώνονται άμεσα και συνεχώς, προκειμένου να είναι σε θέση να αναζητούν και να συγκρίνουν αρχεία στο σύστημα, αποτρέποντας έτσι τις απειλές να μην εντοπιστούν. Τα πλεονεκτήματα που συνδέονται με τη στρατηγική ανίχνευσης βάσει υπογραφής είναι ουσιαστικά τα επόμενα:

- Αποδοτικότητα στον εντοπισμό απειλών που είναι ήδη γνωστές και υπάρχουν στη βάση δεδομένων με τις υπογραφές.
- Η χαμηλή συχνότητα ψευδών θετικών, η οποία μαζί με τα ψευδώς αρνητικά, είναι η κύρια αδυναμία του λογισμικού εντοπισμού κακόβουλου λογισμικού.

Τα όρια αυτής της στρατηγικής ανίχνευσης αντιπροσωπεύονται ουσιαστικά από τη βασική υπόθεση: ότι δηλαδή το κακόβουλο λογισμικό, μόλις αναγνωριστεί, δεν αλλάζει δυαδική αναπαράσταση, επομένως θεωρείται ότι έχει φωτογραφηθεί επαρκώς από την αντίστοιχη υπογραφή. Στην πραγματικότητα, αυτές οι υποθέσεις γρήγορα αποδείχθηκαν μη ρεαλιστικές. Με την πάροδο του χρόνου, έχουμε παρακολουθήσει τη δημιουργικότητα των προγραμματιστών κακόβουλου λογισμικού να προσπαθούν να δημιουργήσουν λογισμικό που ήταν σε θέση να αλλάξει το σχήμα του, στοχεύοντας έτσι τον μηχανισμό ανίχνευσης με βάση τις υπογραφές, διατηρώντας τη δική του επιθετική δυναμική. Ένα από τα πρώτα αντίμετρα που υιοθέτησαν οι δημιουργοί του κακόβουλου λογισμικού ήταν η συσκότιση. Για το σκοπό αυτό, είναι δυνατή η εκτέλεση της κρυπτογράφησης των εκτελέσιμων τμημάτων ενός κακόβουλου λογισμικού, κάθε φορά χρησιμοποιώντας διαφορετικά κλειδιά κρυπτογράφησης ώστε να αλλάξουν τις υπογραφές που σχετίζονται με το antivirus

λογισμικό στο φορτίο του κακόβουλου λογισμικού, ενώ οι εκτελέσιμες οδηγίες παραμένουν αναλλοίωτες και αποκρυπτογραφούνται πριν αποσταλούν για εκτέλεση. Μια πιο εξελιγμένη παραλλαγή της συσκότισης είναι η δημιουργία πολυμορφικού κακόβουλου λογισμικού το οποίο όχι μόνο αλλάζει συνεχώς το κλειδί κρυπτογράφησης κακόβουλου λογισμικού, αλλά και οι ίδιες οι οδηγίες αποκρυπτογράφησης του κακόβουλου λογισμικού αλλάζουν.

Η επακόλουθη εξέλιξη του πολυμορφικού κακόβουλου λογισμικού οδηγεί σε μεταμορφικό κακόβουλο λογισμικό, στο οποίο ακόμη και οι εκτελέσιμες οδηγίες του φορτίου τροποποιούνται σε κάθε εκτέλεση, εμποδίζοντας έτσι τα πιο προηγμένα antivirus να εντοπίσουν το κακόβουλο φορτίο με τη σάρωση της μνήμης χρόνου εκτέλεσης, μετά την αποκρυπτογράφηση του φορτίου. Προκειμένου να τροποποιηθούν οι εκτελέσιμες οδηγίες του φορτίου, το μεταμορφικό κακόβουλο λογισμικό εφαρμόζει έναν τρόπο μετάλλαξης υιοθετώντας τις ακόλουθες μεθόδους:

- Εισαγωγή πρόσθετων οδηγιών (νεκρός κωδικός) που δεν αλλάζουν τη λογική και τη λειτουργία του κακόβουλου λογισμικού.
- Αλλαγή της σειράς των οδηγιών, χωρίς αλλαγή της λογικής και της συνολικής λειτουργικότητας. Αυτή η τεχνική είναι ιδιαίτερα αποτελεσματική στη δημιουργία πολλών παραλλαγών στο θέμα.
- Αντικατάσταση ορισμένων οδηγιών με άλλες ισοδύναμες οδηγίες [61].

3.16 Στρατηγικές αναγνώρισης πολυμορφικού κακόβουλου λογισμικού

Στο συνεχή αντιπαράθεση που δημιουργήθηκε μεταξύ των προγραμματιστών κακόβουλου λογισμικού και των παραγωγών antivirus λογισμικού, οι τελευταίοι προσπάθησαν να συνεχίσουν το ρυθμό, προσαρμόζοντας την ανίχνευσή τους σε στρατηγικές για τις διάφορες μορφές πολυμορφισμού. Στην περίπτωση του πολυμορφικού κακόβουλου λογισμικού, μία από τις στρατηγικές που υιοθετούνται αποτελείται από κώδικα εξομοίωσης: η εκτέλεση του κακόβουλου λογισμικού σε ένα ελεγχόμενο περιβάλλον (όπως το *sandbox*), επιτρέποντας στο κακόβουλο λογισμικό να πραγματοποιήσει τη φάση αποκρυπτογράφησης του κακόβουλου φορτίου του, την οποία ακολουθεί η παραδοσιακή ανίχνευση βάσει υπογραφής που εκτελείται από το λογισμικό προστασίας από ιούς.

Στην περίπτωση μεταμορφικού κακόβουλου λογισμικού, η δραστηριότητα ανίχνευσης που πραγματοποιήθηκε από το πιο εξελιγμένο λογισμικό προστασίας από ιούς προσπαθεί να αναλύσει τη συμπεριφορά του ύποπτου αρχείου, κατανοώντας τη λογική των οδηγιών που εκτελούνται. Ωστόσο, αυτή η στρατηγική ανίχνευσης υποφέρει από μερικούς από τους ακόλουθους σημαντικούς περιορισμούς:

- Αυτό οδηγεί σε υψηλό ποσοστό ψευδών θετικών

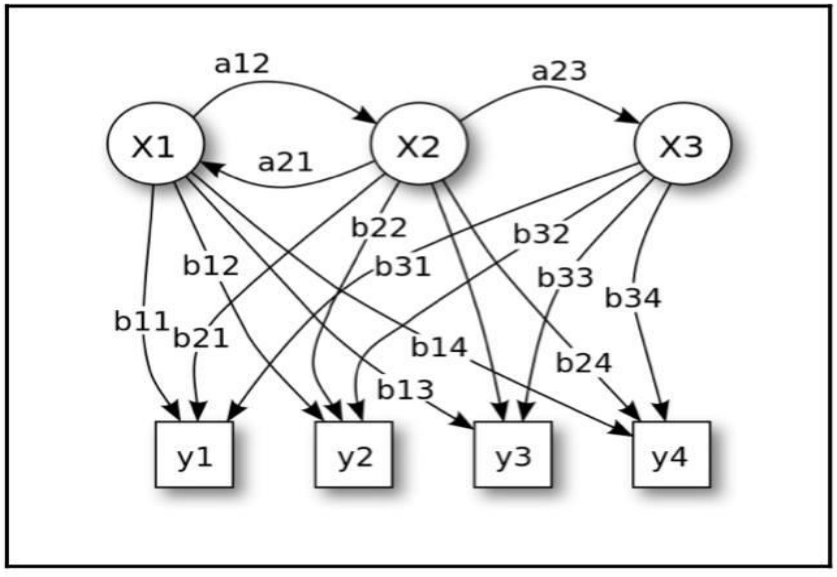
- Η ανάλυση των οδηγιών που εκτελούνται πραγματοποιείται «εν κινήσει», η οποία μπορεί να οδηγήσει σε σημαντικές επιπτώσεις σε υπολογιστικούς όρους

Μια εναλλακτική στρατηγική για την ανίχνευση μεταμορφικών κακόβουλων λογισμικών είναι αυτή που χρησιμοποιεί αλγόριθμους *ML* που βασίζονται σε *HMMs [66]*.

3.17 Οι βασικές αρχές των HMMs

Για την κατανόηση των *HMMs*, πρέπει να γίνει εισαγωγή των διαδικασιών *Markov*. Μια διαδικασία *Markov* (ή αλυσίδα *Markov*) είναι ένα στοχαστικό μοντέλο που αλλάζει την κατάστασή του με βάση ένα προκαθορισμένο σύνολο πιθανοτήτων. Μία από τις παραδοχές της διαδικασίας *Markov* ορίζει ότι η κατανομή πιθανότητας των μελλοντικών καταστάσεων εξαρτάται αποκλειστικά από την τρέχουσα κατάσταση. Επομένως, ένα *HMM* είναι μια διαδικασία *Markov* της οποίας δεν είναι δυνατή η άμεση παρατήρηση της κατάστασης του συστήματος: τα μόνα παρατηρήσιμα στοιχεία είναι τα γεγονότα και τα δευτερεύοντα αποτελέσματα που σχετίζονται με την κατάσταση του συστήματος. Ωστόσο, οι πιθανότητες των γεγονότων καθορίζονται από κάθε κατάσταση του συστήματος είναι σταθερές.

Κατά συνέπεια, οι παρατηρήσεις σε κάθε κατάσταση του συστήματος γίνονται έμμεσα στη βάση των συμβάντων που καθορίζονται από κρυφές καταστάσεις, των οποίων οι εκτιμήσεις πιθανότητας μπορούν να σχετίζονται :



Πηγή: A Revealing Introduction to Hidden Markov Models by Mark Stamp, San Jose State University

Εικόνα 7. Hidden Markov Model

Για την διαισθητική κατανόηση της λειτουργίας των *HMMs*, παρουσιάζεται το ακόλουθο παράδειγμα: έστω ένα εκτελέσιμο αρχείο που ξεκινά σε έναν κεντρικό υπολογιστή. Σε μια

δεδομένη στιγμή, το μηχάνημα μπορεί να συνεχίσει να λειτουργεί σωστά ή να σταματήσει να λειτουργεί σωστά. Αυτή η συμπεριφορά αντιπροσωπεύει το παρατηρήσιμο συμβάν. Ας υποθέσουμε για απλότητα ότι οι λόγοι για τους οποίους το μηχάνημα σταματά να λειτουργεί τακτικά μπορεί να μειωθεί στους ακόλουθους:

- Το εκτελέσιμο εκτελούσε μια κακόβουλη οδηγία
- Το εκτελέσιμο εκτελούσε μια νόμιμη εντολή

Οι πληροφορίες που σχετίζονται με τον συγκεκριμένο λόγο για τον οποίο το μηχάνημα σταματά να λειτουργεί σωστά είναι άγνωστες σε εμάς, μπορούμε να συμπεράνουμε μόνο με βάση παρατηρήσιμα γεγονότα. Αυτά τα παρατηρήσιμα γεγονότα, στο παράδειγμά μας, μειώνονται στα εξής:

- Το μηχάνημα λειτουργεί τακτικά (λειτουργεί)
- Το μηχάνημα σταματά να λειτουργεί (δεν λειτουργεί)

Ομοίως, οι κρυφές οντότητες του παραδείγματος μας αντιπροσωπεύονται από τις οδηγίες που εκτελούνται από το πρόγραμμα:

- Κακόβουλη οδηγία
- Νόμιμη οδηγία

3.18 Προηγμένη ανίχνευση κακόβουλου λογισμικού με deep learning

Στο τελευταίο μέρος του κεφαλαίου, θα παρουσιαστούν μερικές λύσεις εντοπισμού κακόβουλου λογισμικού που χρησιμοποιούν πειραματικές μεθοδολογίες βασισμένες στα νευρωνικά δίκτυα. Εδώ, θα παρουσιαστεί μια καινοτόμος και μη συμβατική προσέγγιση στο πρόβλημα της ταξινόμησης διαφορετικών οικογενειών κακόβουλου λογισμικού, το οποίο χρησιμοποιεί *deep learning* αλγορίθμους που αναπτύχθηκαν σε ένα εντελώς διαφορετικό πεδίο έρευνας, όπως αυτό της αναγνώρισης εικόνας χρησιμοποιώντας *Convolutional Neural Networks (CNN*). Πριν όμως προχωρήσουμε σε αυτό, θα παρουσιαστούν εν συντομία τα Νευρωνικά Δίκτυα (*NN*) και οι κύριες λειτουργίες τους στον τομέα της ανίχνευσης κακόβουλου λογισμικού [66].

3.19 Τα νευρωνικά δίκτυα με λίγα λόγια

Τα *NN* αποτελούν μια κατηγορία αλγορίθμων που προσπαθούν να μιμηθούν τους τυπικούς μηχανισμούς μάθησης του ανθρώπινου εγκεφάλου, αναπαράγοντας τεχνητά το υπόστρωμά του, το οποίο αποτελείται από νευρώνες. Υπάρχουν διαφορετικοί τύποι νευρωνικών

δικτύων, αλλά εδώ εστιάζουμε σε δύο τύπους συγκεκριμένα: τα *CNN* και τα *Feedforward Networks (FFN),* τα οποία αποτελούν τη βάση των *CNN* .

Τα *FFN* αποτελούνται από τουλάχιστον τρία στρώματα νευρώνων, χωρισμένα ως εξής:

1. Επίπεδο εισόδου
2. Επίπεδο εξόδου
3. Κρυφό στρώμα (ένα ή περισσότερα)

Αυτή η πολυεπίπεδη οργάνωση του *FFN* μας επιτρέπει να έχουμε ένα πρώτο επίπεδο για τη διαχείριση των δεδομένων εισόδου και ένα επίπεδο που επιστρέφει τα αποτελέσματα εξόδου. Οι μεμονωμένοι νευρώνες στα διάφορα στρώματα συνδέονται απευθείας με τα γειτονικά στρώματα, ενώ δεν υπάρχουν συνδέσεις μεταξύ των νευρώνων που ανήκουν στο ίδιο στρώμα [66].

3.19.1 CNNs

Το *CNN* είναι ένας συγκεκριμένος τύπος *FFN*, που χαρακτηρίζεται από το γεγονός ότι η οργάνωση των νευρωνικών στρωμάτων του ακολουθούν την ίδια οργάνωση της υπάρχουσας οπτικής συσκευής στον βιολογικό κόσμο, με την προσθήκη περιοχών νευρώνων στο οπτικό πεδίο. Όπως προαναφέρθηκε, στα *CNN*, κάθε νευρώνας συνδέεται με μια γειτονική περιοχή νευρώνων εισόδου, για να χαρτογραφήσουν τις αντίστοιχες περιοχές των εικονοστοιχείων (*pixels*) μιας εικόνας. Με αυτόν τον τρόπο, είναι δυνατή η αναγνώριση των χωρικών συσχετίσεων μέσω της τοπικής συνδεσιμότητας μεταξύ των νευρώνων που βρίσκονται σε γειτονικά στρώματα, τα οποία επιτρέπουν, για παράδειγμα, την αναγνώριση αντικειμένων.

Στα *CNN*, οι γειτονικές περιοχές των νευρώνων οργανώνονται στην πραγματικότητα με τη μίμηση του τρισδιάστατου χώρου με τις τιμές του πλάτους, του ύψους και του βάθους, οι οποίες αντιστοιχούν στα χαρακτηριστικά του πλάτους και του ύψους των εικόνων, ενώ το βάθος αποτελείται από τα κανάλια *RGB*.

Επομένως, τα *CNN* είναι βελτιστοποιημένα για αναγνώριση εικόνας χάρη στο συνελικτικό επίπεδό τους. Συγκεκριμένα, το συνελικτικό επίπεδο μας επιτρέπει να εξαγάγουμε τα σχετικά χαρακτηριστικά των εικόνων εισόδου μέσω των εργασιών συνέλιξης, οι οποίες δημιουργούν μια νέα εικόνα ξεκινώντας από τη πρωτότυπη εικόνα, επισημαίνοντας τις πιο σχετικές λειτουργίες και θολώνοντας τις λιγότερο σχετικές. Με αυτόν τον τρόπο, το συνελικτικό επίπεδο μπορεί να εντοπίσει παρόμοιες εικόνες παρά την πραγματική τους θέση ή τον προσανατολισμό. Η περαιτέρω ανάλυση των αλγορίθμων *deep learning* δεν ανήκει στα πλαίσια της παρούσας εργασίας [67].

ΚΕΦΑΛΑΙΟ 4

ΣΧΕΔΙΑΣΗ ΚΑΙ ΜΕΘΟΔΟΛΟΓΙΑ ΚΙΝΗΣΕΩΝ

Σε αυτό το κεφάλαιο της εργασίας, πρόκειται να δημιουργήσουμε ένα φίλτρο ανεπιθύμητης αλληλογραφίας χρησιμοποιώντας την *Python* και τον πολυμερή αλγόριθμο *Naive Bayes* και στη συνέχεια θα αναπτυχθεί. Στόχος μας είναι να κωδικοποιήσουμε ένα φίλτρο ανεπιθύμητης αλληλογραφίας που ταξινομεί τα μηνύματα με ακρίβεια μεγαλύτερη από 80%.
Για να δημιουργήσουμε το φίλτρο ανεπιθύμητων μηνυμάτων, θα χρησιμοποιήσουμε ένα σύνολο δεδομένων 5.572 μηνυμάτων SMS το οποίο μπορεί να βρεθεί στην σελίδα του UCI Machine Learning Repository.

4.1 SMS spam filter

4.1.1 Εξερευνώντας το σύνολο δεδομένων

Ξεκινάμε ανοίγοντας το αρχείο *SMSSpamCollection* με τη συνάρτηση *read_csv()* από το πακέτο *pandas*. Στη συνέχεια θα χρησιμοποιήσουμε:

- *Sep = ΄\t’* επειδή τα σημεία δεδομένων είναι χωρισμένα ανά καρτέλα
- *Header = None* επειδή το σύνολο δεδομένων δεν διαθέτει γραμμή κεφαλίδας
- *Names = [‘Label’ , ‘SMS’]* για την ονομασία των στηλών

```
import pandas as pd

sms_spam = pd.read_csv('SMSSpamCollection', sep='\t',
header=None, names=['Label', 'SMS'])

print(sms_spam.shape)
sms_spam.head()
```

Πηγή: Max Kuhn, Kjell Johnson. Applied Predictive Modeling 1st ed. 2013, Corr. 2nd printing 2018 Edition

Κομμάτι Κώδικα 2.

>>(5572,2)

	Label	SMS
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

Πίνακας 1 Δεδομένα 'SMSSpamCollection'

Παρακάτω, βλέπουμε ότι περίπου το 87% των μηνυμάτων είναι ham (non-spam) και το υπόλοιπο 13% είναι spam. Αυτό το δείγμα φαίνεται αντιπροσωπευτικό, καθώς στην πράξη τα περισσότερα μηνύματα που λαμβάνουν οι άνθρωποι είναι ham.

	Label	secret	prize	claim	now	coming	to	my	party	winner
0	spam	2	2	1	1	0	0	0	0	0
1	ham	1	0	0	0	1	1	1	1	0
2	spam	1	1	1	1	0	0	0	0	1

```
sms_spam['Label'].value_counts(normalize=True)
```

>>ham 0.865937
>>spam 0.134063
>>Name: Label , dtype : float64

4.1.2 Δεδομένα εκπαίδευσης και δοκιμών

Τώρα πρόκειται να χωρίσουμε το σύνολο δεδομένων μας σε ένα σετ εκπαίδευσης και ένα σετ δοκιμών. Θα χρησιμοποιήσουμε το 80% των δεδομένων για εκπαίδευση και το υπόλοιπο 20% για δοκιμές.

Θα ανακατέψουμε τυχαία ολόκληρο το σύνολο δεδομένων πριν από τον διαχωρισμό για να διασφαλίσουμε ότι τα μηνύματα ανεπιθύμητης αλληλογραφίας spam και τα ham εμφανίζονται σωστά σε ολόκληρο το σύνολο δεδομένων.

```
data_randomized = sms_spam.sample(frac=1, random_state=1)

training_test_index = round(len(data_randomized) * 0.8)

training_set =
data_randomized[:training_test_index].reset_index(drop=True)
test_set =
data_randomized[training_test_index:].reset_index(drop=True)
```

```
print(training_set.shape)
print(test_set.shape)
```

Πηγή: Max Kuhn, Kjell Johnson. Applied Predictive Modeling 1st ed. 2013, Corr. 2nd printing 2018 Edition

Κομμάτι Κώδικα 3 .

```
>> (4458, 2)
>> (1114, 2)
```

Θα αναλύσουμε τώρα το ποσοστό των ανεπιθύμητων μηνυμάτων spam και των μηνυμάτων ham στα σετ εκπαίδευσης και δοκιμών. Αναμένουμε ότι τα ποσοστά θα είναι κοντά σε αυτό που έχουμε στο πλήρες σύνολο δεδομένων, όπου περίπου το 87% των μηνυμάτων είναι ham, και το υπόλοιπο 13% είναι spam.

```
training_set['Label'].value_counts(normalize=True)
test_set['Label'].value_counts(normalize=True)
```

Κομμάτι Κώδικα 4 . Max Kuhn, Kjell Johnson. Applied Predictive Modeling 1st ed. 2013, Corr. 2nd printing 2018 Edition

>> ham 0.86541
>> spam 0.13459
>> Name: Label, dtype: float 64

>> ham 0.868043
>> spam 0.131957
>> Name: Label, dtype: float64

4.1.3 Καθαρισμός δεδομένων

Όταν εισέλθει ένα νέο μήνυμα, ο αλγόριθμος *Naive Bayes* θα κάνει την ταξινόμηση με βάση τα αποτελέσματα που παίρνει από τις δύο εξισώσεις των Naïve Bayes Classifier παρακάτω, όπου το "*w1*" είναι η πρώτη λέξη και *w1, w2, ..., wn* είναι το συνολικό μήνυμα:

$$P(Spam \mid w_1, w_2, ..., w_n) \propto P(Spam) \cdot \prod_{i=1}^{n} P(w_i \mid Spam)$$

$$P(Ham \mid w_1, w_2, ..., w_n) \propto P(Ham) \cdot \prod_{i=1}^{n} P(w_i \mid Ham)$$

Εάν το P(Spam | w_1,w_2, …, w_n) είναι μεγαλύτερο από το P(Ham | w_1, w_2, …, w_n), τότε το συγκεκριμένο μήνυμα ανήκει στην κατηγορία spam.
Για τον υπολογισμό των P(w_i|Spam) και P(w_i|Ham) πρέπει να χρησιμοποιηθούν οι παρακάτω εξισώσεις:

$$P(w_i|Spam) = \frac{N_{w_i|Spam} + a}{N_{Spam} + a \cdot N_{Vocabulary}}$$

$$P(w_i|Ham) = \frac{N_{w_i|Ham} + a}{N_{Ham} + a \cdot N_{Vocabulary}}$$

Πηγή :Sunil Ray. 6 Easy Steps to Learn Naive Bayes Algorithm with codes in Python and R.
https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/

Αναλυτικότερα :

$N_{w_i|Spam}$ = ο αριθμός των φορών που η λέξη w_i εμφανίζεται στα μηνύματα spam

$N_{w_i|Ham}$ = ο αριθμός των φορών που η λέξη w_i εμφανίζεται στα μηνύματα ham

N_{Spam} = ο συνολικός αριθμός των λέξεων σε όλα τα μηνύματα spam

N_{Ham} = ο συνολικός αριθμός των λέξεων σε όλα τα μηνύματα ham

$N_{Vocabulary}$ = ο συνολικός αριθμός των λέξεων στο λεξιλόγιο

$a = 1$ = το α είναι μια παράμετρος εξομάλυνσης

Για να τον υπολογισμό όλων αυτών των πιθανοτήτων, θα πρέπει πρώτα να γίνει καθαρισμός στα δεδομένα ώστε να έρθουν σε μορφή που μας επιτρέπει να εξάγουμε εύκολα όλες τις πληροφορίες που χρειαζόμαστε. Αυτήν τη στιγμή, τα σετ εκπαίδευσης και δοκιμών έχουν αυτήν τη μορφή (τα παρακάτω μηνύματα είναι φανταστικά για να κάνουν το παράδειγμα πιο κατανοητό):

	Label	SMS
0	spam	SECRET PRIZE! CLAIM SECRET PRIZE NOW!!
1	ham	Coming to my secret party?
2	spam	Winner! Claim secret prize now!

Πίνακας 2 Μορφή πίνακα δεδομένων πριν τον καθορισμό

	Label	secret	prize	claim	now	coming	to	my	party	winner
0	spam	2	2	1	1	0	0	0	0	0
1	ham	1	0	0	0	1	1	1	1	0
2	spam	1	1	1	1	0	0	0	0	1

Πίνακας 3 Μορφή πίνακα δεδομένων μετά τον καθορισμό

Για να γίνουν ευκολότερος οι υπολογισμοί, πρέπει να φέρουμε τα δεδομένα στην παρακάτω μορφή (ο παρακάτω πίνακας είναι η μετατροπή του ακριβώς από πάνω):

Παρατηρήσεις :

- Η στήλη SMS έχει αντικατασταθεί από μια σειρά νέων στηλών που αναπαριστούν τις μοναδικές λέξεις από το λεξιλόγιο – το λεξιλόγιο είναι ένα σετ από τις μοναδικές λέξεις από όλες μας τις προτάσεις .
- Κάθε γραμμή περιγράφει ένα μήνυμα. Η πρώτη γραμμή έχει τις τιμές **spam, 2, 2 ,1, 1, 0, 0, 0, 0, 0** το οποίο σημαίνει :
  - Το μήνυμα είναι spam .
  - Η λέξη "**secret**" εμφανίζεται δυο φορές μέσα στο μήνυμα .
  - Η λέξη "**prize**" εμφανίζεται δυο φορές μέσα στο μήνυμα.
  - Η λέξη "**claim**" εμφανίζεται μια φορά μέσα στο μήνυμα.
  - Η λέξη "**now**" εμφανίζεται μια φορά μέσα στο μήνυμα.
  - Οι λέξεις "**coming**", "**to**", "**my**", "**party**" και "**winner**" δεν εμφανίζονται καθόλου μέσα στο μήνυμα .
- Όλες οι λέξεις μέσα στο λεξιλόγιο είναι σε πεζά γράμματα, όποτε οι λέξεις "**SECRET**" και "**secret**"λαμβάνονται υπ' όψη σαν ίδια λέξη .
- Η σειρά των λέξεων από την κανονική πρόταση έχει χαθεί .
- Τα σημεία στίξης δεν λαμβάνονται πλέον υπ' όψιν (για παράδειγμα δεν μπορούμε να κοιτάξουμε τον παραπάνω πίνακα και να καταλήξουμε στο συμπέρασμα ότι το πρώτο μήνυμα είχε τρία θαυμαστικά).

Ξεκινάμε την διαδικασία του καθαρισμού δεδομένων με την αφαίρεση των σημείων στίξης και κάνοντας όλες τις λέξεις να είναι γραμμένες σε πεζά.

```
# μετά τον καθαρισμό
training_set['SMS'] = training_set['SMS'].str.replace('\W', ' ')
training_set['SMS'] = training_set['SMS'].str.lower()
training_set.head(3)
```

Πηγή: Max Kuhn, Kjell Johnson. Applied Predictive Modeling 1st ed. 2013, Corr. 2nd printing 2018 Edition

Κομμάτι Κώδικα 5

4.1.4 Δημιουργώντας το λεξιλόγιο

Όπως προαναφέρθηκε το λεξιλόγιο αποτελεί μια λίστα με όλες τις μοναδικές λέξεις στο σετ των δεδομένων εκπαίδευσης. Στον παρακάτω κώδικα :

- Κάθε μήνυμα τις στήλης *SMS* μετατρέπεται σε μια λίστα διαχωρίζοντας τους αλφαριθμητικούς χαρακτήρες (*string*) στο σημείο του κενού χαρακτήρα (*space character*) .
- Δημιουργούμε μια κενή λίστα με το όνομα *vocabulary*.
- Χρησιμοποιώντας εμφολευμένες επαναλήψεις, κάνουμε το ίδιο για κάθε μήνυμα της στήλης *SMS* και τέλος εισάγουμε κάθε λέξη στο τέλος της λίστας *vocabulary*.
- Μετατρέπουμε την λίστα *vocabulary* σε ένα σετ με την συνάρτηση *set()* και με αυτόν τον τρόπο αφαιρούνται όλες οι διπλότυπες λέξεις από την λίστα *vocabulary*.
- Έπειτα μετατρέπουμε ξανά το σετ *vocabulary* σε λίστα χρησιμοποιώντας την συνάρτηση *list().*

```
training_set['SMS'] = training_set['SMS'].str.split()

vocabulary = []
for sms in training_set['SMS']:
   for word in sms:
      vocabulary.append(word)

vocabulary = list(set(vocabulary))
```

Πηγή: Max Kuhn, Kjell Johnson. Applied Predictive Modeling 1st ed. 2013, Corr. 2nd printing 2018 Edition

Κομμάτι Κώδικα 6.

Στο λεξιλόγιο που μόλις δημιουργήθηκε υπάρχουν 7783 μοναδικές λέξεις από το σετ των δεδομένων εκπαίδευσης.

```
len(vocabulary)
```

>>7783

4.1.5 Το τελικό σετ δεδομένων εκπαίδευσης

Στη συνέχεια χρησιμοποιείται το λεξιλόγιο που φτιάχτηκε παραπάνω έτσι ώστε όλα τα δεδομένα να πάρουν την επιθυμητή μορφή .

	Label	SMS
0	spam	SECRET PRIZE! CLAIM SECRET PRIZE NOW!!
1	ham	Coming to my secret party?
2	spam	Winner! Claim secret prize now!

	Label	secret	prize	claim	now	coming	to	my	party	winner
0	spam	2	2	1	1	0	0	0	0	0
1	ham	1	0	0	0	1	1	1	1	0
2	spam	1	1	1	1	0	0	0	0	1

Πίνακας 4 Μετασχηματισμός δεδομένων

Εν τέλη, θα χρειαστεί ένα νέο πλαίσιο δεδομένων (DataFrame). Πρώτα βέβαια πρέπει να δημιουργηθεί ένα λεξιλόγιο το οποίο θα μετατραπεί στο καινούριο πλαίσιο δεδομένων .
Για παράδειγμα, για την δημιουργία του παραπάνω πίνακα μπορεί να χρησιμοποιηθεί το παρακάτω λεξιλόγιο :

```
word_counts_per_sms = {'secret': [2,1,1],
                       'prize': [2,0,1],
                       'claim': [1,0,1],
                       'now': [1,0,1],
                       'coming': [0,1,0],
```

```
                        'to': [0,1,0],
                        'my': [0,1,0],
                        'party': [0,1,0],
                        'winner': [0,0,1]
                       }

word_counts = pd.DataFrame(word_counts_per_sms)
word_counts.head()
```

Πηγή: Max Kuhn, Kjell Johnson. Applied Predictive Modeling 1st ed. 2013, Corr. 2nd printing 2018 Edition

Κομμάτι Κώδικα 7.

Για την δημιουργία του απαιτούμενου λεξιλογίου για το σετ των δεδομένων εκπαίδευσης χρησιμοποιείται ο παρακάτω κώδικας :

- Αρχικά, δημιουργούμε το λεξιλόγιο με το όνομα, *word_counts_per_sms*, όπου κάθε στοιχείο του είναι μια μοναδική λέξη από το λεξιλόγιο και κάθε τιμή είναι μια λίστα από τα μήκη του σετ δεδομένων εκπαίδευσης, όπου κάθε στοιχείο της λίστας αυτής είναι ο αριθμός μηδέν.
  - Ο κώδικας *[0]*5* δίνει το αποτέλεσμα [**0,0,0,0,0**]. Όποτε ο κώδικας **[0]*len(training_set['SMS'])** έχεις ως έξοδο μια λίστα από τα μήκη του **training_set['SMS'])**
- Χρησιμοποιώντας τον βρόγχο επανάληψης for και την συνάρτηση *enumerate()* παίρνουμε τα στοιχεία index και *sms* από το **training_set['SMS']**

```
word_counts_per_sms = {unique_word: [0] *
len(training_set['SMS']) for unique_word in vocabulary}

for index, sms in enumerate(training_set['SMS']):
   for word in sms:
      word_counts_per_sms[word][index] += 1
```

Πηγή: Max Kuhn, Kjell Johnson. Applied Predictive Modeling 1st ed. 2013, Corr. 2nd printing 2018 Edition

Κομμάτι Κώδικα 8.

Αφού δημιουργήθηκε το λεξιλόγιο πρέπει να γίνουν οι τελευταίες μετατροπές στο σετ των δεδομένων εκπαίδευσης.

```
word_counts = pd.DataFrame(word_counts_per_sms)
word_counts.head()
```

Πηγή: Max Kuhn, Kjell Johnson. Applied Predictive Modeling 1st ed. 2013, Corr. 2nd printing 2018 Edition

Κομμάτι Κώδικα 9

4.1.6 Υπολογίζοντας τις σταθερές

Αφού τελείωσε η διαδικασία του καθαρισμού των δεδομένων εκπαιδευσης, ξεκινάει η διαδικασία του προγραμματισμού του φίλτρου ανεπιθύμητης αλληλογραφίας. Ο αλγόριθμος Naïve Bayes που θα χρησιμοποιηθεί θα πρέπει να ειναι σε θέση να απαντήσει

τις παρακάτω ερωτήσεις πιθανοτήτων για μπορέσει εν τέλη να κατηγοριοποιήσει τα καινούργια μηνύματα :

$$P(Spam|w_1, w_2, ..., w_n) \propto P(Spam) \cdot \prod_{i=1}^{n} P(w_i|Spam)$$

$$P(Ham|w_1, w_2, ..., w_n) \propto P(Ham) \cdot \prod_{i=1}^{n} P(w_i|Ham)$$

Επίσης για τον υπολογισμό του $P(w_i|Spam)$ και του $P(w_i|Ham)$ στις παραπάνω σχέσεις πρέπει να χρησιμοποιηθούν οι παρακάτω σχέσεις :

$$P(w_i|Spam) = \frac{N_{w_i|Spam} + a}{N_{Spam} + a \cdot N_{Vocabulary}}$$

Τύποι 5. Naïve Bayes Classifier

$$P(w_i|Ham) = \frac{N_{w_i|Ham} + a}{N_{Ham} + a \cdot N_{Vocabulary}}$$

Πήγη: Sunil Ray. 6 Easy Steps to Learn Naive Bayes Algorithm with codes in Python and R. https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/

Κάποιες μεταβλητές των παρακάνω τεσσάρων σχέσεων θα έχουν την ίδια τιμή για κάθε νέο μήνυμα. Θα υπολογιστεί αυτή η τιμή μια φορά αποφεύγοντας έτσι τον επανυπολογισμό της σε κάθε νέο μήνυμα. Πρώτα θα γίνει ο υπολογισμός των παρακάτω ισούται :

- P(Spam) και P(Ham)
- N_{Spam}, N_{Ham}, $N_{Vocabulary}$
- Η N_{Spam} ισούται με τον αριθμό των λέξεων σε σε όλα τα spam μηνύματα. Δεν είναι ίση με τον αριθμό των μηνυμάτων spam και επίσης δεν είναι ίση με τον συνολικό αριθμό των μοναδικών λέξεων στα spam μηνύματα
- Η N_{Ham} ισούται με τον αριθμό των λέξεων σε σε όλα τα ham μηνύματα. Δεν είναι ίση με τον αριθμό των μηνυμάτων ham και επίσης δεν είναι ίση με τον συνολικό αριθμό των μοναδικών λέξεων στα ham μηνύματα

Επίσης θα χρησιμοποιηθεί η μεταβλητή εξομάλυνσης Laplace α=1

```
# απομόνωση μηνυμάτων spam και ham
spam_messages = training_set_clean[training_set_clean['Label'] ==
'spam']
ham_messages = training_set_clean[training_set_clean['Label'] ==
'ham']

# P(Spam) and P(Ham)
p_spam = len(spam_messages) / len(training_set_clean)
p_ham = len(ham_messages) / len(training_set_clean)
```

```
# N_Spam
n_words_per_spam_message = spam_messages['SMS'].apply(len)
n_spam = n_words_per_spam_message.sum()

# N_Ham
n_words_per_ham_message = ham_messages['SMS'].apply(len)
n_ham = n_words_per_ham_message.sum()

# N_Vocabulary
n_vocabulary = len(vocabulary)

# Μεταβλητή εξομάλυνσης Laplace
alpha = 1
```

Πηγή: Max Kuhn, Kjell Johnson. Applied Predictive Modeling 1st ed. 2013, Corr. 2nd printing 2018 Edition

Κομμάτι Κώδικα 10

4.1.7 Υπολογισμός παραμέτρων

Έχοντας υπολογίσει τις σταθερές παραπάνω, η διαδικασία συνεχίζεται με τον υπολογισμό των παραμέτρων $P(w_i|Spam)$ και $P(w_i|Ham)$. Τα $P(w_i|Spam)$ και $P(w_i|Ham)$ θα διαφέρουν ανάλογα τις μεμονωμένες λέξεις. Για παράδειγμα, η λέξη P("secret"|Spam) θα έχει συγκεκριμένη τιμή πιθανότητας διαφορετική από τις λέξεις P("cousin"|Spam) και P("lovely"|Spam). Οι παράμετροι αυτοί υπολογίζονται από τις παρακάτω εξισώσεις :

$$P(w_i|Spam) = \frac{N_{w_i|Spam} + a}{N_{Spam} + a \cdot N_{Vocabulary}}$$

$$P(w_i|Ham) = \frac{N_{w_i|Ham} + a}{N_{Ham} + a \cdot N_{Vocabulary}}$$

Πηγή: Sunil Ray. Easy Steps to Learn Naive Bayes Algorithm with codes in Python and R.
https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/

```
parameters_spam = {unique_word:0 for unique_word in vocabulary}
parameters_ham = {unique_word:0 for unique_word in vocabulary}

for word in vocabulary:
   n_word_given_spam = spam_messages[word].sum() # spam_messages
already defined
   p_word_given_spam = (n_word_given_spam + alpha) / (n_spam +
alpha*n_vocabulary)
   parameters_spam[word] = p_word_given_spam

   n_word_given_ham = ham_messages[word].sum() # ham_messages already
defined
   p_word_given_ham = (n_word_given_ham + alpha) / (n_ham +
alpha*n_vocabulary)
   parameters_ham[word] = p_word_given_ham
```

Πηγή: Max Kuhn, Kjell Johnson. Applied Predictive Modeling 1st ed. 2013, Corr. 2nd printing 2018 Edition

Κομμάτι Κώδικα 11

4.1.8 Κατηγοριοποίηση Καινούργιου μηνύματος

Αφού τελείωσαν οι υπολογισμοί των σταθερών και των παραμέτρων μπορούμε να δημιουργήσουμε το φίλτρο ανεπιθύμητης αλληλογραφίας. Το φίλτρο αυτό είναι κατανοητό σαν μια συνάρτηση η οποία:

- Δέχεται σαν είσοδο ένα νέο μήνυμα (w_1, w_2, ..., w_n).
- Υπολογίζει τα P(Spam|w_1, w_2, ..., w_n) και τα P(Ham|w_1, w_2, ..., w_n).
- Συγκρίνει τις τιμές των P(Spam|w_1, w_2, ..., w_n) και P(Ham|w_1, w_2, ..., w_n) και :
  - Αν P(Ham|w_1, w_2, ..., w_n) > P(Spam|w_1, w_2, ..., w_n), τότε το μήνυμα κατηγοριοποιείται σαν ham.
  - Αν P(Ham|w_1, w_2, ..., w_n) < P(Spam|w_1, w_2, ..., w_n), τότε το μήνυμα κατηγοριοποιείται σαν spam.
  - Αν P(Ham|w_1, w_2, ..., w_n) = P(Spam|w_1, w_2, ..., w_n), τότε ο αλγόριθμος χρήζει ανθρώπινη βοήθεια.

Κάποια καινούργια μηνύματα θα εμπεριέχουν λέξεις οι οποίες δεν θα ανήκουν στο λεξιλόγιο. Οι λέξεις αυτές θα αγνοούνται κατά τη διαδικασία υπολογισμού των πιθανοτήτων.

```
import re

def classify(message):
   '''
   message: a string
   '''

   message = re.sub('\W', ' ', message)
   message = message.lower().split()

   p_spam_given_message = p_spam
   p_ham_given_message = p_ham

   for word in message:
      if word in parameters_spam:
         p_spam_given_message *= parameters_spam[word]

      if word in parameters_ham:
         p_ham_given_message *= parameters_ham[word]

   print('P(Spam|message):', p_spam_given_message)
   print('P(Ham|message):', p_ham_given_message)

   if p_ham_given_message > p_spam_given_message:
      print('Label: Ham')
```

```
   elif p_ham_given_message < p_spam_given_message:
      print('Label: Spam')
   else:
      print('Equal proabilities, need help from a human!')
```

Πηγή: Max Kuhn, Kjell Johnson. Applied Predictive Modeling 1st ed. 2013, Corr. 2nd printing 2018 Edition

Κομμάτι Κώδικα 12

4.1.9 Υπολογίζοντας το ποσοστό ευστοχία του αλγορίθμου

Στην προκειμένη περίπτωση θα χρησιμοποιηθεί μια συνάρτηση που επιστρέφει τις τιμές την κατηγοριοποίησης αντί να τις εκτυπώνει.

```
def classify_test_set(message):
   '''
   message: a string
   '''

   message = re.sub('\W', ' ', message)
   message = message.lower().split()

   p_spam_given_message = p_spam
   p_ham_given_message = p_ham

   for word in message:
      if word in parameters_spam:
         p_spam_given_message *= parameters_spam[word]

      if word in parameters_ham:
         p_ham_given_message *= parameters_ham[word]

   if p_ham_given_message > p_spam_given_message:
      return 'ham'
   elif p_spam_given_message > p_ham_given_message:
      return 'spam'
   else:
      return 'needs human classification'
```

Πηγή: Max Kuhn, Kjell Johnson. Applied Predictive Modeling 1st ed. 2013, Corr. 2nd printing 2018 Edition

Κομμάτι Κώδικα 13

Έχοντας την παραπάνω συνάρτηση που επιστρέφει τις τιμές την κατηγοριοποίησης, μας δίνεται η δυνατότητα να δημιουργήσουμε μια νέα στήλη στο σετ των δεδομένων δοκιμών.

```
test_set['predicted']=test_set['SMS'].apply(classify_test_set)
test_set.head()
```

Πηγή: Max Kuhn, Kjell Johnson. Applied Predictive Modeling 1st ed. 2013, Corr. 2nd printing 2018 Edition

Κομμάτι Κώδικα 14

Τέλος θα συγκρίνουμε τις τιμές που προβλέψαμε με τις αντίστοιχες πραγματικές για να υπολογίζουμε ποσό καλό είναι το παραπάνω φίλτρο ανεπιθύμητης αλληλογραφίας. Για τον υπολογισμό αυτό θα χρησιμοποιηθεί ο παρακάτω τύπος για την Ευστοχία Αλγορίθμου:

$$Ευστοχία\ (accuracy) = \frac{Αριθμός\ των\ σωστά\ κατηγοριοποιημένων\ μηνυμάτων}{Συνολικός\ αριθμός\ κατηγοριοποιημένων\ μηνυματων}$$

```
correct = 0
total = test_set.shape[0]

for row in test_set.iterrows():
   row = row[1]
   if row['Label'] == row['predicted']:
      correct += 1

print('Correct:', correct)
print('Incorrect:', total - correct)
print('Accuracy:', correct/total)
```

Πηγή: Max Kuhn, Kjell Johnson. Applied Predictive Modeling 1st ed. 2013, Corr. 2nd printing 2018 Edition

Κομμάτι Κώδικα 15

4.2 Φίλτρο ανεπιθύμητης αλληλογραφίας με τη βοήθεια της μηχανικής εκμάθησης

Αρχικά γίνεται ο ακριβής προσδιορισμός της θέσης των καταλόγων με τα spam και τα ham e-mail και στη συνέχεια δημιουργούμε τον πίνακα *labeled_file_direcroties* ο οποίος αποθηκεύει τις παραπάνω θέσεις ακολουθούμενες από τις τιμές 0 και 1 για τα spam και τα ham e-mail αντίστοιχα.

```
import os
spam_emails_path = os.path.join("spamassassin-public-corpus",
"spam")
ham_emails_path = os.path.join("spamassassin-public-corpus", "ham")
labeled_file_directories = [(spam_emails_path, 0),
(ham_emails_path, 1)]
```

Έπειτα δημιουργούμε δύο πίνακες, τον *email_corpus* ο οποίος αποτελείται από το περιεχόμενο όλων των e-mail και τον πίνακα labels του οποίου τα στοιχεία αποτελούνται από τις τιμές μηδέν (0) ή ένα (1) και αντιπροσωπεύουν το αντίστοιχο e-mail του πίνακα *email_corpus.* Αν η τιμή είναι μηδέν (0) τότε το e-mail είναι spam και αν η τιμή είναι ένα (1) το e-mail είναι *ham* όπως προαναφέρθηκε.

```
email_corpus = []
labels = []
for class_files, label in labeled_file_directories:
files = os.listdir(class_files)
for file in files:
```

```
file_path = os.path.join(class_files, file)
try:
with open(file_path, "r") as currentFile:
email_content = currentFile.read().replace("\n",
"")
email_content = str(email_content)
email_corpus.append(email_content)
labels.append(label)
except:
pass
```

Στη συνέχεια γίνεται ο διαχωρισμός των δεδομένων σε δεδομένα εκπαίδευσης και δεδομένα δοκιμών με ποσοστό 80% των δεδομένων σε δεδομένα εκπαίδευσης και 20% σε δεδομένα δοκιμών. Το αποτέλεσμα είναι η δημιουργία των παρακάτω πινάκων:

- **X_train**: αποτελείται από τα e-mail που ανήκουν στο σετ των δεδομένων εκπαίδευσης.
- **y_train** = αποτελείται από τις τιμές κατηγοριοποίησης (μηδέν ή ένα) των e-mail του σετ των δεδομένων εκπαίδευσης.
- **X_test** = αποτελείται από τα e-mail που ανήκουν στο σετ των δεδομένων δοκιμών.
- **Y_test** = αποτελείται από τις τιμές κατηγοριοποίησης (μηδέν ή ένα) των e-mail του σετ των δεδομένων δοκιμών.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
email_corpus, labels, test_size=0.2, random_state=11
)
```

Στη συνέχεια εκπαιδεύουμε τον αλγόριθμο με τα δεδομένα εκπαίδευσης. Αρχικά ο αλγόριθμος θα μετατρέψει τα e-mail σε αριθμούς ώστε να μπορεί να τα επεξεργαστεί. Έπειτα θα βρει τη συχνότητα εμφάνισης κάθε λέξης σε όλα τα e-mail και στη συνέχεια με τη βοήθεια του δέντρου απόφασης θα βγάλει τα συμπεράσματα για την κατηγοριοποίηση.

```
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import HashingVectorizer,
TfidfTransformer
from sklearn import tree
nlp_followed_by_dt = Pipeline(
[
("vect", HashingVectorizer(input="content", ngram_range=(1,
3))),
("tfidf", TfidfTransformer(use_idf=True,)),
("dt",
tree.DecisionTreeClassifier(class_weight="balanced")),
]
)
nlp_followed_by_dt.fit(X_train, y_train)
```

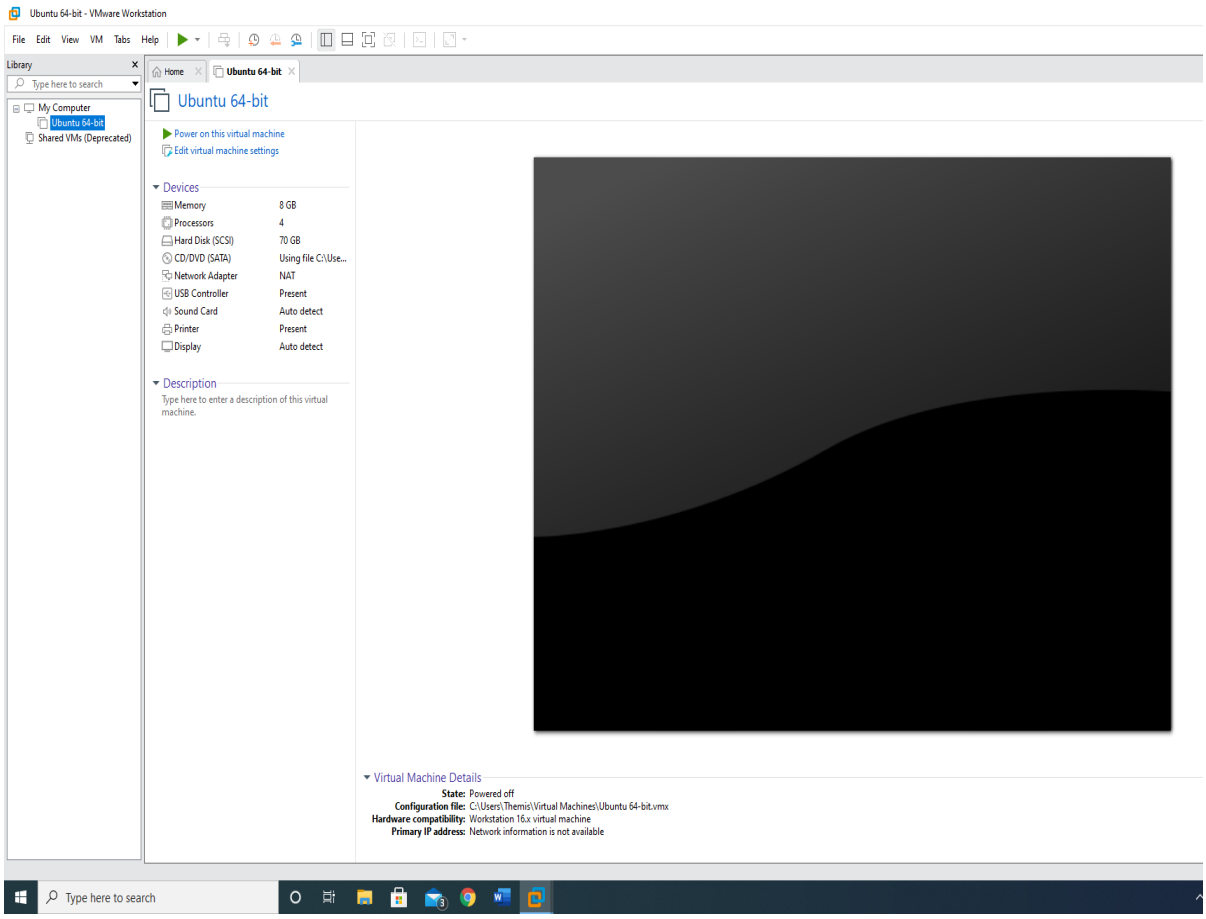
4.3 Virtual Machines

Ένα *virtual machine* χρησιμοποιείται για να προσομοιάσει ένα ιδανικό περιβάλλον αντίγραφο ή όχι του κανονικού περιβάλλοντος επεξεργασίας με σκοπό την μελέτη της συμπεριφοράς των κακόβουλων αρχείων. Η ικανότητα που διαθέτουν τα *virtual machines* να προσομοιάζουν διαφορετικές περιστάσεις λειτουργικών συστημάτων στην ίδια μηχανή και να παρέχουν ένα κανονικό περιβάλλον ανάλυσης το οποίο στην πραγματικότητα είναι πολύ πιο ασφαλές είναι ένα πολύ σημαντικό και απαραίτητο όργανο για την ανάλυση συμπεριφορών κακόβουλου λογισμικού. Τα πλεονεκτήματα που παρέχει η χρήση virtual machine είναι αρκετά, μερικά από τα οποία αναφέρονται παρακάτω:

- Υποστηρίζουν αναλύσεις μεταξύ πολλών λειτουργικών συστημάτων: Η ανάλυση κακόβουλου λογισμικού γίνεται μεταξύ μιας μεγάλης ποικιλίας συστημάτων ώστε να εκπαιδευτεί το μοντέλο στους διαφορετικούς τρόπους αντίδρασης του δείγματος σε διάφορα λειτουργικά συστήματα όπως *Windows*, *MacOS* και *Linux*. Με τη χρήση *Virtual Machine* τα περιβάλλοντα αυτά δημιουργούνται εικονικά χωρίς να χρειαστεί η αγορά και η εγκατάσταση πολλαπλών και μεγάλων συστημάτων από την αρχή.

- Παρέχουν ασφάλεια στο σύστημα : Κάνοντας τις εκπαιδεύσεις και τις δοκιμές του μοντέλου μέσα σε ένα εικονικό περιβάλλον, το πραγματικό περιβάλλον του υπολογιστή παραμένει ανέπαφο με τα κακόβουλα αρχεία και σε προέκταση ασφαλέστερο.

Για την στατική ανάλυση κακόβουλων εκτελέσιμων αρχείων που θα ακολουθήσει η διαδικασία εξαγωγής χρήσιμων στοιχείων δεν απαιτεί την εκτέλεση των αρχείων. Κατά την επεξεργασία των κακόβουλων εκτελέσιμων αρχείων των *windows* βέβαια το *anti-virus* θα τα θέσει σε καραντίνα και η εξαγωγή στοιχείων από αυτά δεν θα είναι πλέον δυνατή. Οπότε θα δημιουργήσουμε ένα *virtual machine* σε λειτουργικό σύστημα τύπου *linux* όπου τα εκτελέσιμα αυτά αρχεία θα είναι απολύτως αβλαβή και κατά την επεξεργασία τους δεν δημιουργούν πλέον προβλήματα στο πραγματικό μας σύστημα.



Εικόνα 8 Linux Virtual Machine σε Windows 10

4.4 Static Malware Detector

4.4.1 Στατική ανάλυση κακόβουλου λογισμικού

Στη στατική ανάλυση, εξετάζουμε ένα δείγμα χωρίς να το εκτελέσουμε. Το ποσό των πληροφοριών που μπορεί να ληφθεί με αυτόν τον τρόπο είναι μεγάλο, που κυμαίνεται από κάτι τόσο απλό όσο το όνομα του αρχείου (*PE*) σε κάτι πολύ πιο περίπλοκο, όπως εξειδικευμένες υπογραφές *YARA*. Πριν ξεκινήσουμε την ανάπτυξη του προγράμματος αρχικά θα αναφέρουμε τα βασικά χαρακτηριστικά των εκτελέσιμων αρχείων και τις διεργασίες εξαγωγής τους. Χαρακτηριστικά τα οποία στη συνέχεια θα χρησιμοποιηθούν στην εκπαίδευση του αλγορίθμου μας [5].

4.4.2 Τα αρχεία PE

Τα αρχεία *PE* (*PORTABLE EXECUTABLE*) είναι ένας κοινός τύπος αρχείου των *Windows*. Τα αρχεία PE περιλαμβάνουν τα .exe, .dll και τα .sys αρχεία. Όλα τα αρχεία *PE* διακρίνονται με μια κεφαλίδα *PE*, η οποία είναι ενότητα κεφαλίδας του κώδικα που δίνει οδηγίες στα

Windows σχετικά με τον τρόπο ανάλυσης του επόμενου κώδικα. Τα πεδία από την κεφαλίδα *PE* χρησιμοποιούνται συχνά ως χαρακτηριστικά για τον εντοπισμό κακόβουλου λογισμικού. Για να εξαχθεί πιο εύκολα το πλήθος των τιμών της κεφαλίδας *PE*, θα χρησιμοποιήσουμε το *pefile Python module*.

4.4.3 Εξαγωγή N-Grams

Στην τυπική ποσοτική ανάλυση του κειμένου, τα *N-Grams* είναι αλληλουχίες *N* ενδείξεων (για παράδειγμα, λέξεις ή χαρακτήρες). Για παράδειγμα, δεδομένου του κειμένου *Η γρήγορη καφέ αλεπού πήδηξε πάνω από τον τεμπέλη σκύλο*, αν οι ενδείξεις μας είναι λέξεις, τότε το 1-Gram είναι *η, γρήγορη, καφέ, αλεπού, πήδηξε, πάνω, από, τον, τεμπέλη* και *σκύλο*. Το *2-Gram* είναι *η γρήγορη, γρήγορη καφέ, καφέ αλεπού* και ούτω καθεξής. Το *3-Gram* είναι *η γρήγορη καφέ, γρήγορη καφέ αλεπού, καφέ αλεπού πήδηξε,* και ούτω καθεξής. Τα N-gram μας επιτρέπουν να μοντελοποιήσουμε τις τοπικές στατιστικές ιδιότητες του κορμού μας. Με απώτερο στόχο την αξιοποίηση των μετρήσεων *N-Gram* για να μας βοηθήσουν να προβλέψουμε εάν ένα δείγμα είναι κακόβουλο ή καλοήθη.

4.4.4 Επιλογή των καλύτερων N-Grams

Ο αριθμός των διαφορετικών *N-Grams* αυξάνεται εκθετικά στο *N*. Ακόμη και για ένα σταθερό μικροσκοπικό *N*, όπως *N=3*, υπάρχουν **256x256x256=16,777,216** πιθανά N-Grams. Αυτό σημαίνει ότι ο αριθμός των χαρακτηριστικών των *N-Grams* είναι πρακτικά πολύ μεγάλος. Κατά συνέπεια, πρέπει να επιλέξουμε ένα μικρότερο υποσύνολο των *N-Grams* το οποίο θα έχει μεγαλύτερη αξία για τον ταξινομητή μας.

1. Ξεκινάμε απαριθμώντας τα δείγματά μας και εκχωρούμε τις ετικέτες τους

```
import os
            from os import listdir

            directories_with_labels = [("Benign PE Samples", 0),
("Malicious PE
            Samples", 1)]
list_of_samples = []
     labels = []
     for dataset_path, label in directories_with_labels:

         samples = [f for f in listdir(dataset_path)]
         for sample in samples:
             file_path = os.path.join(dataset_path, sample)
             list_of_samples.append(file_path)
             labels.append(label)
```

Πηγή: Abou-Assaleh, T., Cercone, N., Kešelj, V., Sweidan, R.: N-gram-based detection of new mali-cious code. In: Proceedings of the 28th Annual International Computer Software and Applica-tions Conference, 2004. COMPSAC 2004, vol. 2, IEEE (2004

Κομμάτι Κώδικα 16

2. Κάνουμε διαχωρισμό δεδομένων σε δεδομένα εκπαίδευσης και δεδομένα δοκιμών.

```
from sklearn.model_selection import train_test_split
samples_train, samples_test, labels_train, labels_test =
train_test_split(
list_of_samples, labels, test_size=0.3, stratify=labels,
random_state=11
)
```

Πηγή: Abou-Assaleh, T., Cercone, N., Kešelj, V., Sweidan, R.: N-gram-based detection of new mali-cious code. In: Proceedings of the 28th Annual International Computer Software and Applica-tions Conference, 2004. COMPSAC 2004, vol. 2,. IEEE (2004)

Κομμάτι Κώδικα 17

3. Καθορίζουμε χρήσιμες συναρτήσεις οι οποίες οι οποίες θα μας βοηθήσουν στην εξαγωγή στοιχείων από τα δεδομένα.

```
import collection
from nltk import ngrams
import numpy as np
import pefile
def read_file(file_path):
"""Reads in the binary sequence of a binary file."""
with open(file_path, "rb") as binary_file:
data = binary_file.read()
return data
def byte_sequence_to_Ngrams(byte_sequence, N):
"""Creates a list of N-grams from a byte sequence."""
Ngrams = ngrams(byte_sequence, N)
return list(Ngrams)
def binary_file_to_Ngram_counts(file, N):
"""Takes a binary file and outputs the N-grams counts of its
binary sequence."""
filebyte_sequence = read_file(file)
file_Ngrams = byte_sequence_to_Ngrams(filebyte_sequence, N)
return collections.Counter(file_Ngrams)
```

Πηγή: Abou-Assaleh, T., Cercone, N., Kešelj, V., Sweidan, R.: N-gram-based detection of new mali-cious code. In: Proceedings of the 28th Annual International Computer Software and Applica-tions Conference, 2004. COMPSAC 2004, vol. 2, IEEE (2004)

Κομμάτι Κώδικα 18

```
def get_NGram_features_from_sample(sample,
K1_most_frequent_Ngrams_list):
"""Takes a sample and produces a feature vector.
The features are the counts of the K1 N-grams we've selected.
"""
K1 = len(K1_most_frequent_Ngrams_list)
feature_vector = K1 * [0]
file_Ngrams = binary_file_to_Ngram_counts(sample, N)
for i in range(K1):
feature_vector[i] =
file_Ngrams[K1_most_frequent_Ngrams_list[i]]
return feature_vector
```

Πηγή: Abou-Assaleh, T., Cercone, N., Kešelj, V., Sweidan, R.: N-gram-based detection of new mali-cious code. In: Proceedings of the 28th Annual International Computer Software and Applica-tions Conference, 2004. COMPSAC 2004, vol. 2,. IEEE (2004)

Κομμάτι Κώδικα 19

```
def preprocess_imports(list_of_DLLs):
"""Normalize the naming of the imports of a PE file."""
temp = [x.decode().split(".")[0].lower() for x in list_of_DLLs]
return " ".join(temp)
def get_imports(pe):
"""Get a list of the imports of a PE file."""
list_of_imports = []
for entry in pe.DIRECTORY_ENTRY_IMPORT:
list_of_imports.append(entry.dll)
return preprocess_imports(list_of_imports)
def get_section_names(pe):
"""Gets a list of section names from a PE file."""
list_of_section_names = []
for sec in pe.sections:
normalized_name = sec.Name.decode().replace("\x00",
"").lower()
list_of_section_names.append(normalized_name)
return "".join(list_of_section_names)
```

Πηγή: Abou-Assaleh, T., Cercone, N., Kešelj, V., Sweidan, R.: N-gram-based detection of new mali-cious code. In: Proceedings of the 28th Annual International Computer Software and Applica-tions Conference, 2004. COMPSAC 2004, vol. 2, IEEE (2004)

Κομμάτι Κώδικα 20

4. Στη συνέχεια επιλέγουμε τα 100 πιο συχνά N-Grams για να χρησιμοποιήσουμε.

```
N = 2
Ngram_counts_all = collections.Counter([])
for sample in samples_train:
Ngram_counts_all += binary_file_to_Ngram_counts(sample, N)
K1 = 100
K1_most_frequent_Ngrams = Ngram_counts_all.most_common(K1)
K1_most_frequent_Ngrams_list = [x[0] for x in
K1_most_frequent_Ngrams]
```

Πηγή: Abou-Assaleh, T., Cercone, N., Kešelj, V., Sweidan, R.: N-gram-based detection of new mali-cious code. In: Proceedings of the 28th Annual International Computer Software and Applica-tions Conference, 2004. COMPSAC 2004, vol. 2, IEEE (2004)

Κομμάτι Κώδικα 21

5. Έπειτα υπολογίζουμε τις μετρήσεις των N-gram, τα ονόματα των ενοτήτων και τον αριθμό τους που ανήκουν στην PE Header του κάθε αρχείου των δεδομένων εκπαίδευσης, ενώ τα δεδομένα των οποίων η PE Header δεν μπορεί να αναλυθεί προσπερνούνται.

```
imports_corpus_train = []
num_sections_train = []
section_names_train = []
Ngram_features_list_train = []
y_train = []
for i in range(len(samples_train)):
sample = samples_train[i]
```

```
try:
NGram_features = get_NGram_features_from_sample(
sample, K1_most_frequent_Ngrams_list
)
pe = pefile.PE(sample)
imports = get_imports(pe)
n_sections = len(pe.sections)
sec_names = get_section_names(pe)
imports_corpus_train.append(imports)
num_sections_train.append(n_sections)
section_names_train.append(sec_names)
Ngram_features_list_train.append(NGram_features)
y_train.append(labels_train[i])
except Exception as e:
print(sample + ":")
print(e)
```

Πηγή: Abou-Assaleh, T., Cercone, N., Kešelj, V., Sweidan, R.: N-gram-based detection of new mali-cious code. In: Proceedings of the 28th Annual International Computer Software and Applica-tions Conference, 2004. COMPSAC 2004, vol. 2, IEEE (2004)

Κομμάτι Κώδικα 22

6. Χρησιμοποιούμε έναν hashing vectorizer σε συνδυασμό με έναν Tfidf transformer (term-frequency times inverse document frequency) οι οποίοι θα μετατρέψουν τις εισόδους και τα ονόματα των ενοτήτων που είναι χαρακτηριστικά σε μορφή κειμένου, σε αριθμητική μορφή.

```
from sklearn.feature_extraction.text import HashingVectorizer,
TfidfTransformer
from sklearn.pipeline import Pipeline
imports_featurizer = Pipeline(
[
("vect", HashingVectorizer(input="content", ngram_range=(1,
2))),
("tfidf", TfidfTransformer(use_idf=True,)),
]
)
section_names_featurizer = Pipeline(
[
("vect", HashingVectorizer(input="content", ngram_range=(1,
2))),
("tfidf", TfidfTransformer(use_idf=True,)),
]
)
imports_corpus_train_transformed =
imports_featurizer.fit_transform(
imports_corpus_train
)
section_names_train_transformed =
section_names_featurizer.fit_transform(
section_names_train
```

```
)
```

Πηγή: Abou-Assaleh, T., Cercone, N., Kešelj, V., Sweidan, R.: N-gram-based detection of new mali-cious code. In: Proceedings of the 28th Annual International Computer Software and Applica-tions Conference, 2004. COMPSAC 2004, vol. 2, IEEE (2004)

Κομμάτι Κώδικα 23

7. Συνδυάζουμε όλα τα χαρακτηριστικά και τα τοποθετούμε σε έναν πίνακα.

```
from scipy.sparse import hstack, csr_matrix
X_train = hstack(
[
Ngram_features_list_train,
imports_corpus_train_transformed,
section_names_train_transformed,
csr_matrix(num_sections_train).transpose(),
])
```

Πηγή: Abou-Assaleh, T., Cercone, N., Kešelj, V., Sweidan, R.: N-gram-based detection of new mali-cious code. In: Proceedings of the 28th Annual International Computer Software and Applica-tions Conference, 2004. COMPSAC 2004, vol. 2, IEEE (2004)

Κομμάτι Κώδικα 24

8. Στη συνέχεια εκπαιδεύουμε έναν ταξινομητή Random Forest με το σετ των δεδομένων εκπαίδευσης και εκτυπώνουμε το αποτέλεσμά του.

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=100)
clf = clf.fit(X_train, y_train)
```

Κομμάτι Κώδικα 25

9. Έπειτα επαναλαμβάνουμε την παραπάνω διαδικασία υπολογισμού και εξαγωγής των χαρακτηριστικών αλλά αυτή τη φορά για το σετ των δεδομένων δοκιμών.

```
imports_corpus_test = []
num_sections_test = []
section_names_test = []
Ngram_features_list_test = []
y_test = []
for i in range(len(samples_test)):
file = samples_test[i]
try:
NGram_features = get_NGram_features_from_sample(
sample, K1_most_frequent_Ngrams_list
)
pe = pefile.PE(file)
imports = get_imports(pe)
n_sections = len(pe.sections)
sec_names = get_section_names(pe)
imports_corpus_test.append(imports)
num_sections_test.append(n_sections)
section_names_test.append(sec_names)
Ngram_features_list_test.append(NGram_features)
```

```
y_test.append(labels_test[i])
except Exception as e:
print(sample + ":")
print(e)
```

Πηγή: Bruschi, D., Martignoni, L., Monga, M.: Detecting self-mutating malware using control-flowgraph matching. In: Detection of Intrusions and Malware & Vulnerability Assessment. Springer (2006)

Κομμάτι Κώδικα 26

10. Τέλος, τροποποιούμαι τα χαρακτηριστικά κειμένου των δεδομένων δοκιμών σε αριθμούς με τη βοήθεια του hashing vectorizer και του Tfidf transformer (term-frequency times inverse document frequency) και δοκιμάζουμε τον ταξινομητή, που εκπαιδεύσαμε προηγουμένως, πάνω στο σετ των δεδομένων δοκιμών.

```
imports_corpus_test_transformed =
imports_featurizer.transform(imports_corpus_test)
section_names_test_transformed =
section_names_featurizer.transform(section_names_test)
X_test = hstack(
[
Ngram_features_list_test,
imports_corpus_test_transformed,
section_names_test_transformed,
csr_matrix(num_sections_test).transpose(),
]
)
print("Accuracy :", clf.score(X_test, y_test))
```

Πηγή: . Bruschi, D., Martignoni, L., Monga, M.: Detecting self-mutating malware using control-flowgraph matching. In: Detection of Intrusions and Malware & Vulnerability Assessment. Springer (2006)

Κομμάτι Κώδικα 27

Αρχικά δηλώνουμε την ακριβής τοποθεσία των δειγμάτων μας και εκχωρούμε την τιμή μηδέν για κάθε αβλαβή αρχείο και την τιμή ένα για κάθε κακόβουλο. Φτιάχνουμε δύο καινούριες λίστες την *list_of_samples* η οποία εμπεριέχει όλα τα εκτελέσιμα αρχεία τα οποία θα χρησιμοποιήσουμε και την λίστα *labels* η οποία εμπεριέχει τις τιμές μηδέν ή ένα προσδιορίζοντας το είδος του αντίστοιχου αρχείου στην λίστα *list_of _samples* (βήμα 1). Επειδή τα δεδομένα είναι ασταθή, πραγματοποιούμε στρωματοποιημένο διαχωρισμό δεδομένων εκπαίδευσης και δοκιμών. Σε ένα στρωματοποιημένο διαχωρισμό δεδομένων, το ποσοστό κάθε κλάσης, στην προκειμένη περίπτωση αβλαβές ή κακόβουλο, είναι το ίδιο στο σετ δεδομένων εκπαίδευσης, στο σετ δοκιμών και στο αρχικό σετ. Αυτό διασφαλίζει ότι δεν υπάρχει πιθανότητα το σετ εκπαίδευσης να αποτελείται από μία μόνο κλάση λόγω τυχαίου γεγονότος. Το 70% των δεδομένων θα χρησιμοποιηθεί για την εκπαίδευση του ταξινομητή που θα δημιουργήσουμε και το 30% θα χρησιμοποιηθεί για τις δοκιμές. Μετά τον διαχωρισμό των δεδομένων (βήμα 2) έχουμε τους τέσσερις παρακάτω πίνακες :

- **Samples_train** : αποτελείται από τα εκτελέσιμα αρχεία που θα χρησιμοποιηθούν για την εκπαίδευση του ταξινομητή.
- **Samples_test** : αποτελείται από τα εκτελέσιμα αρχεία που θα χρησιμοποιηθούν για τις δοκιμές του ταξινομητή.

- **Labels_train** : αποτελείται από τις τιμές μηδέν ή ένα των αντίστοιχων εκτελέσιμων αρχείων που ανήκουν στο σετ των δεδομένων εκπαίδευσης.
- **Labels_test** : αποτελείται από τις τιμές μηδέν ή ένα των αντίστοιχων εκτελέσιμων αρχείων που ανήκουν στο σετ των δεδομένων δοκιμών.

Στη συνέχεια δηλώνουμε τις παρακάτω χρήσιμες συναρτήσεις οι οποίες θα χρησιμοποιηθούν για την εξαγωγή των στοιχείων που επιθυμούμε από τα δεδομένα μας:

- **Read_file(file_path)** : προσδιορίζουμε την τοποθεσία του αρχείου που επιθυμούμε να αναλυθεί και στη συνέχεια η συνάρτηση διαβάζει όλα τα byte του.
- **Byte_sequence_to_Ngram_counts** (**byte_sequence, N**): προσδιορίζουμε την ακολουθία byte ενός αρχείου και τον αριθμό των N-gram που θα αναζητήσει και θα επιστρέψει η συνάρτηση σε λίστα.
- **Binary_file_to_Ngram_counts(file, N)** : προσδιορίζουμε το αρχείο που επιθυμούμε σε δυαδική μορφή και η συνάρτηση θα επιστρέψει τις φορές που εμφανίζεται το κάθε N-gram στο συγκεκριμένο αρχείο.
- **Get_Ngram_features_from_sample(sample, K1_most_frequent_Ngrams_list):** η συγκεκριμένη συνάρτηση παίρνει ως είσοδο ένα αρχείο και έχει ως έξοδο τον αριθμό των εμφανίσεων των K1 ο οποίος είναι ο αριθμός των πιο συχνών N-gram.

Οι παραπάνω συναρτήσεις με λίγα λόγια χρησιμοποιούνται για την εξαγωγή των πιο συχνών N-gram από κάποιο αρχείο του ενδιαφέροντός μας καθώς αυτά μας παρέχουν τις πιο χρήσιμες πληροφορίες για την ανάλυση του αρχείου. Στη συνέχεια ακολουθούν συναρτήσεις οι οποίες εξάγουν χρήσιμα στοιχεία από την *PE header* του κάθε αρχείου:

- **Preprocess_imports(list_ofDLLs):** η συγκεκριμένη συνάρτηση συλλέγει τα ονόματα από τα τμήματα ενός εκτελέσιμου αρχείου με σκοπό την τυποποίησή τους.
- **Get_imports(pe)** : η συνάρτηση αυτή συλλέγει τις εισόδους από ένα εκτελέσιμο αρχείο και δημιουργεί μια λίστα με αυτές.
- **Get_section_names(pe)** : η συγκεκριμένη συνάρτηση συλλέγει τα ονόματα από τα τμήματα των εκτελέσιμων αρχείων και τα προετοιμάζει ώστε να είναι ευανάγνωστα και ομαλοποιημένα.

Στη συνέχεια, χρησιμοποιούμε τις παραπάνω τεχνικές εξαγωγής χρήσιμων στοιχείων για να υπολογίζουμε τα 100 πιο συχνά *2-gram* (βήμα 4) και έπειτα επαναλαμβάνουμε την προηγούμενη διαδικασία σε όλα τα αρχεία ώστε να εξάγουμε όλα τα χαρακτηριστικά που μας ενδιαφέρουν (βήμα 5). Παίρνουμε τα χαρακτηριστικά της *PE header* τα οποία αποκτήσαμε προηγουμένως, όπως τα ονόματα των τμημάτων και τις εισόδους και τα περνάμε από έναν *hashing vectorizer* σε συνδυασμό με έναν *Tfidf transformer (term-frequency times inverse document frequency)* οι οποίοι θα τα μετατρέψουν από μορφή κειμένου, σε αριθμητική μορφή, το οποίο αποτελεί μια βασική προσέγγιση *NLP (Natural Language Processing)* (βήμα 6). Έχοντας εξάγει όλα τα παραπάνω διαφορετικά χαρακτηριστικά, χρησιμοποιούμε το *scipy hstack* για να τα συγκεντρώσουμε όλα μαζί σε έναν μεγάλο πίνακα (βήμα 7). Συνεχίζουμε εκπαιδεύοντας τον ταξινομητή *Random Forest* με

το σετ των δεδομένων εκπαίδευσης και εκτυπώνουμε την ευστοχία του. Έπειτα επαναλαμβάνουμε την παραπάνω διαδικασία εξαγωγής χρήσιμων χαρακτηριστικών από το σετ δεομένων δοκιμών (βήμα 9) και τέλος στο βήμα 10 δοκιμάζουμε τον εκπαιδευμένο ταξινομητή και εκτυπώνουμε την ευστοχία του.

4.1 Deep learning for malicious PE detection

Μια από τις νέες μεθόδους ανάπτυξης στατικής αναγνώρισης κακόβουλου λογισμικού είναι η χρήση μεθόδων deep learning. Το *deep learning* είναι υποσύνολο του *machine learning* στο οποίο χρησιμοποιούνται αλγόριθμοι εμπνευσμένοι από τη δομή και τη λειτουργία του νευρωνικού δικτύου του ανθρώπινου εγκεφάλου. Στην ουσία με το *deep learning* συνεχίζουμε να μιλάμε για αλγορίθμους οι οποίοι μαθαίνουν από ένα σύνολο δεδομένων όπως στο machine learning, αλλά με τη διαφορά ότι πλέον οι αλγόριθμοι είναι μοντέλα τα οποία βασίζονται στη δομή και τη λειτουργία των νευρωνικών δικτύων του εγκεφάλου. Η διαδικασία της εκμάθησης γίνεται είτε με την μορφή της επιτήρησης (*supervised learning*) είτε χωρίς αυτήν (*unsupervised learning*). Στην περίπτωση της μάθησης με επιτήρηση το *deep learning* μοντέλο μαθαίνει και βγάζει συμπεράσματα από δεδομένα τα οποία έχουν προηγουμένως κατηγοριοποιηθεί ενώ αντίθετα στην περίπτωση της εκμάθησης χωρίς επίβλεψη το μοντέλο μαθαίνει και βγάζει συμπεράσματα από δεδομένα που δεν έχουν κατηγοριοποιηθεί [68].

Παρακάτω θα αναπτυχθεί ένα μοντέλο *deep learning* το οποίο θα χρησιμοποιεί πρακτικές στατικής ανάλυσης κακόβουλου λογισμικού αλλά στη συγκεκριμένη περίπτωση θα αγνοήσουμε τα χαρακτηριστικά της αρχιτεκτονικής των εκτελέσιμων αρχείων ή οποιονδήποτε άλλων χαρακτηριστικών τα οποία θα μας έκαναν γνωστή την ύπαρξη κακόβουλων εκτελέσιμων αρχείων. Αυτό που θα κάνουμε είναι να μετατρέψουμε τα εκτελέσιμα αρχεία σε απλά bytes, να τα φορτώσουμε στο τεχνητό νευρωνικό μας δίκτυο και στη συνέχεια να εκπαιδεύσουμε το μοντέλο μας. Η συγκεκριμένη αρχιτεκτονική είναι γνωστή ως *MalConv [69].*

1. Ξεκινάμε κάνοντας *import* χρήσιμες βιβλιοθήκες που θα μας βοηθήσουν στην επεξεργασία των δεδομένων.

```
import numpy as np
from tqdm import tqdm
```

2. Ορίζουμε μια συνάρτηση η οποία εκχωρεί τα byte σε έναν πίνακα.

```
def embed_bytes(byte):
    binary_string = "{0:08b}".format(byte)
    vec = np.zeros(8)
    for i in range(8):
        if binary_string[i] == "1":
            vec[i] = float(1) / 16
```

```
    else:
        vec[i] = -float(1) / 16
    return vec
```

Πηγή: Tsukerman, E. (2019). Machine Learning for Cybersecurity Cookbook. Packt Publishing.

Κομμάτι Κώδικα 28

3. Διαβάζουμε την ακριβής τοποθεσία των εκτελέσιμων αρχείων και δημιουργούμε μια λίστα με τις ταμπέλες τους.

```
import os
from os import listdir

directories_with_labels = [("Benign PE Samples", 0), ("Malicious PE
Samples", 1)]
list_of_samples = []
labels = []
for dataset_path, label in directories_with_labels:
    samples = [f for f in listdir(dataset_path)]
    for sample in samples:
        file_path = os.path.join(dataset_path, sample)
        list_of_samples.append(file_path)
        labels.append(label)
```

Πηγή: Tsukerman, E. (2019). Machine Learning for Cybersecurity Cookbook. Packt Publishing.

Κομμάτι Κώδικα 29

4. Ορίζουμε μια συνάρτηση η οποία διαβάζει την ακολουθία των byte ενός αρχείου.

```
def read_file(file_path):
    """Reads in the binary sequence of a binary file."""
    with open(file_path, "rb") as binary_file:
        data = binary_file.read()
    return data
```

5. Ορίζουμε το μέγιστο όριο των byte που θα διαβάζει η συνάρτησή μας ανά δείγμα και εκχωρούμε όλα τα byte των δειγμάτων μας στον πίνακα X.

```
max_size = 15000
num_samples = len(list_of_samples)
X = np.zeros((num_samples, 8, max_size))
Y = np.asarray(labels)
file_num = 0
for file in tqdm(list_of_samples):
    sample_byte_sequence = read_file(file)
    for i in range(min(max_size, len(sample_byte_sequence))):
        X[file_num, :, i] = embed_bytes(sample_byte_sequence[i])
    file_num += 1
```

Πηγή: Tsukerman, E. (2019). Machine Learning for Cybersecurity Cookbook. Packt Publishing

Κομμάτι Κώδικα 30

6. Κάνουμε διαχωρισμό δεδομένων σε δεδομένα εκπαίδευσης και δεδομένα δοκιμών όπως στο μοντέλο static malware detector.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, Y, test_size=0.3, stratify=Y, random_state=11
)
```

7. Ορίζουμε έναν optimizer για τον ρυθμό εκμάθησης του μοντέλου μας.

```
from keras import optimizers

my_opt = optimizers.SGD(lr=0.01, decay=1e-5, nesterov=True)
```

8. Δημιουργούμε το νευρωνικό μας δίκτυο.

```
from keras import Input
from keras.layers import Conv1D, Activation, multiply,
GlobalMaxPool1D, Dense
from keras import Model

inputs = Input(shape=(8, max_size))
conv1 = Conv1D(kernel_size=(128), filters=32, strides=(128),
               padding='same')(inputs)
conv2 = Conv1D(kernel_size=(128), filters=32, strides=(128),
               padding='same')(inputs)
a = Activation('sigmoid', name='sigmoid')(conv2)
mul = multiply([conv1, a])
b = Activation('relu', name='relu')(mul)
p = GlobalMaxPool1D()(b)
d = Dense(16)(p)
predictions = Dense(1, activation='sigmoid')(d)
model = Model(inputs=inputs, outputs=predictions)
```

Πηγή: Tsukerman, E. (2019). Machine Learning for Cybersecurity Cookbook. Packt Publishing

Κομμάτι Κώδικα 31

9. Συντάσσουμε το μοντέλο μας και ορίζουμε τον αριθμό των μερίδων των αρχείων για επεξεργασία.

```
model.compile(optimizer=my_opt, loss="binary_crossentropy",
metrics=["acc"])
print(model.summary())
```

10. Εκπαιδεύουμε το μοντέλο μας και στη συνέχεια το εφαρμόζουμε πάνω στα δεδομένα δοκιμών και εκτυπώνουμε τα αποτελέσματα.

```
model.fit(X_train, y_train, epochs=5, batch_size=16)
```

```
print(model.metrics_names)
print(model.evaluate(X_test, y_test))
```

Αρχικά κάνουμε import τις βιβλιοθήκες *numpy* και *tqdm* οι οποίες θα μας χρησιμεύουν στο στον καλύτερο έλεγχο της προόδου του μοντέλου μας εμφανίζοντας μια μπάρα προόδου με ποσοστό επί τις εκατό (βήμα 1). Κατά τη διαδικασία εκχώρησης των *byte* ενός αρχείου σε έναν πίνακα αυτό που πραγματικά συμβαίνει είναι το εξής, κάθε *byte* αντιπροσωπεύει συντεταγμένες σε έναν πίνακα 8 διαστάσεων (βήμα 2). Ένα bit το οποίο ισούται με 1 σημαίνει ότι η αντίστοιχη τιμή των συντεταγμένων είναι ίση με 1/16, ενώ αν ένα bit ισούται με 0 τότε η αντίστοιχη τιμή των συντεταγμένων είναι ίση με -1/16. Για παράδειγμα, το byte **10010001** παίρνει την παρακάτω μορφή ως διάνυσμα πίνακα (1/16, -1/16, -1/16, 1/16, -1/16, -1/16,-1/16, 1/16). Στο βήμα 3 δημιουργούμε 2 πίνακες, έναν με τα αρχεία και έναν με τις αντίστοιχες τιμές τους και στο βήμα 4 ορίζουμε μια συνάρτηση η οποία διαβάζει ένα αρχείο σαν μια ακολουθία από Bytes. Στο βήμα 5 χρησιμοποιούμε τη βιβλιοθήκη *tqdm* για να παρακολουθήσουμε την πρόοδο του βρόγχου επανάληψης. Για κάθε αρχείο, διαβάζουμε την ακολουθία των byte και εκχωρούμε κάθε byte σε έναν περιβάλλον 8 διαστάσεων και στη συνέχεια στον πίνακα *X*. Εάν κάποιο αρχείο ξεπερνά σε μέγεθος τα 15000 byte, τότε η ανάγνωσή του σταματάει επί τόπου και συνεχίζει ο βρόγχος με το επόμενο αρχείο. Αν ο αριθμός των Byte είναι μικρότερος από 15000 τότε τα bytes θεωρούνται ότι είναι ίσα με το μηδέν. Η παράμετρος *max_size = 15000* η οποία εκπροσωπεί τον αριθμό των Byte που διαβάζονται για κάθε αρχείο μπορεί να μεταβληθεί ανάλογα την υπολογιστική ισχύ, τη διαθέσιμη μνήμη και τον αριθμό των δεδομένων προς επεξεργασία. Στα επόμενα βήματα (βήματα 6 και 7) ορίζουμε τον *SGD* (*stochastic gradient descent*) optimizer ο οποίος εφαρμόζει τη βελτιστοποίηση με τη μέθοδο της στοχαστικής κλίσης καθόδου και ορίζει τον ρυθμό εκμάθησης του μοντέλου και στη συνέχεια φτιάχνουμε την αρχιτεκτονική του νευρωνικού μας δικτύου ορίζοντας τα διάφορα στρώματα και τις συναρτήσεις ενεργοποίησης. Έπειτα επιλέγουμε τον αριθμό των μερίδων που θα εισέρχονται τα δεδομένα στο νευρωνικό δίκτυο (βήμα 8) και τέλος εκπαιδεύουμε το μοντέλο μας και το εφαρμόζουμε πάνω στο σετ των δεδομένων δοκιμών και εκτυπώνουμε την ευστοχία του μοντέλου.

ΚΕΦΑΛΑΙΟ 5

ΠΑΡΟΥΣΙΑΣΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ-ΑΞΙΟΛΟΓΗΣΗ ΚΑΙ ΣΥΜΠΕΡΑΣΜΑΤΑ

5.1Σύνολα δεδομένων

Στη συγκεκριμένη εργασία, χρησιμοποιήθηκαν τα παρακάτω σύνολα δεδομένων:

- *SMSSpamCollection1* το οποίο εμπεριέχει 5572 διαφορετικά μηνύματα SMS από τα οποία τα 4825 είναι ham και 746 είναι spam. Το σύνολο δεδομένων βρίσκεται στην ιστοσελίδα https://archive.ics.uci.edu/ml/datasets/sms+spam+collection
- *SMSSpamCollection2* το οποίο εμπεριέχει 5559 διαφορετικά μηνύματα SMS από τα οποία τα 4812 είναι ham και 747 είναι spam. Το σύνολο δεδομένων βρίσκεται στην ιστοσελίδα https://www.kaggle.com/uciml/sms-spam-collection-dataset
- *Spam_set_1* το οποίο εμπεριέχει στο σύνολο 4198 διαφορετικά email από τα οποία τα 2801 είναι ham και 1397 spam. Το σύνολο δεδομένων βρίσκεται στην ιστοσελίδα https://github.com/PacktPublishing/Machine-Learning-for-Cybersecurity-Cookbook/blob/master/Chapter06/Spam%20Filtering%20Using%20Machine%20Learning/spamassassin-public-corpus.7z
- *Spam_set_2* το οποίο εμπεριέχει στο σύνολο 3052 διαφορετικά email από τα οποία τα 2551 είναι ham και 501 spam. Το σύνολο δεδομένων βρίσκεται στην ιστοσελίδα https://www.kaggle.com/veleon/ham-and-spam-dataset
- *Benign PE Samples και Benign PE Samples 2* τα οποία εμπεριέχουν 49 και 63 καλοήθη εκτελέσιμα αρχεία των windows αντίστοιχα. Τα σύνολα δεδομένων συλλέχθηκαν από τους φακέλους συστήματος των windows που βρίσκονται σε κάθε υπολογιστή.
- *Malicious PE Samples* και *Malicious PE Samples 2* τα οποία εμπεριέχουν 47 και 59 κακοήθη εκτελέσιμα αρχεία των windows. Το σύνολο δεδομένων Malicious PE Samples βρίσκεται στην ιστοσελίδα https://github.com/PacktPublishing/Machine-Learning-for-Cybersecurity-Cookbook/tree/master/PE%20Samples%20Dataset και το Malicious PE Samples 2 στην ιστοσελίδα http://www.tekdefense.com/downloads/malware-samples/ τα οποία αποτελούν σύνολα δεδομένων μόνο για εκπαιδευτικούς σκοπούς.

5.2 Περιγραφή πειραμάτων

Έγιναν τα εξής πειράματα:

- Κατηγοριοποίηση μηνυμάτων τύπου SMS σε spam και ham χρησιμοποιώντας τον αλγόριθμο *Naïve Bayes* πάνω στα σύνολα δεδομένων *SMSSpamCollection1* και *SMSSpamCollection2* που εμπεριέχουν SMS μηνύματα. Πρόγραμμα *SMS_spam_filter*.
- Κατηγοριοποίηση email σε spam και ham χρησιμοποιώντας κατηγοριοποιητή της μορφής δέντρου απόφασης, πάνω στο σύνολα δεδομένων *Spam_set_1* με τα οποία θα γίνει και η εκπαίδευση του μοντέλου. Μετά την εκπαίδευσή του, το μοντέλο εφαρμόζεται πάνω στο σύνολο δεδομένων *Spam_set_2* το οποίο δεν έχει ξανασυναντήσει. Πρόγραμμα *Email_spam_detection*.
- Κατηγοριοποίηση μέσω στατικής ανάλυσης των εκτελέσιμων αρχείων των *windows* του συνόλου δεδομένων *Benign PE Samples* και *Malicious PE Samples* με το οποίο θα γίνει και η εκπαίδευση του μοντέλου χρησιμοποιώντας κατηγοριοποιητή *Random Forest*. Στη συνέχεια το μοντέλο θα δοκιμαστεί με το σύνολο δεδομένων *Benign PE Samples 2 και Malicious PE Samples 2* το οποίο δεν έχει ξανασυναντήσει. Πρόγραμμα *Static_Malware_Detector*.

- Κατηγοριοποίηση μέσω τεχνητών νευρωνικών δικτύων των εκτελέσιμων αρχείων των windows του συνόλου δεδομένων *Benign PE Samples* και *Malicious PE Samples* με το οποίο θα γίνει και η εκπαίδευση του μοντέλου. Στη συνέχεια το μοντέλο θα δοκιμαστεί με το σύνολο δεδομένων *Benign PE Samples 2* και *Malicious PE Samples 2* το οποίο δεν έχει ξανασυναντήσει. Πρόγραμμα *Deep_Learning_Malware_Detection*.

5.2.1 SMS spam filter

SMSSpamCollection1	**SMSSpamCollection2**
Correct: 1100	Correct: 1096
Incorrect: 14	Incorrect: 16
Accuracy: 0.9874326750448833	Accuracy: 0.9856115107913669

Πίνακας 5 SMSSpamCollection

Το ποσοστό ευστοχίας του αλγορίθμου είναι κοντά στο 98% και με τα δύο σετ δεδομένων το οποίο είναι πολύ υψηλό. Συνοψίζοντας, καταφέραμε να κωδικοποιήσουμε ένα φίλτρο ανεπιθύμητης αλληλογραφίας για μηνύματα SMS χρησιμοποιώντας τον αλγόριθμο Naive Bayes. Το φίλτρο είχε ακρίβεια περίπου 98% στο σύνολο δοκιμών που χρησιμοποιήσαμε, το οποίο είναι ένα πολλά υποσχόμενο αποτέλεσμα. Ο αρχικός μας στόχος ήταν η ακρίβεια άνω του 80% και καταφέραμε να το πετύχουμε. Το φίλτρο ανεπιθύμητης αλληλογραφίας πέρασε 1114 μηνύματα τα οποία δεν τα είχε ξαναδεί και κατηγοριοποίησε σωστά τα 1100 στο σετ δεδομένων *SMSSpamCollection1* ενώ στο σετ δεδομένων SMSSpamCollection2 πέρασε 1112 μηνύματα τα οποία δεν τα είχε ξαναδεί και κατηγοριοποίησε σωστά τα 1096.

5.2.2 Email spam detector

Spam_set_1	Spam (Πραγματικά)	Ham (Πραγματικά)
Spam (Προβλέψεις)	281	4
Ham (Προβλέψεις)	4	547
Accuracy	0.9904306220095693	

Πίνακας 6 Spam_set

Spam_set_2	Spam (Πραγματικά)	Ham (Πραγματικά)
Spam (Προβλέψεις)	489	0
Ham (Προβλέψεις)	0	2550
Accuracy	1	

Πίνακας 7 Spam_set

Το Decision Tree μοντέλο εφαρμόστηκε στο 20% των δεδομένων εκπαίδευσης (Spam_set_1) και έχει ένα πάρα πολύ υψηλό ποσοστό ευστοχίας, της τάξης του 99%. Στη συνέχεια έγινε δοκιμή σε ένα καινούριο σετ δεδομένων το οποίο είναι διαφορετικό από αυτό της

εκπαίδευσης και παρατηρούμε ότι το ποσοστό ευστοχίας είναι στο 100%. Σύμφωνα με τα παραπάνω αποτελέσματα των πειραμάτων συμπεραίνουμε ότι το συγκεκριμένο μοντέλο λειτουργεί σε άριστα επίπεδα.

5.2.3 Static malware detector

Το μοντέλο Random Forest που εφαρμόστηκε πάνω στα σετ δεδομένων των εκτελέσιμων αρχείων έχει πολλά διαφορετικά αποτελέσματα καθώς είναι πολυπλοκότερο από τα προηγούμενα και κάθε φορά που τρέχει οι μεταβλητές του τυχαίου δάσους εφαρμόζονται με τυχαία σειρά. Παρατηρήθηκε επίσης ότι το συγκεκριμένο μοντέλο έχει καλύτερα αποτελέσματα όταν το ποσό των κακόβουλων εκτελέσιμων αρχείων είναι το ίδιο με τα καλόβουλα. Σε αντίθετη περίπτωση παρατηρούνται πολύ χαμηλά ποσοστά ευστοχίας και μη αναγνώρισης της συγκεκριμένης ομάδας αρχείων με τα λιγότερα δείγματα. Τα πιο αισιόδοξα ποσοστά ευστοχίας του συγκεκριμένου μοντέλου προκύπτουν με ποσοστό 85% στα δεδομένα επικύρωσης (20% των δεδομένων εκπαίδευσης) και φαίνονται στον παρακάτω πίνακα :

	Benign (Πραγματικά)	Malware (Πραγματικά)
Benign (Προβλέψεις)	50	21
Malware (Προβλέψεις)	13	38
Accuracy	0.7001030927835051	

Πίνακας 8 Προβλέψεις

Το συγκεκριμένο μοντέλο παρουσιάζει ποσοστό ευστοχίας κοντά στο 70% το οποίο θεωρείται αποδεκτό αν και είναι αρκετά χαμηλό. Αυτό συμβαίνει λόγω χαμηλού αριθμού δεδομένων εκπαίδευσης και συγκεκριμένα κακόβουλων εκτελέσιμων αρχείων των windows τα οποία είναι αρκετά δυσεύρετα.

Η δυναμική ανάλυση κακόβουλου λογισμικού αποτελεί μια πιο προχωρημένη διαδικασία ανάλυσης η οποία θα μπορούσε να χρησιμοποιηθεί μελλοντικά για τη επίλυση του υπάρχοντος προβλήματος. Σε αντίθεση με τη στατική ανάλυση, η δυναμική ανάλυση είναι μια τεχνική ανάλυσης κακόβουλου λογισμικού στην οποία ο ειδικός εκτελεί το δείγμα και στη συνέχεια μελετά τη συμπεριφορά του δείγματος καθώς εκτελείται. Το κύριο πλεονέκτημα της δυναμικής ανάλυσης έναντι της στατικής είναι ότι επιτρέπει την παράκαμψη της συσκότισης απλά παρατηρώντας πώς συμπεριφέρεται ένα δείγμα, αντί να προσπαθούμε να αποκρυπτογραφήσουμε το περιεχόμενο και τη συμπεριφορά του δείγματος. Δεδομένου ότι το κακόβουλο λογισμικό είναι εγγενώς μη ασφαλές, οι ερευνητές καταφεύγουν στην εκτέλεση των δειγμάτων σε μια εικονική μηχανή (Virtual Machine). Η διαδικασία αυτή ονομάζεται sandboxing.

5.2.4 Deep Learning Malware Detector

Το μοντέλο των νευρωνικών δικτύων εκπαιδεύτηκε με το 70% των *Benign* και *Malicious PE Samples*. Στη συνέχεια έγινε η αξιολόγησή του με το υπόλοιπο 30% του σετ και τέλος έγινε

δοκιμή σε αυτό με το σετ *Benign/Malicious PE Samples 2.* Όπως και με το προηγούμενο μοντέλο *Random Forest* έτσι και σε αυτό, κάθε φορά που τρέχει το πρόγραμμα οι μεταβλητές των στρωμάτων του δικτύου εφαρμόζονται με τυχαία σειρά, οπότε το μοντέλο κάθε φορά θα βγάζει διαφορετικό αποτέλεσμα. Μετά από 100 περίπου δοκιμές το καλύτερο μοντέλο αποθηκεύτηκε με το όνομα *Model811.h5* και παρακάτω παρουσιάζονται τα αποτελέσματά του:

	Benign (Πραγματικά)	Malware (Πραγματικά)
Benign (Προβλέψεις)	10	1
Malware (Προβλέψεις)	7	13
Accuracy	0.7419354915618896	

Πίνακας 9 Σετ δεδομένων αξιολόγησης (Validation Set)

	Benign (Πραγματικά)	Malware (Πραγματικά)
Benign (Προβλέψεις)	44	4
Malware (Προβλέψεις)	19	55
Accuracy	0.811475396156311	

Πίνακας 10 Σετ δεδομένων δοκιμών (Benign / Malicious PE Samples 2)

Το ποσοστό ευστοχίας του Model811.h5 πάνω σε ένα καινούριο σετ δεδομένων είναι της τάξης του 81%, το οποίο αποτελεί αρκετά καλό αποτέλεσμα. Επίσης στη συγκεκριμένη περίπτωση όπως και στην προηγούμενη τα δεδομένα εκπαίδευσης δεν είναι αρκετά λόγω δυσκολίας εύρεσής τους.

ΣΥΜΠΕΡΑΣΜΑΤΑ

Αυτή η διατριβή ερεύνησε και στόχευε στη μελέτη της εφαρμογής μοντέλων Μηχανικής Μάθησης για την ανίχνευση κακόβουλου λογισμικού και κακόβουλων δραστηριοτήτων. Αναφέρθηκαν οι κύριες αρχές της ταξινόμησης και της ομαδοποίησης στην περίπτωση της εποπτευόμενης μάθησης. Σύνολα δεδομένων εισήχθησαν και υποβλήθηκαν σε επεξεργασία για την εκτέλεση εντοπισμού κακόβουλου λογισμικού.

Παρόλο που έχουν αναπτυχθεί αρκετές αποτελεσματικές λύσεις για την αντιμετώπιση κακόβουλου λογισμικού και άγνωστων επιθέσεων που σχετίζονται με δραστηριότητες στο διαδίκτυο, οι μελλοντικές εκδόσεις παρόμοιων κακόβουλων προγραμμάτων αναμένεται να γίνουν πιο περίπλοκες και προβληματικές. Τα μέσα μόλυνσης είναι πιθανό να αλλάξουν από υπολογιστές συνδεδεμένους στο διαδίκτυο σε τερματικά κινητών τηλεφώνων, καθώς οι εισβολείς τείνουν να προσελκύονται στα συστήματα που χρησιμοποιούνται ευρέως. Επίσης, τα περιβάλλοντα νέφους είναι εύκολοι στόχοι και πρέπει να σχεδιαστούν σωστά για να αποτρέψουν τη διείσδυσή τους από κακόβουλο λογισμικό.

Για το μέλλον, υπάρχει ανάγκη βελτίωσης των αλγορίθμων που προτείνονται ή να προταθούν καινούργιοι. Η ανακάλυψη νέων και καλύτερων αλγορίθμων ομαδοποίησης και ταξινόμησης, καθώς και το πεδίο ανάλυσης και ανίχνευσης κακόβουλου λογισμικού, είναι καινούριο και τώρα βρίσκεται στη διαδικασία της έρευνας. Μια άλλη προσέγγιση για το πώς μπορεί να γίνει ορατό το κακόβουλο λογισμικό είναι η ανάπτυξη αλγορίθμων οπτικοποίησης. Έτσι, μια ενδιαφέρουσα προσέγγιση θα μπορούσε να είναι η δοκιμή του ίδιου συνόλου δεδομένων με τους νέους μετασχηματισμούς και άλλα σύνολα δεδομένων επίσης.

Εν κατακλείδι, αυτή η διατριβή είχε σκοπό να τονίσει την ανάγκη εφαρμογής της τεχνητής νοημοσύνης για την αντιμετώπιση των δειγμάτων κακόβουλου λογισμικού, την ανάλυση της συμπεριφοράς τους και να δούμε πώς διάφοροι αλγόριθμοι λειτουργούν την ταξινόμηση και την ομαδοποίηση. Τα προγράμματα κακόβουλου λογισμικού καθημερινά δημιουργούνται για να παρακάμψουν τα anti-virus, τα τείχη προστασίας και γενικά τα συστήματα ανίχνευσης και υπάρχει ανάγκη επανεκτίμησης των τρεχουσών προσεγγίσεων και επανεξέτασης του τρόπου προσέγγισής τους.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Y. G. Kim J. Koo and S. H. Lee. Security requirements for cloud-based C4I security architecture. In 2019 International Conference on Platform Technology and Service (PlatCon), pages 1–4, January 2019.

[2] N. Doukas O. P. Markovskyi P. Stavroulakis M. Kolisnyk V. Kharchenk and N. G. Bardis. Reliability, Fault Tolerance and Other Critical Components for Survivability in Information Warfare, volume 990, pages 346–370. Springer, Cham, 2017.

[3] Wesley Chai. Confidentiality, integrity and availability (CIA triad) https://whatis.techtarget.com/definition/Confidentiality-integrity-and-availability-CIA#:~:text=Confidentiality%2C%20integrity%20and%20availability%2C%20also,with%20the%20Central%20Intelligence%20Agency. (Πρόσβαση: Δεκέμβριος 2020)

[4] Peter Stavroulakis. Reliability, survivability and quality of large-scale telecommunication systems: case study: Olympic games. John Wiley and Sons, 2004.

[5] Peter Stavroulakis Doukas, Nikolaos and Nikolaos Bardis. Review of Artificial Intelligence Cyber Threat Assessment Techniques for Increased System Survivability., pages 207–222. Springer, Cham, 2021.

[6] Taddeo M. McCutcheon T. and Floridi L. Trusting artificial intelligence in cybersecurity is a double-edged sword. Nature Machine Intelligence, 1(12):557–560, 2019.

[7] Vyacheslav Kharchenko, Oleg Illiashenko, Olga Morozova, and Sergii Sokolov. Combination of digital twin and artificial intelligence in manufacturing using industrial IoT. In 2020 IEEE 11th international conference on dependable systems, services and technologies (DESSERT), pages 196–201. IEEE, 2020.

[8] Doukas, N. et al. (2022). Survivability Using Artificial Intelligence Assisted Cyber Risk Warning. In: Stamp, M., Aaron Visaggio, C., Mercaldo, F., Di Troia, F. (eds) Artificial Intelligence for Cybersecurity. Advances in Information Security, vol 54. Springer, Cham. https://doi.org/10.1007/978-3-030-97087-1_12

[9] Eric M. Hutchins. The Cyber Kill Chain https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html (Πρόσβαση: Δεκέμβριος 2020)
[10] Russell, Stuart J.; Norvig, Peter (2010). Artificial Intelligence: A Modern Approach (Third ed.). Prentice Hall.
[11] Pavel Brazdil, Christophe Giraud Carrier, Carlos Soares, Ricardo Vilalta (2009). *Metalearning: Applications to Data Mining*(Fourth ed.). Springer Science+Business Media.
[12] Jason Brownlee. Supervised and Unsupervised Machine Learning Algorithms https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/ (Πρόσβαση: Νοέμβριος 2020)
[13] Stuart J. Russell, Peter Norvig (2010) *Artificial Intelligence: A Modern Approach*, *Third Edition*, Prentice Hall
[14] Hinton, Geoffrey; Sejnowski, Terrence (1999). Unsupervised Learning: Foundations of Neural Computation. MIT Press.
[15] Jordan, Michael I.; Bishop, Christopher M. (2004). "Neural Networks". In Allen B. Tucker (ed.). *Computer Science Handbook, Second Edition (Section VII: Intelligent Systems)*. Boca Raton, Florida: Chapman & Hall/CRC Press LLC
[16] van Otterlo, M.; Wiering, M. (2012). Reinforcement learning and markov decision processes. Reinforcement Learning. Adaptation, Learning, and Optimization.
[17] larence Chio, D. F. (n.d.). Machine Learning & Cybersecurity. O'REILLY.
[18] Soma Halder, S. O. (2018). *Machine Learning for Cybersecurity* . Birmingham: Packt.
[19] S. Bozinovski "Teaching space: A representation concept for adaptive pattern classification" COINS Technical Report No. 81-28, Computer and Information Science Department, University of Massachusetts at Amherst, MA, 1981. https://web.cs.umass.edu/publication/docs/1981/UM-CS-1981-028.pdf
[20] Aggarwal, Charu C., editor. Reddy, Chandan K., editor. Data Clustering : Algorithms and Applications.
[21] Agresti, Alan (2002). Categorical Data Analysis. Hoboken: John Wiley and Sons.
[22] Cisco. (2011). What is network security?. www.cisco.com (Πρόσβαση Οκτώβριος 2020)
[23] "Endpoint security management overview". https://searchsecurity.techtarget.com/definition/endpoint-security-management . (Πρόσβαση Οκτώβριος 2020).
[24] Velasco, Roberto (7 May 2020). "What is IAST? All About Interactive Application Security Testing
[25] Soma Halder, S. O. (2018). *Machine Learning for Cybersecurity* . Birmingham: Packt.
[26] Komari, I. E., Fedorenko, M., Kharchenko, V., Yehorova, Y., Bardis, N., & Lutai, L. The Neural Modules Network with Collective Relearning for the Recognition of Diseases: Fault-Tolerant Structures and Reliability Assessment. neural networks, 1, 3.
[27] What is a Perceptron https://deepai.org/machine-learning-glossary-and-terms/perceptron#:~:text=A%20Perceptron%20is%20an%20algorithm,a%20single%2Dlayer%20neural%20network. (Πρόσβαση: Οκτώβριος 2020)
[28] Erica Kastner. What is email filtering and how does it work? https://www.soscanhelp.com/blog/what-is-email-spam-filtering-and-how-does-it-work#:~:text=As%20described%20above%2C%20email%20filtering,to%20assess%20an%20incoming%20email. (Πρόσβαση : Δεκέμβριος 2020)
[29] Email filtering. . *Zonk. "How Pervasive is ISP Outbound Email Filtering?". Slashdot.org. Slashdot.org.*

[30] IBM Cloud Education. Supervised Learning https://www.ibm.com/cloud/learn/supervised-learning (Πρόσβαση: Δεκέμβριος 2020)
[31] Ankit Bisht. ML | Classification vs Regression https://www.geeksforgeeks.org/ml-classification-vs-regression/ (Πρόσβαση : Δεκέμβριος 2020)
[32] Jason Brownlee. 4 Types of Classification Tasks in Machine Learning https://machinelearningmastery.com/types-of-classification-in-machine-learning/ (Πρόσβαση: Δεκέμβριος 2020)
[33] Stuart J. Russell, Peter Norvig (2010) Artificial Intelligence: A Modern Approach, Third Edition, Prentice Hall
[34] Jason Brownlee. Overfitting and Underfitting With Machine Learning Algorithms https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/#:~:text=Overfitting%20in%20Machine%20Learning,the%20model%20on%20new%20data. (Πρόσβαση: Νοέμβριος 2020)
[35] Alex Castrounis. Machine Learning: An In-Depth Guide - Model Evaluation, Validation, Complexity, and Improvement. https://www.innoarchitech.com/blog/machine-learning-an-in-depth-non-technical-guide-part-3 (Πρόσβαση : Δεκέμβριος 2020)
[36] Onel Harrison. Machine Learning Basics with the K-Nearest Neighbors Algorithm. https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761#:~:text=Summary-,The%20k%2Dnearest%20neighbors%20(KNN)%20algorithm%20is%20a%20simple,that%20data%20in%20use%20grows. (Πρόσβαση: Οκτώβριος 2020)
[37] Jason Brownlee. K-Nearest Neighbors for Machine Learning https://machinelearningmastery.com/k-nearest-neighbors-for-machine-learning/ (Πρόσβαση: Οκτώβριος 2020)
[38] Parisi, A. (n.d.). *Artificial Intelligence for Cybesecurity.*
[39] Bailey, T., Jain, A. A note on distance-weighted k-nearest neighbor rules. IEEE Trans. Systems, Man, Cybernetics, Vol. 8
[40] Dudani, S.A. The distance-weighted k-nearest-neighbor rule. IEEE Trans. Syst. Man Cybern
[41] Naresh Kumar. Advantages and Disadvantages of KNN Algorithm in Machine Learning http://theprofessionalspoint.blogspot.com/2019/02/advantages-and-disadvantages-of-knn.html (Πρόσβαση: Δεκέμβριος 2020)
[42] Jason Brownlee. Linear Regression for Machine Learning. https://machinelearningmastery.com/linear-regression-for-machine-learning/ (Πρόσβαση : Δεκέμβριος 2020)
[43] Charles Darwin. The Variation of Animals and Plants under Domestication. (1868) (Chapter XIII describes what was known about reversion in Galton's time. Darwin uses the term "reversion".)
[44] Jason Brownlee. Difference Between Classification and Regression in Machine Learning https://machinelearningmastery.com/classification-versus-regression-in-machine-learning/#:~:text=Fundamentally%2C%20classification%20is%20about%20predicting,is%20about%20predicting%20a%20quantity.&text=That%20classification%20is%20the%20problem,quantity%20output%20for%20an%20example. (Πρόσβαση: Δεκέμβριος 2020)
[45] Amiya Ranjan Rout. ML – Advantages and Disadvantages of Linear Regression https://www.geeksforgeeks.org/ml-advantages-and-disadvantages-of-linear-regression/ (Πρόσβαση: Δεκέμβριος 2020)
[46] Sambhav Khurana. Naive Bayes Classifiers https://www.geeksforgeeks.org/naive-bayes-classifiers/ (Πρόσβαση: Δεκέμβριος 2020)

[47] Pavan Vadapalli. Naive Bayes Explained: Function, Advantages & Disadvantages, Applications in 2021 https://www.upgrad.com/blog/naive-bayes-explained/ (Πρόσβαση: Φεβρουάριος 2021)
[48] Guido, A. C. (2016). *Introduction to Machine Learning with Python.* O'REILLY.
[49] Shalev-Shwartz, Shai; Ben-David, Shai (2014). "18. Decision Trees". Understanding Machine Learning. Cambridge University Press.
[50] Sanjeev Arora (n.d.). Decision Tree Complexity. Princeton University
[51] Rokach, Lior; Maimon, O. (2008). Data mining with decision trees: theory and applications. World Scientific Pub Co Inc.
[52] Breiman, Leo; Friedman, J. H.; Olshen, R. A.; Stone, C. J. (1984). Classification and regression trees. Monterey, CA: Wadsworth & Brooks/Cole Advanced Books & Software.
[53] "Malware: Types, Protection, Prevention, Detection & Removal - Ultimate Guide". EasyTechGuides.
[54] Klein, Tobias (11 October 2011). A Bug Hunter's Diary: A Guided Tour Through the Wilds of Software Security. No Starch Press
[55] A Comparative Assessment of Malware Classification using Binary Texture Anal-ysis and Dynamic Analysis-Lakshmanan Nataraj, Vinod Yegneswaran, Phillip Porras, Jian Zhang *Proceedings of ACM CCS Workshop on Artificial Intelligence and Secu-rity(AISEC,2011*).
[56] Anderson, B., Storlie, C. and Lane, T. (2012) Improving Malware Classification: Bridging the Static/Dynamic Gap. *Proceedings of* 5*th ACM Workshop on Security and Artificial Intelligence* (*AISec*)
[57] Bailey, M., Oberheide, J., Andersen, J., Mao, Z. M., Jahanian, F., & Nazario, J. (2007). Automated classification and analysis of Internet malware. In *Recent Advances in Intrusion Detection - 10th International Symposium, RAID 2007, Proceedings.* (Vol. 4637 LNCS, pp. 178-197). (Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics); Vol. 4637 LNCS).
[58] Forensics. Portable Executable File https://malware.news/t/portable-executable-file/32980 (Πρόσβαση: Οκτώβριος 2020)
[59] Savan Gadhiya, Kaushal Bhavsa , Techniques for Malware Analysis*, International Journal of Advanced Research in Computer Science and Software Engineering*. http://www.ijarcsse.com/docs/papers/Volume_3/4_April2013/V3I4-0371.pdf (Πρόσβαση: Δεκέμβριος 2020)
[60] Kris Kendall Practical Malware Analysis. www.black-hat.com/presentations/bh-dc-07/Kendall_McMillan/Paper/bh-dc-07-Kendall_McMillan-WP.pdf (Πρόσβαση: Δεκέμβριος 2020)
[61] Malware Analysis and Classification: A Survey- Gandotra, E., Bansal, D. and Sofat, S. *Journal of Information Security, 5, 56-64.* doi: 10.4236/jis.2014.52006 (2014).
[62] Joshua Saxe, H. S. (September 25, 2018). *Malware Data Science: Attack Detection and Attribution.*
[63] Rains, T. (May 2020). *Cybersecurity Threats, Malware Trends, and Strategies.* Packt.
[64] Mohd Shaiful Anuar Bin Mohamad Sari,Mohd Aizaini Maarof. Classification of Malware Family Using Decision Tree Algorithm.(2017)
[65] Annachhatre, C.: Hidden Markov models for malware classification. Department of Computer Science, San Jose State University, Master's report (2013)
[66] Padmavathi Ganapathi, D. S. (July, 2019). *Handbook of Research on Machine and Deep Learning Applications for Cyber Security.*

[67] Bailey, Katherine. "Adventures Learning Neural Nets And Python." *ohAI*. N.p., 2016. https://katbailey.github.io/post/neural-nets-in-python/ (Πρόσβαση: Νοέμβριος 2020]

[68] Iracleous, D. P., Bardis, N. G., & Doukas, N. (2013). Consensus algorithms within the C4ISR architecture. Journal of Computations & Modelling, 3(4), 103-114. http://www.scienpress.com/Upload/JCM/Vol%203_4_7.pdf

[69] Edward Ralf, John Barker, Jared Sylvester, Robert Brandon. Malware Detection by Eating a Whole EXE. https://arxiv.org/pdf/1710.09435.pdf

ΠΑΡΑΡΤΗΜΑΤΑ

1) **SMS spam filter**

```
import pandas as pd

sms_spam = pd.read_csv('SMSSpamCollection2', sep='\t', header=None,
names=['Label', 'SMS'])

data_randomized = sms_spam.sample(frac=1, random_state=7)

training_test_index = round(len(data_randomized) * 0.8)

training_set =
data_randomized[:training_test_index].reset_index(drop=True)
test_set =
data_randomized[training_test_index:].reset_index(drop=True)

training_set['SMS'] = training_set['SMS'].str.replace('\W', ' ',
regex=True
training_set['SMS'] = training_set['SMS'].str.lower()

training_set['SMS'] = training_set['SMS'].str.split
vocabulary = []
for sms in training_set['SMS']:
    for word in sms:
        vocabulary.append(word)
vocabulary = list(set(vocabulary))

word_counts_per_sms = {unique_word: [0] * len(training_set['SMS'])
for unique_word in vocabulary}

for index, sms in enumerate(training_set['SMS']):
    for word in sms:
        word_counts_per_sms[word][index] += 1

word_counts = pd.DataFrame(word_counts_per_sms)


training_set_clean = pd.concat([training_set, word_counts],
                               axis=1)

spam_messages = training_set_clean[training_set_clean['Label'] ==
'spam']
ham_messages = training_set_clean[training_set_clean['Label'] ==
'ham']
p_spam = len(spam_messages) / len(training_set_clean)
p_ham = len(ham_messages) / len(training_set_clean)
n_words_per_spam_message = spam_messages['SMS'].apply(len)
n_spam = n_words_per_spam_message.sum()
```

```
n_words_per_ham_message = ham_messages['SMS'].apply(len)
n_ham = n_words_per_ham_message.sum()
n_vocabulary = len(vocabulary)
alpha = 1

parameters_spam = {unique_word: 0 for unique_word in vocabulary}
parameters_ham = {unique_word: 0 for unique_word in vocabulary}

for word in vocabulary:
    n_word_given_spam = spam_messages[word].sum()
    p_word_given_spam = (n_word_given_spam + alpha) / (n_spam + alpha
* n_vocabulary)
    parameters_spam[word] = p_word_given_spam

    n_word_given_ham = ham_messages[word].sum()
    p_word_given_ham = (n_word_given_ham + alpha) / (n_ham + alpha *
n_vocabulary)
    parameters_ham[word] = p_word_given_ham


import re


def classify(message):
    '''
    message: a string
    '''

    message = re.sub('\W', ' ', message)
    message = message.lower().split()

    p_spam_given_message = p_spam
    p_ham_given_message = p_ham

    for word in message:
        if word in parameters_spam:
            p_spam_given_message *= parameters_spam[word]

        if word in parameters_ham:
            p_ham_given_message *= parameters_ham[word]

    print('P(Spam|message):', p_spam_given_message)
    print('P(Ham|message):', p_ham_given_message)

    if p_ham_given_message > p_spam_given_message:
        print('Label: Ham')
    elif p_ham_given_message < p_spam_given_message:
        print('Label: Spam')
    else:
        print('Equal proabilities, have a human classify this!')
```

```
def classify_test_set(message):
    '''
    message: a string
    '''

    message = re.sub('\W', ' ', message)
    message = message.lower().split()

    p_spam_given_message = p_spam
    p_ham_given_message = p_ham

    for word in message:
        if word in parameters_spam:
            p_spam_given_message *= parameters_spam[word]

        if word in parameters_ham:
            p_ham_given_message *= parameters_ham[word]

    if p_ham_given_message > p_spam_given_message:
        return 'ham'
    elif p_spam_given_message > p_ham_given_message:
        return 'spam'
    else:
        return 'needs human classification'


test_set['predicted'] = test_set['SMS'].apply(classify_test_set)

correct = 0
total = test_set.shape[0]

for row in test_set.iterrows():
    row = row[1]
    if row['Label'] == row['predicted']:
        correct += 1

print('Correct:', correct)
print('Incorrect:', total - correct)
print('Accuracy:', correct / total)
```

2) Email spam detector

```
import os

spam_emails_path = os.path.join("spam_set_2", "spam")
ham_emails_path = os.path.join("spam_set_2", "ham")
labeled_file_directories = [(spam_emails_path, 0), (ham_emails_path,
1)]
```

```
print(labeled_file_directories)

# 2
email_corpus = []
labels = []
for class_files, label in labeled_file_directories:
    files = os.listdir(class_files)
    for file in files:
        file_path = os.path.join(class_files, file)
        try:
            with open(file_path, "r") as currentFile:
                email_content = currentFile.read().replace("\n", "")
                email_content = str(email_content)
                email_corpus.append(email_content)
                labels.append(label)
        except:
            pass

#3
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(email_corpus,
labels, test_size=0.2, random_state=11)
print(labels)

#4
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import HashingVectorizer,
TfidfTransformer
from sklearn import tree

nlp_followed_by_dt = Pipeline(
    [
        ("vect", HashingVectorizer(input="content", ngram_range=(1,
3))),
        ("tfidf", TfidfTransformer(use_idf=True,)),
        ("dt", tree.DecisionTreeClassifier(class_weight="balanced")),
    ]
)
nlp_followed_by_dt.fit(X_train, y_train)


#5
from sklearn.metrics import accuracy_score, confusion_matrix

y_test_pred = nlp_followed_by_dt.predict(X_test)
print(accuracy_score(y_test, y_test_pred))
print(confusion_matrix(y_test, y_test_pred))
```

3) Static Malware Detector

```
import os
from os import listdir

directories_with_labels = [("Benign PE Samples", 0), ("Malicious PE
Samples", 1)]
list_of_samples = []
labels = []
for dataset_path, label in directories_with_labels:
    samples = [f for f in listdir(dataset_path)]
    for sample in samples:
        file_path = os.path.join(dataset_path, sample)
        list_of_samples.append(file_path)
        labels.append(label)

# 2
from sklearn.model_selection import train_test_split

samples_train, samples_test, labels_train, labels_test =
train_test_split(
    list_of_samples, labels, test_size=0.3, stratify=labels,
random_state=11
)

# 3
import collections
from nltk import ngrams
import numpy as np
import pefile


def read_file(file_path):
    """Reads in the binary sequence of a binary file."""
    with open(file_path, "rb") as binary_file:
        data = binary_file.read()
    return data


def byte_sequence_to_Ngrams(byte_sequence, N):
    """Creates a list of N-grams from a byte sequence."""
    Ngrams = ngrams(byte_sequence, N)
    return list(Ngrams)


def binary_file_to_Ngram_counts(file, N):
    """Takes a binary file and outputs the N-grams counts of its
binary sequence."""
    filebyte_sequence = read_file(file)
```

```
    file_Ngrams = byte_sequence_to_Ngrams(filebyte_sequence, N)
    return collections.Counter(file_Ngrams)


def get_NGram_features_from_sample(sample,
K1_most_frequent_Ngrams_list):
    """Takes a sample and produces a feature vector.
    The features are the counts of the K1 N-grams we've selected.
    """
    K1 = len(K1_most_frequent_Ngrams_list)
    feature_vector = K1 * [0]
    file_Ngrams = binary_file_to_Ngram_counts(sample, N)
    for i in range(K1):
        feature_vector[i] =
file_Ngrams[K1_most_frequent_Ngrams_list[i]]
    return feature_vector


def preprocess_imports(list_of_DLLs):
    """Normalize the naming of the imports of a PE file."""
    temp = [x.decode().split(".")[0].lower() for x in list_of_DLLs]
    return " ".join(temp)


def get_imports(pe):
    """Get a list of the imports of a PE file."""
    list_of_imports = []
    for entry in pe.DIRECTORY_ENTRY_IMPORT:
        list_of_imports.append(entry.dll)
    return preprocess_imports(list_of_imports)


def get_section_names(pe):
    """Gets a list of section names from a PE file."""
    list_of_section_names = []
    for sec in pe.sections:
        normalized_name = sec.Name.decode().replace("\x00",
"").lower()
        list_of_section_names.append(normalized_name)
    return "".join(list_of_section_names)


# 4
N = 2
Ngram_counts_all = collections.Counter([])
for sample in samples_train:
    Ngram_counts_all += binary_file_to_Ngram_counts(sample, N)
K1 = 100
K1_most_frequent_Ngrams = Ngram_counts_all.most_common(K1)
K1_most_frequent_Ngrams_list = [x[0] for x in
K1_most_frequent_Ngrams]
```

```
# 5
imports_corpus_train = []
num_sections_train = []
section_names_train = []
Ngram_features_list_train = []
y_train = []
for i in range(len(samples_train)):
    sample = samples_train[i]
    try:
        NGram_features = get_NGram_features_from_sample(
            sample, K1_most_frequent_Ngrams_list
        )
        pe = pefile.PE(sample)
        imports = get_imports(pe)
        n_sections = len(pe.sections)
        sec_names = get_section_names(pe)
        imports_corpus_train.append(imports)
        num_sections_train.append(n_sections)
        section_names_train.append(sec_names)
        Ngram_features_list_train.append(NGram_features)
        y_train.append(labels_train[i])
    except Exception as e:
        print(sample + ":")
        print(e)

# 6
from sklearn.feature_extraction.text import HashingVectorizer,
TfidfTransformer
from sklearn.pipeline import Pipeline

imports_featurizer = Pipeline(
    [
        ("vect", HashingVectorizer(input="content", ngram_range=(1,
2))),
        ("tfidf", TfidfTransformer(use_idf=True, )),
    ]
)
section_names_featurizer = Pipeline(
    [
        ("vect", HashingVectorizer(input="content", ngram_range=(1,
2))),
        ("tfidf", TfidfTransformer(use_idf=True, )),
    ]
)
imports_corpus_train_transformed = imports_featurizer.fit_transform(
    imports_corpus_train
)
section_names_train_transformed =
section_names_featurizer.fit_transform(
    section_names_train
```

```
)

# 7
from scipy.sparse import hstack, csr_matrix

X_train = hstack(
    [
        Ngram_features_list_train,
        imports_corpus_train_transformed,
        section_names_train_transformed,
        csr_matrix(num_sections_train).transpose(),
    ]
)

# 8
from sklearn.ensemble import RandomForestClassifier

forest = RandomForestClassifier(n_estimators=100)
clf = forest.fit(X_train, y_train)

# 9
clf.score(X_train, y_train)

# 10
imports_corpus_test = []
num_sections_test = []
section_names_test = []
Ngram_features_list_test = []
y_test = []
for i in range(len(samples_test)):
    file = samples_test[i]
    try:
        NGram_features = get_NGram_features_from_sample(
            sample, K1_most_frequent_Ngrams_list
        )
        pe = pefile.PE(file)
        imports = get_imports(pe)
        n_sections = len(pe.sections)
        sec_names = get_section_names(pe)
        imports_corpus_test.append(imports)
        num_sections_test.append(n_sections)
        section_names_test.append(sec_names)
        Ngram_features_list_test.append(NGram_features)
        y_test.append(labels_test[i])
    except Exception as e:
        print(sample + ":")
        print(e)

# 11
imports_corpus_test_transformed =
imports_featurizer.transform(imports_corpus_test)
```

```
section_names_test_transformed =
section_names_featurizer.transform(section_names_test)
X_test = hstack(
    [
        Ngram_features_list_test,
        imports_corpus_test_transformed,
        section_names_test_transformed,
        csr_matrix(num_sections_test).transpose(),
    ]
)
print(clf.score(X_test, y_test))




directories_with_labels = [("Benign PE Samples 2", 0), ("Malicious PE
Samples 2", 1)]
list_of_samples = []
labels = []
for dataset_path, label in directories_with_labels:
    samples = [f for f in listdir(dataset_path)]
    for sample in samples:
        file_path = os.path.join(dataset_path, sample)
        list_of_samples.append(file_path)
        labels.append(label)

samples_test = list_of_samples
labels_test = labels

N = 2
Ngram_counts_all = collections.Counter([])
for sample in samples_train:
    Ngram_counts_all += binary_file_to_Ngram_counts(sample, N)
K1 = 100
K1_most_frequent_Ngrams = Ngram_counts_all.most_common(K1)
K1_most_frequent_Ngrams_list = [x[0] for x in
K1_most_frequent_Ngrams]

imports_corpus_test = []
num_sections_test = []
section_names_test = []
Ngram_features_list_test = []
y_test = []
for i in range(len(samples_test)):
    file = samples_test[i]
    try:
        NGram_features = get_NGram_features_from_sample(
            sample, K1_most_frequent_Ngrams_list
        )
        pe = pefile.PE(file)
        imports = get_imports(pe)
```

```
        n_sections = len(pe.sections)
        sec_names = get_section_names(pe)
        imports_corpus_test.append(imports)
        num_sections_test.append(n_sections)
        section_names_test.append(sec_names)
        Ngram_features_list_test.append(NGram_features)
        y_test.append(labels_test[i])
    except Exception as e:
        print(sample + ":")
        print(e)

# 11
imports_corpus_test_transformed =
imports_featurizer.transform(imports_corpus_test)
section_names_test_transformed =
section_names_featurizer.transform(section_names_test)
X_test = hstack(
    [
        Ngram_features_list_test,
        imports_corpus_test_transformed,
        section_names_test_transformed,
        csr_matrix(num_sections_test).transpose(),
    ]
)
print(clf.score(X_test, y_test))

from sklearn.metrics import accuracy_score, confusion_matrix

y_test_pred = clf.predict(X_test)
print(accuracy_score(y_test, y_test_pred))
print(confusion_matrix(y_test, y_test_pred))
print(y_test_pred)
```

4) Deep Learning Malware Detector

```
import numpy as np
from tqdm import tqdm


# 1
def embed_bytes(byte):
    binary_string = "{0:08b}".format(byte)
    vec = np.zeros(8)
    for i in range(8):
        if binary_string[i] == "1":
            vec[i] = float(1) / 16
    else:
        vec[i] = -float(1) / 16
```

```
    return vec


# 2
import os
from os import listdir

directories_with_labels = [("Benign PE Samples", 0),
("Malicious PE Samples", 1)]
list_of_samples = []
labels = []
for dataset_path, label in directories_with_labels:
    samples = [f for f in listdir(dataset_path)]
    for sample in samples:
        file_path = os.path.join(dataset_path, sample)
        list_of_samples.append(file_path)
        labels.append(label)


# 3
def read_file(file_path):
    """Reads in the binary sequence of a binary file."""
    with open(file_path, "rb") as binary_file:
        data = binary_file.read()
    return data


# 4
max_size = 15000
num_samples = len(list_of_samples)
X = np.zeros((num_samples, 8, max_size))
Y = np.asarray(labels)
file_num = 0
for file in tqdm(list_of_samples):
    sample_byte_sequence = read_file(file)
    for i in range(min(max_size, len(sample_byte_sequence))):
        X[file_num, :, i] =
embed_bytes(sample_byte_sequence[i])
    file_num += 1

print(X.shape)
"""
(425, 8, 15000)
"""

from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, Y, test_size=0.3, stratify=Y, random_state=11
)

# 5

from keras import optimizers

my_opt = optimizers.SGD(lr=0.01, decay=1e-5, nesterov=True)
my_opt2 = optimizers.Adam(lr=0.01, )
# 6
from keras import Input
from keras.layers import Conv1D, Activation, multiply,
GlobalMaxPool1D, Dense
from keras import Model
from keras.models import load_model

inputs = Input(shape=(8, max_size))
conv1 = Conv1D(kernel_size=(128), filters=32, strides=(128),
               padding='same')(inputs)
conv2 = Conv1D(kernel_size=(128), filters=32, strides=(128),
               padding='same')(inputs)
a = Activation('sigmoid', name='sigmoid')(conv2)
mul = multiply([conv1, a])
b = Activation('relu', name='relu')(mul)
p = GlobalMaxPool1D()(b)
d = Dense(16)(p)
predictions = Dense(1, activation='sigmoid')(d)
model = Model(inputs=inputs, outputs=predictions)

# 6
model.compile(optimizer=my_opt2, loss="binary_crossentropy",
metrics=["acc"])
print(model.summary())

model.fit(X_train, y_train, validation_data=(X_test, y_test),
epochs=14, batch_size=16)
cwd = os.getcwd()
model_path = os.path.join(cwd, "Model.h5")
model.save(model_path)
print(model.metrics_names)
print(model.evaluate(X_test, y_test))
NN = model.predict(X_test)
cm = np.zeros((2, 2))
for i in range(NN.shape[0]):
```

```
    pred = 0 if NN[i] <= 0.5 else 1
    real = y_test[i]
    cm[pred, real] += 1
print(cm)

model = load_model(model_path)

directories_with_labels = [("Benign PE Samples 2", 0),
("Malicious PE Samples 2", 1)]
list_of_samples = []
labels = []
for dataset_path, label in directories_with_labels:
    samples = [f for f in listdir(dataset_path)]
    for sample in samples:
        file_path = os.path.join(dataset_path, sample)
        list_of_samples.append(file_path)
        labels.append(label)

max_size = 15000
num_samples = len(list_of_samples)
X = np.zeros((num_samples, 8, max_size))
Y = np.asarray(labels)
file_num = 0
for file in tqdm(list_of_samples):
    sample_byte_sequence = read_file(file)
    for i in range(min(max_size, len(sample_byte_sequence))):
        X[file_num, :, i] =
embed_bytes(sample_byte_sequence[i])
    file_num += 1

X_test = X
y_test = Y

print(model.evaluate(X_test, y_test))

NN = model.predict(X_test)
cm = np.zeros((2, 2))
for i in range(NN.shape[0]):
    pred = 0 if NN[i] <= 0.5 else 1
    real = y_test[i]
    cm[pred, real] += 1
print(cm)
```