Ανοικτό Πανεπιστήμιο Κύπρου

Σχολή Θετικών και Εφαρμοσμένων Επιστημών

Μεταπτυχιακή Διατριβή
Στην Ασφάλεια Υπολογιστών και Δικτύων

Αντιμετώπιση Εσωτερικών Απειλών με χρήση Τεχνητής Νοημοσύνης

Ευθύμιος Παντελίδης

Επιβλέπων Καθηγητής
Δρ. Σταύρος Σιαηλής

Δεκέμβριος 2020

Ανοικτό Πανεπιστήμιο Κύπρου

Σχολή Θετικών και Εφαρμοσμένων Επιστημών

Αντιμετώπιση Εσωτερικών Απειλών με χρήση Τεχνητής Νοημοσύνης

Ευθύμιος Παντελίδης

Επιβλέπων Καθηγητής
Δρ. Σταύρος Σιαηλής

Η παρούσα μεταπτυχιακή διατριβή υποβλήθηκε
προς μερική εκπλήρωση των απαιτήσεων για απόκτηση

μεταπτυχιακού τίτλου σπουδών
στην Ασφάλεια Υπολογιστών και Δικτύων

από τη Σχολή Θετικών και Εφαρμοσμένων Επιστημών
του Ανοικτού Πανεπιστημίου Κύπρου

Δεκέμβριος 2020

Περίληψη

Οι εσωτερικές απειλές αποτελούν σήμερα έναν από τους μεγαλύτερους κινδύνους επιθέσεων στα εταιρικά δίκτυα. Παρόλα τα εγκατεστημένα περιμετρικά συστήματα ασφαλείας, ο κίνδυνος να επηρεαστούν αρνητικά η εμπιστευτικότητα, ακεραιότητα ή και η μη διαθεσιμότητα των πληροφοριών ή των πληροφοριών συστημάτων του οργανισμού παραμένει μεγάλος. Σύμφωνα με την Verizon (2019), το 57 τοις εκατό (57%) των παραβιάσεων σε έναν οργανισμό αφορά εσωτερικές απειλές. Η πλειονότητα των παραβιάσεων, ένα ποσοστό που φτάνει έως και το σαράντα τοις εκατό, χρειάζεται από μήνες έως και χρόνια να εντοπιστεί.

Σκοπός της διατριβής είναι η υλοποίηση δύο [2] αλγορίθμων Τεχνητής Νοημοσύνης για την αντιμετώπιση εσωτερικών απειλών χρησιμοποιώντας διαδεδομένο dataset. Στόχος είναι να μην απαιτούν ανθρώπινη παρέμβαση (unsupervised) και να αξιολογηθούν με βάση την αποτελεσματικότητα τους στον εντοπισμό εσωτερικών απειλών.

Οι αλγόριθμοι που επιλέξαμε είναι Τεχνητά Νευρωνικά Δίκτυα και χρησιμοποιούνται για εντοπισμό ανωμαλιών, Autoencoder, Variational Autoencoder, και υλοποιήθηκαν σε γλώσσα προγραμματισμού Python και περιβάλλον TensorFlow. Το dataset που χρησιμοποιήσαμε έχει δημιουργηθεί από το Software Engineering Institute (SEI) του Carnegie Mellon University Division. Είναι διαθέσιμο στους ερευνητές χωρίς περιορισμούς με σκοπό την αξιολόγηση των διάφορων αλγορίθμων στατιστικής και μηχανικής μάθησης.

Summary

Internal threats are currently one of the biggest attacks risks on corporate networks. Despite the perimeter security systems installed, the risk of adversely affecting the confidentiality, integrity or even availability of organization's data or information systems, remains high. According to Verizon (2019), 57 percent (57%) of violations in an organization involve internal threats. The majority of violations, up to forty percent (40%), usually take months or even years to be detected.

The aim of the dissertation is to implement two [2] Artificial Intelligence algorithms internal threats mitigation, using a widespread dataset. The aim is not to require human intervention (unsupervised) and to be evaluated based on their effectiveness of identifying internal threats.

The algorithms we have selected are Artificial Neural Networks and used for be able to detect outliers, Autoencoder and Variational Autoencoder using Python programming language and TensorFlow environment, pyOD. The dataset we used was created by Software Engineering Institute (SEI) of the Carnegie Mellon University Division. It is available for researchers without restrictions for the purpose of evaluating various statistical and machine learning algorithms.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω αρχικά την σύζυγο μου Γεωργία που με στήριξε σε όλη τη διάρκεια φοίτησης μου σε αυτό το πρόγραμμα σπουδών, τον γιο μου Γιώργο για την υπομονή που κάνει όταν δεν μπορώ να είμαι μαζί του, παρόλο που λόγω της ηλικίας του δεν μπορεί να κατανοήσει με τι ασχολούμαι.

Επίσης θα ήθελα να ευχαριστήσω τον καθηγητή μου, Δρ. Σταύρο Σιαηλή για την ευκαιρία που μου έδωσε να ασχοληθώ με αυτό το θέμα, την βοήθεια του και την υπομονή του.

Περιεχόμενα

Κεφάλαιο 1
Εισαγωγή

Οι εσωτερικές απειλές αποτελούν σήμερα έναν από τους μεγαλύτερους κινδύνους επιθέσεων στα εταιρικά δίκτυα. Παρόλα τα εγκατεστημένα περιμετρικά συστήματα ασφαλείας, ο κίνδυνος να επηρεαστούν αρνητικά η εμπιστευτικότητα, ακεραιότητα ή και η μη διαθεσιμότητα των πληροφοριών ή των πληροφοριών συστημάτων του οργανισμού παραμένει μεγάλος.

Εσωτερικός χρήστης μπορεί να είναι κάποιος νυν ή πρώην υπάλληλος, εργολάβος, ή άλλος επιχειρηματικός συνεργάτης, ο οποίος έχει ή είχε εξουσιοδοτημένη πρόσβαση στο δίκτυο, στο σύστημα ή στα δεδομένα ενός οργανισμού.

Ο όρος εσωτερική απειλή διαχωρίζεται σε δύο κατηγορίες. Μία κατηγορία είναι η λανθασμένη χρήση (misuse), κατά την οποία ο εσωτερικός χρήστης άθελα του πιθανώς λόγω έλλειψης σωστής ενημέρωσης και εκπαίδευσης μπορεί να προκαλέσει κάποια βλάβη ή και απώλεια πληροφοριών. Η δεύτερη κατηγορία εσωτερικών απειλών την οποία και θα εξετάσουμε είναι κακόβουλη χρήση κατά την οποία εσκεμμένα οι εσωτερικοί χρήστες έχουν σαν σκοπό να επηρεάσουν αρνητικά την εμπιστευτικότητα, ακεραιότητα ή τη διαθεσιμότητα των πληροφοριών ή των πληροφοριών συστημάτων του οργανισμού.

1.3 Ερευνητικά Ερωτήματα

(α) Οι αλγόριθμοι που επιλέξαμε μπορούν να ανιχνεύσουν απειλές χωρίς ανθρώπινη παρέμβαση;

(β) Τί ποσοστό αποτελεσματικότητας είχε ο κάθε αλγόριθμος και πως μας επηρεάζει το ποσοστό αναποτελεσματικότητας τους;

(γ) Η χρήση ενός αλγορίθμου χωρίς επίβλεψη είναι αρκετή για να φτάσουμε στο επιθυμητό αποτέλεσμα ή μήπως υπάρχει ανάγκη για χρήση συνδυασμού αλγορίθμων με/χωρίς ανθρώπινη παρέμβαση;

Κεφάλαιο 2
Ανασκόπηση Βιβλιογραφίας

2.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα ανατρέξουμε σε διάφορες μελέτες οι οποίες έχουν γίνει σε σχέση με το θέμα μας, τον εντοπισμό και την αντιμετώπιση των εσωτερικών απειλών μέσω διάφορων μεθόδων εποπτευόμενης και μη εποπτευόμενης μάθησης.

2.2 Προηγούμενες μελέτες

Οι εσωτερικές απειλές στην ασφάλεια πληροφοριών θεωρούνται από τα πιο πολύπλοκα προβλήματα στην ασφάλεια πληροφοριών. Σύμφωνα με τον Kuheli [3], όλες οι συναλλαγές πλέον είναι ηλεκτρονικές με αποτέλεσμα να υπάρχει αυξημένος αριθμός προσωπικών και ευαίσθητων δεδομένων. Οι εσωτερικές απειλές είναι φανερά αυξημένες τα τελευταία χρόνια και φτάνουν σε βαθμό κοντά στις εξωτερικές απειλές. Οι εξωτερικές απειλές απαιτούν περισσότερη έρευνα σε αντίθεση με κάποιον ο οποίος έχει άμεση γνώση και προσβασιμότητα στα εσωτερικά συστήματα. Ο εντοπισμός μιας εσωτερικής παραβίασης είναι δύσκολος στις περισσότερης περιπτώσεις και εντοπίζεται πολλές φορές μετά που θα ολοκληρωθεί. Πολλές παραβιάσεις

γίνονται από προσωπικό το οποίο έχει πρόσβαση σε ευαίσθητα δεδομένα λόγω του πειρασμού. Κατηγορίες των ανθρώπων πίσω από τις εσωτερικές απειλές συμπεριλαμβάνουν παραπονεμένους υπαλλήλους, προσωπικό το οποίο το προσέλαβε η εταιρεία που εργάζεται και το κάνει για προσωπικό όφελος και προσωπικό το οποίο δρα εκ μέρους κάποιου ανταγωνιστή (κατασκοπεία). Οι εσωτερικές απειλές μπορεί να έχουν συνέπειες ως προς την απώλεια εμπιστοσύνης από τους πελάτες, την κλοπή πνευματικής ιδιοκτησίας, σαμποτάζ και κατασκοπεία.

Οι Papadaki Maria και Shiaeles Stavros [4], αναφέρονται αναλύουν τις εσωτερικές απειλές και τις διαχωρίζουν σύμφωνα με τους Collins et al. 2016 [5] στις πιο κάτω κατηγορίες:

1. Σαμποτάζ πληροφοριακών συστημάτων (IT): η χρήση των πληροφοριακών συστημάτων από τρίτους, για να προκαλέσει συγκεκριμένη ζημιά σε ένα οργανισμό ή κάποιο άτομο.

2. Κλοπή πνευματικής ιδιοκτησίας (intellectual property) : χρήση πληροφοριακών συστημάτων από τρίτους για κλοπή πνευματικής ιδιοκτησίας από τον οργανισμό. Αυτή η κατηγορία περιλαμβάνει τη βιομηχανική κατασκοπεία που περιλαμβάνει και άτομα εκτός του οργανισμού.

3. Απάτη: η χρήση πληροφοριακών συστημάτων από τρίτους για μη εξουσιοδοτημένη τροποποίηση, προσθήκη ή διαγραφή δεδομένων (όχι προγραμμάτων ή συστημάτων) ενός οργανισμού για προσωπικό κέρδος ή κλοπή πληροφορίες που οδηγούν σε έγκλημα ταυτότητας (π.χ. κλοπή ταυτότητας ή απάτη με πιστωτική κάρτα).

4. Διάφορα: αφορά περιπτώσεις στις οποίες η δραστηριότητα του ατόμου δεν εμπίπτει σε μία από τις πιο πάνω κατηγορίες.

Σύμφωνα με τον Graves [5], η χρήση των υφιστάμενων μέτρων ασφαλείας για την αντιμετώπιση των εσωτερικών απειλών δεν θεωρείται αρκετή. Η χρήση των μεθόδων ανάπτυξης τεχνητής νοημοσύνης βοηθά στην στήριξη των μέτρων ασφαλείας. Η απεξάρτηση του από τα ‘signatures’ αλλά και η μελέτη της συμπεριφοράς δίνει την δυνατότητα να

εντοπιστούν συμπεριφορές οι οποίες δεν έπρεπε να υπάρχουν (ανωμαλίες). Τα υφιστάμενα μέσα για αντιμετώπιση εσωτερικών απειλών είναι τα SIEM, DLP και IPS. Τα συγκεκριμένα μέσα έχουν ως μειονέκτημα την αναγκαιότητα του ανθρώπινου παράγοντα. Οι ρυθμίσεις ανίχνευσης παραμένουν στατικές και δεν έχουν την δυνατότητα να εντοπίσουν καινούργιες συμπεριφορές. Machine learning είναι συνδυασμός της ανάλυσης δεδομένων και της αυτοματοποίησης της διαδικασίας με τη χρήση αλγορίθμων. Η χρήση του δεν υφίσταται μόνο για ανίχνευση απειλών αλλά και για ανάλυση συμπεριφοράς των χρηστών. Μειώνει τον χρόνο που χρειάζεται το προσωπικό για να αναλύσει τα δεδομένα τα οποία έχουν συλλέξει και είναι αντικειμενικό στον εντοπισμό εσωτερικών απειλών.

Οι V. Koutsouvelis, S. Shiaeles, B. Ghita και G. Bendiab [6], δημιουργήσαν μία μέθοδο πρόβλεψης και ανίχνευσης της συμπεριφοράς κακόβουλων χρηστών με τη χρήση Conventional Neural Network (CNN), την οποία υλοποίησαν με το πρόγραμμα Tensorflow της Google. Η τεχνική που χρησιμοποιήθηκε βασίστηκε στην εξόρυξη δεδομένων, στη οπτικοποίηση τους και τελικά στην χρήση του αλγόριθμου CNN μέσω μηχανικής μάθησης (machine learning), για την κατηγοριοποίηση της δραστηριότητας των υπό εξέταση χρηστών σε κακόβουλη και φυσιολογική. Οι τιμές της ακρίβειας εκπαίδευση και επικύρωσης (training and validation accuracy) προσέγγισαν ποσοστά 100% και 90% αντίστοιχα.

Οι Nicolaou Andreas, Shiaeles Stavros, και Savage Nick. 2020 [7], χρησιμοποίησαν Bio-Inspired models και συγκεκριμένα Swarm Intelligence με σκοπό να βελτιώσουν την απόδοση των μοντέλων μηχανικής μάθησης, με τη σωστή επιλογή χαρακτηριστικών (Feature Selection Optimization) . Έπειτα εισάγοντας τα χαρακτηριστικά (features) στον αλγόριθμο μηχανικής μάθησης LOF (Local Outlier Factor) προσέγγισαν πολύ ψηλά ποσοστά ανίχνευσης στο λόγος των ορθών θετικών προβλέψεων επί του συνόλου των θετικών προτύπων (recall).

Οι Li et al. [8] στην προσπάθεια τους να βοηθήσουν τα συστήματα ανίχνευσης απειλών (Intrusion Detection Systems), να συλλέξουν και να μάθουν εμπειρικά μεταξύ τους, χρησιμοποιήσανε τον αλγόριθμο K-Nearest Neighbor (KNN) για να εισαγάγουν αξίες ευαισθησίας εισβολής, σε Collaborative Intrusion Detection Networks (CIDNs). Ενσωμάτωσαν 200 συναγερμούς με τρία επίπεδα: υψηλό, μεσαίο και χαμηλό. Αρχικά πειραματίστηκαν με τους αλγόριθμους K-Nearest Neighbor (KNN), Back-propagation Neural Network (BNN) και Decision Tree (DT), επιλέγοντας τον KNN έχοντας καλύτερη απόδοση.

Οι Sarma et al. [9] χρησιμοποιήσαν και πάλι τον αλγόριθμο K-Nearest Neighbor (KNN), για να ταξινομήσουν τους χρήστες σε τέσσερις [4] κατηγορίες. Οι κατηγορίες ήταν νόμιμος, πιθανώς νόμιμος, πιθανώς μη νόμιμος και μη νόμιμος. Όταν ο χρήστης βρισκόταν να ανήκει σε μια από τις τρεις [3] κατηγορίες εκτός από το νόμιμος, το σύστημα παρέπεμπε τον χρήστη σε διαδικασία αναγνώρισης προσώπου, το οποίο ενεργούσε ως δεύτερο μέρος ελέγχου ταυτότητας.

Οι Choras και Kozik [10], χρησιμοποίησαν ένα βαθύ και επαναλαμβανόμενο μοντέλο νευρωνικού δικτύου (Deep RNN), ως χωρίς επίβλεψη προσέγγιση για τον εντοπισμό μην φυσιολογικών δραστηριοτήτων δικτύου, σε αρχεία καταγραφής συστήματος σε πραγματικό χρόνο. Ο σκοπός ήταν να αναπτύξουν μια προσέγγιση που μπορεί να αξιοποιήσει αποτελεσματικά την υψηλή ταχύτητα, της ανομοιογενής ροής των δεδομένων και αυτοματοποίηση της δημιουργίας αποτελεσμάτων με τη χρήση μειωμένου ανθρώπινου δυναμικού.

Οι Tuor et al [11] χρησιμοποίησαν μοντέλο Deep Neural Network (DNN) για την ανίχνευση ανωμαλιών, από μεγάλη συνεχή ροή καταγραφών δικτύου σε πραγματικό χρόνο. Τα πρωτογενή αρχεία καταγραφής εισήχθησαν στον εξαγωγέα χαρακτηριστικών (feature extractor), με ένα διάνυσμα ανά ημέρα για κάθε χρήστη. Τα διανύσματα στη συνέχεια τροφοδοτούσαν ένα νευρωνικό δίκτυο, το οποίο δημιουργούσε ένα δίκτυο ανά χρήστη. Αυτά τα δίκτυα χρησιμοποιούνταν για την πρόβλεψη του επόμενου διανύσματος για τον κάθε χρήστη, μαθαίνοντας πρώτα τη φυσιολογική συμπεριφορά του κάθε χρήστη. Στη συνέχεια, προβλέπανε την ανωμαλία κάνοντας χρήση του Long Short-Term Memory (LSTM) του RNN (Recurrent Neural Network). Σε σύγκριση που έγινε του μοντέλου σε σχέση με την αποτελεσματικότητα και απόδοση του, μαζί με μοντέλα αλγορίθμων SVM (Support Vector Machine), PCA (Principal Component Analysisis) και Isolation Forrest, κάνοντας χρήση του scikit-learn , αναφέρθηκε ότι έχει καλύτερη απόδοση από τα μοντέλα αλγορίθμων με τα οποία έγινε η σύγκριση.

Οι Lin, Zhong, Jia and Chen [12], χρησιμοποιήσαν μία υβριδική τύπου προσέγγιση για εντοπισμό εσωτερικών απειλών. Έκαναν χρήση του unsupervised αλγορίθμου DBN (Deep Belief Network) με σκοπό να εξάγουν χαρακτηριστικά των αρχείων καταγραφής από διάφορες πηγές. Το μοντέλο αλγορίθμου DBN βασίζεται στο multilayer RBM (Restricted Boltzmann Machine), έτσι έγινε χρήση OCSVM (one-class SVM) για να εκπαιδευτεί ο ταξινομητής με σκοπό να ταξινομήσει ένα μελλοντικό άγνωστο σύνολο δεδομένων. Η χρήση βαθιάς μάθησης (deep learning) έγινε με σκοπό να μειώσουν να μειώσουν την μεγάλη απώλεια δεδομένων που είχαν άλλα μοντέλα.

Ο Legg [13], προσπάθησε να βρει λύση στο πρόβλημα των εσωτερικών απειλών χρησιμοποιώντας πραγματικά δεδομένα οργανισμού. Ο οργανισμός παραχωρούσε 750,000 εγγραφές ανά ημέρα από τις οποίες 44,000 ήταν αυθεντικές και χρησιμοποιήθηκαν από το σύστημα. Η διάρκεια της ανάπτυξης του συστήματος ήταν για τριανταμία [31] μέρες σε δύο [2] διαφορετικές ώρες. Το σύστημα είχε μεγάλο αριθμό εσφαλμένων θετικών (false positive) αποτελεσμάτων, για τα οποία ο Legg δήλωσε ότι ήταν λόγω του μεγάλου αριθμού συμβάντων που δημιουργούντο καθημερινά από τους χρήστες. Το σύστημα είχε τη δυνατότητα να αναγνωρίσει κάποιον χρήστη ο οποίος είχε καταχωρηθεί ως πιθανή απειλή. Η ικανότητα όμως του συστήματος να αναγνωρίσει την εσωτερική απειλή εξαρτιόταν σε μεγάλο βαθμό στα δεδομένα που εισχωρούσαν στο σύστημα. Το σύστημα δεν μπορούσε να αντικαταστήσει εντελώς το ρόλο ενός αναλυτή αλλά να μειώσει σε μεγάλο βαθμό το εύρος των αποτελεσμάτων στην διερεύνηση ενός περιστατικού.

Οι Legg, Buckley, Goldsmith και Creese [14] παρουσίασαν ένα σύστημα το οποίο έχει τη δυνατότητα να κατασκευάσει το προφίλ του κάθε χρήστη, σε μορφή δέντρου με βάση τις ενέργειες του, για να μπορεί να χρησιμοποιηθεί για σύγκριση με άλλους χρήστες του ίδιου ρόλου. Τα προφίλ συγκρίνονται με σκοπό να βρεθεί η απόκλιση μεταξύ των προηγούμενων δραστηριοτήτων που έχουν καταγραφεί, διαπιστώνοντας εάν πρόκειται για ανωμαλία και συνεπώς πιθανή εσωτερική απειλή. Το μοντέλο PCA (Principal Component Analysis) χρησιμοποιείτο σε αυτά τα δεδομένα με σκοπό την αξιολόγηση του βαθμού της ανωμαλίας που εντοπίζανε.

2.3 Συμβολή της παρούσας μεταπτυχιακής διατριβής

Σκοπός της μεταπτυχιακής διατριβής είναι να προσεγγίσουμε το θέμα της ανίχνευσης εσωτερικών απειλών, εξετάζοντας την αποτελεσματικότητα μοντέλων Τεχνητής Νοημοσύνης. Η χρήση Τεχνητής Νοημοσύνης θα είναι αναγκαία στο σύντομο μέλλον διότι θα μπορεί να βοηθήσει στη πρόβλεψη επιθέσεων ενός οργανισμού.

Κεφάλαιο 3
Τεχνητή Νοημοσύνη

3.1 Εισαγωγή

Η Τεχνητή Νοημοσύνη (artificial intelligence), έχει πολλούς ορισμούς αλλά σύμφωνα με τον John McCarthy (1956), ορίζεται ως "Η επιστήμη και η μηχανική της δημιουργίας έξυπνων μηχανών, ιδιαίτερα έξυπνων υπολογιστικών προγραμμάτων". Βασικός γνώμονας της Τεχνητής Νοημοσύνης είναι η κατανόηση της ανθρώπινης νοημοσύνης και η δημιουργία συστημάτων και μηχανών που λειτουργούν με ευφυή και ανεξάρτητο τρόπο, με την έννοια ότι αντιλαμβάνονται το περιβάλλον τους και λειτουργούν με τέτοιο τρόπο ώστε να μεγιστοποιήσουν την πιθανότητα επίτευξης των στόχων τους.

Η Μηχανική Μάθηση (machine learning) είναι υποτομέας της επιστήμης των υπολογιστών, ο οποίος σύμφωνα με τον Arthur Samuel (1959), ορίζεται ως "δίνει στους ηλεκτρονικούς υπολογιστές την δυνατότητα να μαθαίνουν χωρίς να χρειάζονται απαραίτητα να προγραμματιστούν". Εντάσσεται στον κλάδο της Τεχνητής Νοημοσύνης και έχει ως γενικό στόχο την κατανόηση της δομής των δεδομένων και την μοντελοποίηση τους. Επικεντρώνεται στην κατασκευή και την μελέτη αλγορίθμων οι οποίοι μαθαίνοντας από πειραματικά δεδομένα, εκπαιδεύονται και μπορούν να μάθουν και να κάνουν προβλέψεις ή να πάρουν αποφάσεις με βάση τα δεδομένα.

Τα Τεχνητά Νευρωνικά Δίκτυα (artificial neural networks) είναι μία κατηγορία αλγορίθμων μηχανικής μάθησης, τα οποία είναι εμπνευσμένα από τη δομή και λειτουργία των βιολογικών νευρωνικών δικτύων.

Οι Αυτοκωδικοποιητές (autoencoders), είναι αλγόριθμοι μάθησης χωρίς επίβλεψη οι οποίοι ανήκουν στην κατηγορία των τεχνητών νευρωνικών δικτύων. Έχουν την ίδια δομή με ένα τεχνικό νευρωνικό δίκτυο και αποτελείται από τρία [3] στρώματα, το στρώμα εισόδου, το στρώμα εξόδου και κάποιον αριθμό στρωμάτων στο κρυφό ενδιάμεσο επίπεδο.

3.2 Μηχανική Μάθηση

Η Μηχανική Μάθηση (machine learning) είναι πεδίο του κλάδου Τεχνητής Νοημοσύνης, της επιστήμης των υπολογιστών. Ασχολείται με την κατασκευή και μελέτη αλγορίθμων που έχουν την δυνατότητα να μάθουν από τα δεδομένα εισόδου, για να παράγουν προβλέψεις. Σύμφωνα με τον Tom M. Mitchell [17] ο ορισμός της μηχανικής μάθησης είναι «Ένα πρόγραμμα υπολογιστή λέγεται ότι μαθαίνει από εμπειρία E ως προς μια κλάση εργασιών T και ένα μέτρο επίδοσης P, αν η επίδοσή του σε εργασίες της κλάσης T, όπως αποτιμάται από το μέτρο P, βελτιώνεται με την εμπειρία E».

Για την αποτελεσματικότητα των μοντέλων, σημαντικό ρόλο έχει το σύνολο των δεδομένων. Οι διαστάσεις τους, το πλήθος τους, ο αριθμός και η επιλογή των χαρακτηριστικών τους, ο θόρυβος που περιέχουν και ο αριθμός των ομάδων που ανήκουν είναι βασικές παράμετροι για την επιλογή του είδους του αλγόριθμου το οποίο θα χρησιμοποιηθεί για να εκπαιδευτεί το μοντέλο.

Υπάρχουν τρείς [3] βασικές κατηγορίες μάθησης.

1. Μάθηση με επίβλεψη (supervised learning)
2. Μάθηση χωρίς επίβλεψη (unsupervised learning)
3. Ενισχυτική μάθηση (reinforcement learning)

3.2.1 Μάθηση με επίβλεψη (supervised learning)

Κατά την μάθηση με επίβλεψη στόχος είναι ο χαρακτηρισμός των δεδομένων, σύμφωνα με κάποια δεδομένα εκπαίδευσης. Ο αλγόριθμος κατασκευάζει μια συνάρτηση που απεικονίζει δεδομένες εισόδους, σε γνωστές επιθυμητές εξόδους, με απώτερο στόχο τη γενίκευση της συνάρτησης αυτής και για εισόδους με άγνωστη έξοδο.

Πλεονέκτημα της μάθησης χωρίς επίβλεψη, είναι ότι μπορούμε άμεσα να αξιολογήσουμε τα αποτελέσματα των μοντέλων. Εφαρμογές των αλγορίθμων υπάρχουν σε προβλήματα ταξινόμησης (classification), πρόγνωσης (prediction) και διερμηνείας (interpretation)

3.2.2 Μάθηση χωρίς επίβλεψη (unsupervised learning)

Ο αλγόριθμος μάθησης προσπαθεί να ανακαλύψει τυχόν συσχετίσεις μεταξύ των στιγμιότυπών εισόδου με άγνωστη έξοδο προκειμένου να βρεθούν δομικοί σχηματισμοί τους. Εφαρμογές των αλγορίθμων υπάρχουν σε προβλήματα ανάλυσης συσχετισμών (association analysis) και ομαδοποίησης (clustering).

3.2.3 Ενισχυτική Μάθηση (reinforcement learning)

Ο αλγόριθμος μαθαίνει μια στρατηγική ενεργειών μέσα από την άμεση παρατήρηση του τρόπου λειτουργίας ενός δυναμικού περιβάλλοντος, χωρίς κάποιος να του υποδείξει το υλικό εκπαίδευσης. Εφαρμογή βρίσκει σε προβλήματα σχεδιασμού (planning) όπως είναι ένα παιχνίδι σκάκι ή οδήγηση ενός οχήματος.

3.3 Εντοπισμός ανωμαλιών

Ο εντοπισμός ανωμαλιών ή ακροτάτων τιμών (outlier detection) στοχεύει στην εύρεση προτύπων στα δεδομένα τα οποία δεν συμφωνούν με την τυπική «συμπεριφορά» του συνόλου δεδομένων. Η κρισιμότητα των ακροτάτων παρατηρήσεων έγκειται στο γεγονός ότι μπορούν να αποτελέσουν χρήσιμη πληροφορία για πολλές εφαρμογές. Οι ακραίες τιμές μπορούν να προκληθούν στα δεδομένα από ποικίλους λόγους, όπως είναι η κακόβουλη δράση, η απάτη, η εισβολή από χάκερ, ακόμη και τρομοκρατικές δραστηριότητες ή η ίδια η φύση των δεδομένων.

Η ανακάλυψη ακροτάτων τιμών σχετίζεται με το πρόβλημα της αφαίρεσης θορύβου (noise removal), αλλά διακρίνεται από αυτό στο γεγονός ότι ο θόρυβος είναι εξ' ορισμού ανεπιθύμητος ενώ μια ακρότατη παρατήρηση μπορεί να μεταφραστεί σε πολύτιμη πληροφορία. Ο ανεπιθύμητος θόρυβος ορίζεται ως ένα φαινόμενο των δεδομένων το οποίο δεν παρουσιάζει ενδιαφέρον στον αναλυτή, αλλά αντιθέτως μπορεί να μειώσει την ποιότητα του συνόλου δεδομένων και, ως εκ τούτου, γεννιέται η ανάγκη να αφαιρεθεί προτού εφαρμοστεί κάποια τεχνική εξόρυξης.

Μερικές κατηγορίες αλγορίθμων οι οποίοι χρησιμοποιούνται για την ανίχνευση ανωμαλιών είναι οι

1. Γραμμικού μοντέλου (linear model), όπως οι Principal Component Analysis (PCA), Minimum Covariance Determinant (MCD), One-Class Support Vector Machines (OCSVM), και άλλοι.

2. Αλγόριθμοι βασισμένοι στην εγγύτητα (proximity-based), όπως Local Outlier Factor (LOF), Connectivity-Based Outlier Factor (COF), k Nearest Neighbors (KNN), και άλλοι.

3. Πιθανοτικοί αλγόριθμοι (probabilistic), όπως Angle-Based Outlier Detection (ABOD), Copula-Based Outlier Detection (COPOD), Median Absolute Deviation (MAD), Stochastic Outlier Selection (SOS), και άλλοι.

4. Αλγόριθμοι συνόλων ακραίων τιμών (outliers ensembles), όπως Isolation Forest, Feature Bagging, Lightweight On-line Detector of Anomalies (LODA), και άλλοι.

5. Αλγόριθμοι νευρωνικών δικτύων (neural networks), όπως Autoencoders (AE), Variational Autoencoders (VAE), Single-Objective Generative Adversarial Active Learning (SO_GAAL), Multiple-Objective Generative Adversarial Active Learning (MO_GAAL), και άλλοι.

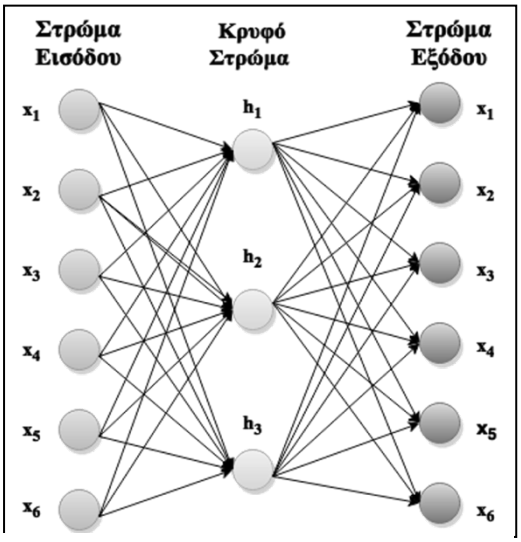
3.4 Αλγόριθμοι Νευρωνικών Δικτύων

3.4.1 Autoencoders

Οι Αυτοκωδικοποιητές (autoencoders), είναι αλγόριθμοι back-propagation χωρίς επίβλεψη, οι οποίοι ανήκουν στην κατηγορία των τεχνητών νευρωνικών δικτύων. Συμπιέζουν την είσοδο σε μία χαμηλότερη διάσταση και ανακατασκευάζουν την έξοδο τους μέσω μιας ενδιάμεσης αναπαράστασης.

Μερικά από τα βασικά χαρακτηριστικά που έχουν είναι, το στρώμα εισόδου έχει τον ίδιο αριθμό κόμβων με το στρώμα εξόδου [17], το κρυφό ενδιάμεσο επίπεδο έχει μικρότερο αριθμό κόμβων από αυτούς στο στρώμα εισόδου και σκοπός είναι παραχθεί στο στρώμα εξόδου ακριβώς το ίδιο διάνυσμα που είχε παραλειφθεί στο στρώμα εισόδου. Αποτελείται από δύο δίκτυα έναν κωδικοποιητή $E : X \leftarrow Z$ ο οποίος προβάλει την είσοδο σε έναν χώρο προβολής Z και έναν αποκωδικοποιητή $D : Z \leftarrow X$ ο οποίος προσπαθεί να ανακατασκευάσει την είσοδο. Το κάθε δίκτυο αποτελεί ένα νευρωνικό δίκτυο με m κρυφά επίπεδα και ο κωδικοποιητής προβάλει την είσοδο σε ένα χώρο προβολής Z συνήθως μικρότερης διάστασης από αυτή της εισόδου X.

Είναι σημαντικό να σχεδιάζεται με τρόπο, ώστε να μην μπορεί να ανακατασκευάζει ακριβώς την είσοδο στην έξοδο στο σύνολο των δεδομένων εκπαίδευσης. Για να είναι αυτό εφικτό, ο χώρος προβολής Z του κωδικοποιητή πρέπει να έχει μικρότερη διάσταση. Με αυτόν τον τρόπο στην προσπάθειά του το δίκτυο να ανακατασκευάσει την είσοδο μαθαίνει χρήσιμες σχέσεις μεταξύ των χαρακτηριστικών του συνόλου των δεδομένων.



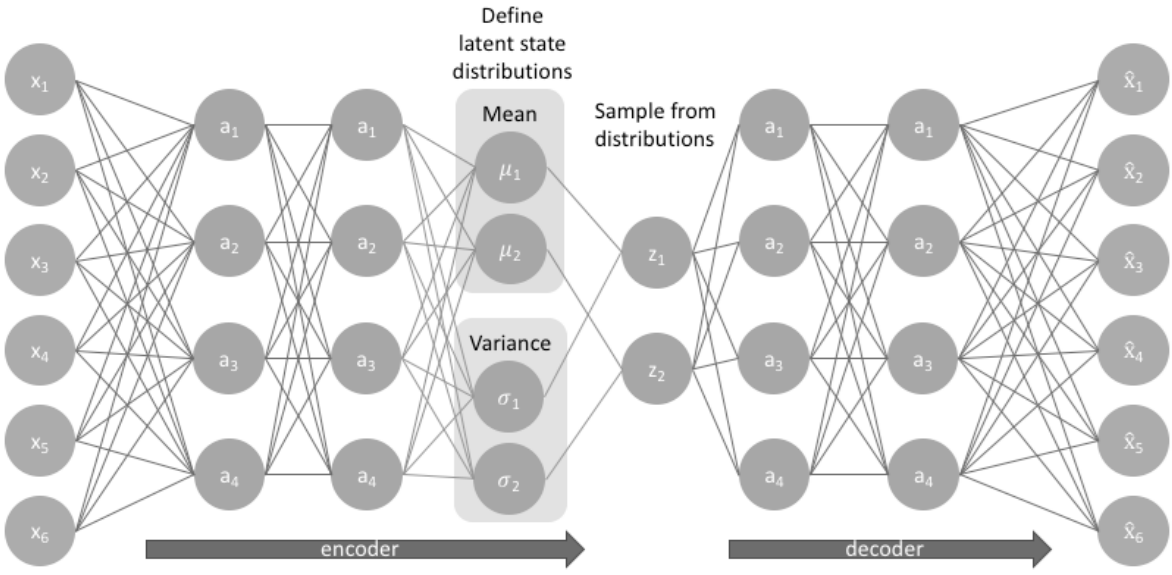
Εικόνα 1: Διάγραμμα Autoencoder

3.4.2 Variational AutoEncoders (VAE)

Οι variational αυτοκωδικοποιητές είναι τεχνητά νευρωνικά δίκτυα, όμοια στην αρχιτεκτονική με τους αυτοκωδικοποιητές, διαφέρουν όμως από αυτούς στη μαθηματική μοντελοποίηση. Πρόκειται για γεννητικά μοντέλα, τα οποία προσπαθούν να ανακαλύψουν την εσωτερική δομή των δεδομένων εισόδου , με σκοπό να παράγουν παρόμοια δεδομένα.

Δίνουν μεγαλύτερη έμφαση στη δημιουργία τεχνητών δεδομένων παρά στην εκμάθηση χαρακτηριστικών για σκοπούς προ εκπαίδευσης και δεν παράγουν απευθείας το coding layer, αλλά παράγουν κωδικοποιήσεις για τη μέση τιμή (μ) και την τυπική απόκλιση (σ). [18] [19]

Από τις πληροφορίες που καταλήγουν στον "decoder", γίνεται δειγματοληψία στη συνέχεια από μία από τις συνηθισμένες κατανομές (Gaussian, ομοιόμορφη) μέσης τιμής μ και τυπικής απόκλισης σ. Αυτό βοηθάει στη περιπλοκότητα των δεδομένων εισόδου διότι η κατανομή των δεδομένων μετασχηματίζεται σε ένα σύνολο από σημεία που μπορούν να διαχωριστούν και να γίνει δειγματοληψία εύκολα, ενώ η δημιουργία νέων δεδομένων απαιτεί μόνο τη δειγματοληψία coding με τα δοσμένα στατιστικά χαρακτηριστικά.

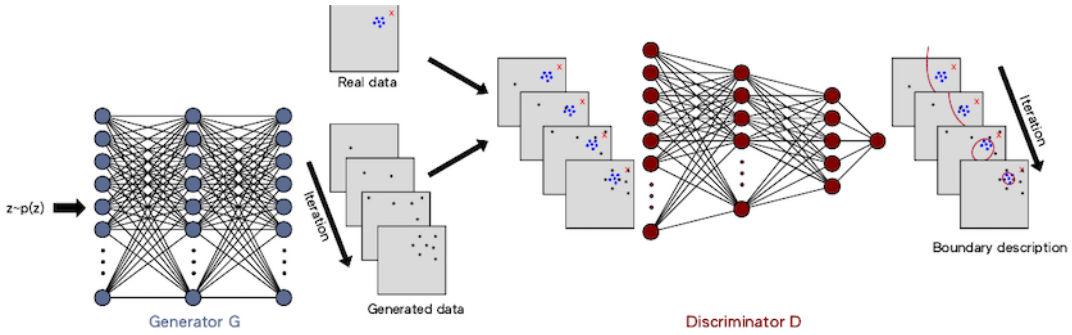


Εικόνα 2: Διάγραμμα VAE

3.4.3 SO-GAAL και MO-GAAL

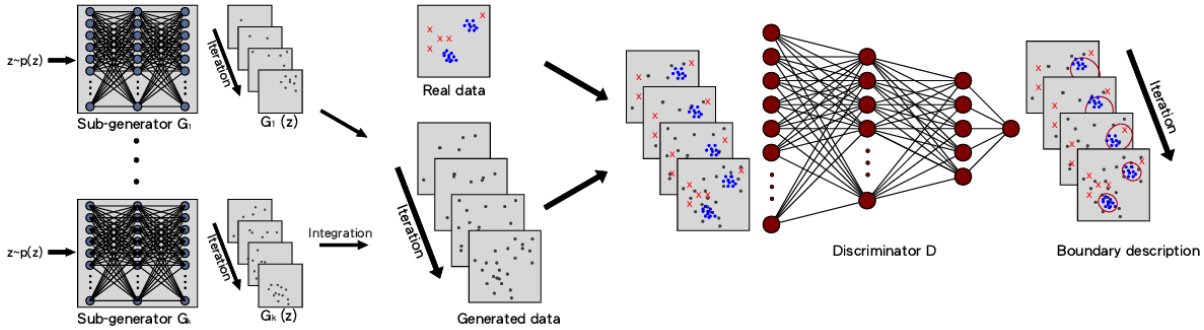
Οι Single-Objective Generative Adversarial Active Learning (SO-GAAL) και οι Multiple-Objective Generative Adversarial Active Learning (MO-GAAL), είναι αλγόριθμοι που χρησιμοποιούνται για εντοπισμό ανωμαλιών και ανήκουν στην κατηγορία νευρωνικών δικτύων[20]. Έχει την δυνατότητα να εντοπίσει πιθανές ανωμαλίες με βάση το παιχνίδι 'mini-max', μεταξύ ενός generator G και ενός discriminator D.

Η διαδικασία ανίχνευσης ανωμαλιών του αλγόριθμου SO-GAAL, έχει ως εξής. Στην αρχή του training, ο generator G δεν μπορεί να δημιουργήσει επαρκή αριθμό πιθανών ακραίων τιμών γύρω από τα πραγματικά δεδομένα, γεγονός που αναγκάζει τον discriminator D να περιγράψει ένα τραχύ όριο. Όμως, μετά από αρκετές επαναλήψεις, generator G μαθαίνει σταδιακά τον μηχανισμό δημιουργίας πραγματικών δεδομένων, με βάση το παιχνίδι mini-max μεταξύ G και D, και δημιουργεί έναν αυξανόμενο αριθμό ενημερωτικών πιθανών ακραίων τιμών. Ως αποτέλεσμα, ο διαχωριστής D μπορεί να εντοπίσει ένα σωστό όριο γύρω από τα συγκεντρωμένα σημεία δεδομένων.



Εικόνα 3: Διαδικασία εντοπισμού ανωμαλιών του SO-GAAL

Ο αλγόριθμος MO-GAAL αποτελείται από k sub-generators G1: k, οι οποίοι δημιουργούν διαφορετικές κατανομές αναφοράς για διαφορετικά υποσύνολα δεδομένων. Παρόλο που υπάρχουν δύο ομάδες στο σύνολο δεδομένων, οι ενσωματωμένοι δυνητικοί ακροδέκτες εξακολουθούν να παρέχουν μια λογική κατανομή αναφοράς για να βοηθήσουν τον discriminator D να περιγράψει ένα σωστό όριο γύρω από τα συγκεντρωμένα σημεία δεδομένων. Με αυτόν τον τρόπο επεκτείνεται η δομή του SO-GAAL από έναν generator σε πολλαπλούς οι οποίοι έχουν διαφορετικούς στόχους και μπορεί να δημιουργηθεί μια λογική κατανομή αναφοράς για όλο το σύνολο δεδομένων.



Εικόνα 4: Διαδικασία εντοπισμού ανωμαλιών του MO-GAAL

Κεφάλαιο 4
Προτεινόμενη Μεθοδολογία

4.1 Εισαγωγή

Η προσέγγιση μας σε σχέση με τον εντοπισμό εσωτερικών απειλών, με τη χρήση Τεχνητής Νοημοσύνης, έχει να κάνει με την υλοποίηση τεσσάρων αλγορίθμων και τον έλεγχο της αποτελεσματικότητας τους σε ένα dataset.

Στόχος μας είναι να εκπαιδεύσουμε τους αλγόριθμους ανίχνευσης ανωμαλιών, Autoencoder, Variational Autoencoder με τη χρήση συνθετικών δεδομένων τα οποία τα οποία είναι δημιουργημένα για ερευνητικούς σκοπούς μέσω του Cert Division και άλλους φορείς.

4.2 Περιγραφή εργαλείων και βιβλιοθηκών

Για τον σκοπό της Μεταπτυχιακής διατριβής, κάναμε χρήση της γλώσσας προγραμματισμού Python σε περιβάλλον Anaconda. Πιο κάτω κάνουμε μια σύντομη αναφορά σε σχέση με τα εργαλεία και τις βιβλιοθήκες που χρησιμοποιήσαμε για σκοπούς της διατριβής.

1. Python

Η Python είναι μια γλώσσα προγραμματισμού με απλό συντακτικό, εξαιρετική αναγνωσιμότητα, φορητότητα (portability) και μοντέρνα χαρακτηριστικά που την κάνουν κατάλληλη ως πρώτη γλώσσα προγραμματισμού. Θεωρείται high level language. Και είναι διερμηνευόμενη (interpreted) γλώσσα προγραμματισμού. Έχει τη δυνατότητα

να χρησιμοποιηθεί τόσο για τη δημιουργία σεναρίων εντολών όσο και για τη γρήγορη ανάπτυξη ολοκληρωμένων εφαρμογών σε διάφορες περιοχές ενδιαφέροντος (διαχείριση συστήματος υπολογιστή, ανάπτυξη εφαρμογών Διαδικτύου, επεξεργασία αρχείων κειμένου, επιστημονικές εφαρμογές, εκπαίδευση, ανάπτυξη παιχνιδιών, κ.λπ.) και στις περισσότερες πλατφόρμες υπολογιστών και Λειτουργικών Συστημάτων (Windows, Unix, Linux, Mac OS X, κ.λπ.).

2. Anaconda

Το Anaconda είναι μια δωρεάν υπό όρους διανομή ανοιχτού κώδικα των γλωσσών προγραμματισμού Python και R για επιστημονικούς υπολογιστές, που στοχεύει στην απλοποίηση της διαχείρισης και της ανάπτυξης πακέτων. Η διανομή περιλαμβάνει πακέτα επιστήμης δεδομένων κατάλληλα για Windows, Linux και macOS. Περιέχει ενα πακέτο ανοιχτού κώδικα και σύστημα διαχείρισης περιβάλλοντος που ονομάζεται Conda, το οποίο διευκολύνει την εγκατάσταση ή/και ενημέρωση πακέτων και τη δημιουργία ή/και φόρτωση περιβαλλόντων.

3. Jupiter Notebook

Το Jupiter Notebook είναι ένα web application, το οποίο σου δίνει την δυνατότητα να δημιουργήσεις και να χρησιμοποιήσεις αρχεία, τα οποία περιέχουν κώδικα προγραμματισμού, κείμενο, εικόνες, πίνακες και άλλα. Είναι λογισμικού ανοιχτού κώδικα και δημιουργήθηκε από το μη κερδοσκοπικό οργανισμό Project Jupiter το 2014 [21].

4. Tensorflow

Το TensorFlow [22] είναι μια ολοκληρωμένη πλατφόρμα ανοιχτού κώδικα για μηχανική μάθηση. Διαθέτει ένα ολοκληρωμένο, ευέλικτο οικοσύστημα εργαλείων, βιβλιοθηκών και πόρων της κοινότητας που επιτρέπει στους ερευνητές να προωθήσουν την τελευταία λέξη της τεχνολογίας στην Μηχανική μάθηση και οι προγραμματιστές να δημιουργούν και να αναπτύσσουν εύκολα εφαρμογές που υποστηρίζονται από Μηχανική μάθηση.

5. Keras

Το Keras είναι μια βιβλιοθήκη ανοιχτού κώδικα που παρέχει μια διεπαφή Python για τεχνητά νευρωνικά δίκτυα και για τη βιβλιοθήκη TensorFlow. Είναι high level API του TensorFlow 2.0 μια προσιτή, πολύ παραγωγική διεπαφή για την επίλυση προβλημάτων μηχανικής μάθησης, με έμφαση στη σύγχρονη βαθιά μάθηση. Παρέχει βασικές αφαιρέσεις και δομικά στοιχεία για την ανάπτυξη και αποστολή λύσεων μηχανικής μάθησης με υψηλή ταχύτητα επανάληψης.

6. Pandas

Τα Pandas είναι μια βιβλιοθήκη λογισμικού γραμμένη για τη γλώσσα προγραμματισμού Python για χειρισμό και ανάλυση δεδομένων. Ειδικότερα, προσφέρει δομές και λειτουργίες δεδομένων για χειρισμό αριθμητικών πινάκων και χρονοσειρών. Για σκοπούς διατριβής χρησιμοποιήσαμε αυτή τη βιβλιοθήκη για χειρισμό και ανάλυση του dataset.

7. pyOD

Το PyOD είναι μια εργαλειοθήκη Python ανοιχτού κώδικα για την εκτέλεση ανίχνευσης εξωτερικών παραμέτρων (outliers) σε πολλαπλές παραλλαγές δεδομένων. Παρέχει πρόσβαση σε ένα ευρύ φάσμα αλγορίθμων ανίχνευσης ανωμαλιών συμπεριλαμβανομένων και των πιο πρόσφατων προσεγγίσεων που βασίζονται σε νευρωνικά δίκτυα, κάτω από ένα ενιαίο, καλά τεκμηριωμένο API σχεδιασμένο για χρήση τόσο από επαγγελματίες όσο και από ερευνητές. Το pyOD είναι συμβατό με Python 2 και 3 και μπορεί να εγκατασταθεί μέσω του Python Package Index (PyPI). Για σκοπούς της διατριβής η εγκατάσταση έγινε μέσω του Anaconda, για το οποίο βρίσκεται διαθέσιμο. Μέσω του pyOD έχουμε τη δυνατότητα να πειραματιστούμε με τους πιο κάτω αλγορίθμους ή ακόμα να συνδυάσουμε μερικούς από αυτούς για καλύτερα αποτελέσματα στον εντοπισμό ανωμαλιών.

Type	Abbrev	Algorithm	Year
Linear Model	PCA	Principal Component Analysis (the sum of weighted projected distances to the eigenvector hyperplanes)	2003
Linear Model	MCD	Minimum Covariance Determinant (use the mahalanobis distances as the outlier scores)	1999
Linear Model	OCSVM	One-Class Support Vector Machines	2001
Linear Model	LMDD	Deviation-based Outlier Detection (LMDD)	1996
Proximity-Based	LOF	Local Outlier Factor	2000
Proximity-Based	COF	Connectivity-Based Outlier Factor	2002
Proximity-Based	CBLOF	Clustering-Based Local Outlier Factor	2003
Proximity-Based	LOCI	LOCI: Fast outlier detection using the local correlation integral	2003
Proximity-Based	HBOS	Histogram-based Outlier Score	2012
Proximity-Based	kNN	k Nearest Neighbors (use the distance to the kth nearest neighbor as the outlier score)	2000

Proximity-Based	AvgKNN	Average kNN (use the average distance to k nearest neighbors as the outlier score)	2002
Proximity-Based	MedKNN	Median kNN (use the median distance to k nearest neighbors as the outlier score)	2002
Proximity-Based	SOD	Subspace Outlier Detection	2009
Probabilistic	ABOD	Angle-Based Outlier Detection	2008
Probabilistic	COPOD	COPOD: Copula-Based Outlier Detection	2020
Probabilistic	FastABOD	Fast Angle-Based Outlier Detection using approximation	2008
Probabilistic	MAD	Median Absolute Deviation (MAD)	1993
Probabilistic	SOS	Stochastic Outlier Selection	2012
Outlier Ensembles	IForest	Isolation Forest	2008
Outlier Ensembles		Feature Bagging	2005
Outlier Ensembles	LSCP	LSCP: Locally Selective Combination of Parallel Outlier Ensembles	2019
Outlier Ensembles	XGBOD	Extreme Boosting Based Outlier Detection **(Supervised)**	2018
Outlier	LODA	Lightweight On-line Detector of Anomalies	2016

Ensembles			
Neural Networks	AutoEncoder	Fully connected AutoEncoder (use reconstruction error as the outlier score)	
Neural Networks	VAE	Variational AutoEncoder (use reconstruction error as the outlier score)	2013
Neural Networks	Beta-VAE	Variational AutoEncoder (all customized loss term by varying gamma and capacity)	2018
Neural Networks	SO_GAAL	Single-Objective Generative Adversarial Active Learning	2019
Neural Networks	MO_GAAL	Multiple-Objective Generative Adversarial Active Learning	2019

8. Scikit-learn

Το Scikit-learn είναι μια δωρεάν βιβλιοθήκη για τη γλώσσα προγραμματισμού Python η οποία χρησιμοποιείται για την δημιουργία μοντέλων μηχανικής μάθησης. Συμπεριλαμβάνει μεγάλο αριθμό έτοιμων αλγορίθμων οι οποίοι γίνονται διαθέσιμοι μέσω modules.

9. Numpy

NumPy (Numerical Python) είναι μια βιβλιοθήκη ανοιχτού κώδικα για τη γλώσσα προγραμματισμού Python, προσθέτοντας υποστήριξη για μεγάλες, πολυδιάστατες

συστοιχίες (arrays), και πίνακες (matrix) μαζί με μια μεγάλη συλλογή μαθηματικών συναρτήσεων υψηλού επιπέδου για λειτουργία σε αυτές τις συστοιχίες.

4.3 Περιγραφή του Dataset

Το συνθετικό σύνολο δεδομένων (dataset) που χρησιμοποιήθηκε για τον σκοπό της μεταπτυχιακής διατριβής, έχει δημιουργηθεί από το Software Engineering Institute (SEI) του Carnegie Mellon University Division. Είναι διαθέσιμο στους ερευνητές χωρίς περιορισμούς με σκοπό την αξιολόγηση των διάφορων αλγορίθμων στατιστικής και μηχανικής μάθησης.

Τα σύνολα δεδομένων που δημιουργήθηκαν, κοινοποιήθηκαν σε μια μεγάλη κοινότητα ερευνητών που ασχολούνται με το πρόγραμμα DARPA ADAMS, το οποίο ασχολείται με την ανάπτυξη τεχνικών για την ανίχνευση εσωτερικών απειλών. Σύμφωνα με τον Glasser, J. and Lindauer, B. (2013) ανάλυση των δεδομένων επιβεβαίωσε την πεποίθησή τους ότι απλά μοντέλα τοπολογίας κοινωνικού δικτύου, μαζί με μοντέλα δραστηριότητας χρηστών, μπορούν να βοηθήσουν με επιτυχία στην ανάπτυξη αυτών των τεχνολογιών. Η εμπειρία τους έδειξε επίσης ότι οι διερευνητικές χρήσεις συνθετικών δεδομένων δεν είναι αρκετές, και θα πρέπει να προχωρήσουν με τη δημιουργία πιο έγκυρων συνθετικών δεδομένων στο μέλλον[23].

Το συνθετικό dataset συμπεριλαμβάνει δέκα [10] διαφορετικά dataset (r1,r2, r3.1, r3.2, r4.1, r4.2, r5.1, r5.2, r6.1, r6.2) με βάση τη πολυπλοκότητα τους.

4.3.1 Ανάλυση του Dataset

Για την υλοποίηση των μοντέλων μας επιλέξαμε το dataset r4.2 το οποίο περιέχει αρχεία καταγραφής που σχετίζονται με χρήση αφαιρούμενων δίσκων (USB), χρήση ηλεκτρονικού ταχυδρομείου (email), πρόσβαση σε ιστοσελίδες (web), σύνδεση και

αποσύνδεση στους υπολογιστές του και ψυχομετρικά χαρακτηριστικά. Το dataset r4.2 είναι χωρισμένο στα πιο κάτω αρχεία στον Πίνακα 2.

Όνομα Αρχείου	**Περιγραφή**
logon.csv	Καταγράφει την είσοδο και έξοδο του χρήστη
device.csv	Καταγράφει την σύνδεση και αποσύνδεση USB
http.csv	Καταγράφει τη χρήση διαδικτύου
email.csv	Καταγράφει τη χρήση ηλεκτρονικού ταχυδρομείου
file.csv	Καταγράφει την αποθήκευση αρχείων σε αφαιρούμενες συσκευές
psychometric.csv	Ψυχομετρικά αποτελέσματα
ldap.csv	Χρήστες του οργανισμού και οι ρόλοι τους

Πίνακας 2. Πληροφορίες αρχείων csv

Το συνθετικό dataset 4.2 περιέχει τρία[3] σενάρια εσωτερικής απειλής:

1. Ένας χρήστης ο οποίος δεν εργαζόταν σε ώρες εκτός ωραρίου εργασίας ή δεν χρησιμοποιούσε αφαιρούμενους δίσκους παλαιότερα, άρχισε να εισέρχεται στο πληροφοριακό σύστημα του οργανισμού και να χρησιμοποιεί αφαιρούμενους δίσκους για να ανεβάσει δεδομένα στο wikileaks.org

2. Ένας υπάλληλος αρχίζει να ψάχνει ιστοσελίδες εύρεσης εργασίας και τελικά βρίσκει μία νέα θέση εργασίας σε ανταγωνίστρια εταιρία. Πριν φύγει από την τωρινή εργασία, αρχίζει να χρησιμοποιεί μία φορητή συσκευή αποθήκευσης, με μεγαλύτερη συχνότητα από ότι παλαιότερα για να κλέψει δεδομένα.

3. Ένας διαχειριστής συστήματος γίνεται δυσαρεστημένος. Κατεβάζει ένα λογισμικό καταγραφής πληκτρολογίου και το εγκαθιστά στον υπολογιστή του προϊσταμένου του. Την επόμενη μέρα εισέρχεται στον υπολογιστή και στέλνει

μαζικό μήνυμα ηλεκτρονικού ταχυδρομείου στο προσωπικό του οργανισμού για να δημιουργήσει πανικό.

Επιλέξαμε αυτή την έκδοση γνωρίζοντας πως υπάρχουν πολλαπλοί χρήστες που πραγματοποιούν το κάθε σενάριο.

4.3.2 Εισαγωγή και επεξεργασία δεδομένων

Τα βήματα που ακολουθήσαμε για την προετοιμασία του CERT dataset r4.2 για να το χρησιμοποιήσουμε στο πείραμα είναι τα ακόλουθα. Στόχος μας είναι να εισάγουμε τις καταγραφές (logs) και να τις μετατρέψουμε σε μορφή η οποία θα μπορούν να τροφοδοτηθούν στα μοντέλα μας. [24]

1. Αρχικά λόγω του μεγάλου όγκου των αρχείων και τις υψηλές απαιτήσεις μνήμης δημιουργήσαμε δείγματα των 50000 καταχωρήσεων ανά στήλη.

```
#create samples files for our experiment using 50000 rows per csv file

#logins
sample_Logon = pd.read_csv('/Volumes/CERT-DATA/r4.2/logon.csv',low_memory=False, nrows=50000)
sample_Logon.to_csv('/Volumes/CERT-DATA/r4.2/sample-logon.csv', index=False)
#devices
sample_Device = pd.read_csv('/Volumes/CERT-DATA/r4.2/device.csv', low_memory=False, nrows=50000)
sample_Device.to_csv('/Volumes/CERT-DATA/r4.2/sample-device.csv', index=False)
#files
sample_File = pd.read_csv('/Volumes/CERT-DATA/r4.2/file.csv', low_memory=False, nrows=50000)
sample_File.to_csv('/Volumes/CERT-DATA/r4.2/sample-file.csv', index=False)
#http
sample_http = pd.read_csv('/Volumes/CERT-DATA/r4.2/http.csv', low_memory=False, nrows=50000)
sample_http.to_csv('/Volumes/CERT-DATA/r4.2/sample-http.csv', index=False)
#email
sample_email = pd.read_csv('/Volumes/CERT-DATA/r4.2/email.csv', low_memory=False, nrows=50000)
sample_email.to_csv('/Volumes/CERT-DATA/r4.2/sample-email.csv', index=False)
```

Εικόνα 6: Create sample files

2. Δημιουργήσαμε τα ακόλουθα panda dataframes με τις στήλες (features) τις οποίες θα χρειαστούν επεξεργασία για χρησιμοποιήσουμε στους αλγόριθμους μας.

1. logins
2. devices

3. files
4. http
5. email

6. Δημιουργία ενός ενιαίου dataframe το οποίο περιέχει τα στοιχεία από τα υπόλοιπα dataframes.

7. Μετατροπή όλων των features που θα χρησιμοποιήσουμε στο μοντέλο μας σε μορφή (type) integer ή float.

8. Χρησιμοποιήσαμε One Hot Encoding για να μετατρέψουμε τις καταχωρήσεις σε binary 1 και 0

9. Δημιουργήσαμε feature 'insider' και καταχωρήσαμε τους χρήστες οι οποίοι θεωρούνται true-positive με βάση τα σενάρια του dataset 4.2.

10. Δημιουργήσαμε καινούργιο αρχείο μορφής csv με καταχωρήσεις από 1000 γραμμές χρησιμοποιώντας τις στήλες *"day", "time", "Logon", "Logoff", "Connect", "Disconnect", "email", "file", "http",* το οποίο θα κάνουμε εισαγωγή κατά την εκτέλεση των μοντέλων AE και VAE.

    Ο κώδικας προγραμματισμού σε γλώσσα Python παρουσιάζεται στο Παράρτημα Α.1

4.4 Υλοποίηση αλγορίθμων για εντοπισμό ανωμαλιών

Σε αυτό το μέρος της διατριβής θα περιγράψουμε την υλοποίηση των αλγορίθμων autoencoder (AE) και variational autoencoder (VAE).

4.4.1 Autoencoder (AE)

Για την υλοποίηση του autoencoder, δημιουργούμε ένα επίπεδο εισόδου, ένα επίπεδο εξόδου και κρυφά επίπεδα για την κωδικοποίηση και την αποκωδικοποίηση. Επιπλέον καθορίσαμε το activation function στα επίπεδα (tanh, relu) και το layer dense.

```
## input layer
input_layer = Input(shape=(X.shape[1],))

## encoding part
encoded = Dense(100, activation='tanh', activity_regularizer=regularizers.l1(10e-5))(input_layer)
encoded = Dense(50, activation='relu')(encoded)

## decoding part
decoded = Dense(50, activation='tanh')(encoded)
decoded = Dense(100, activation='tanh')(decoded)

## output layer
output_layer = Dense(X.shape[1], activation='relu')(decoded)
```

Εικόνα 7: Autoencoder

Ύστερα από δοκιμές καθορίσαμε ως optimizer τον “adadelta” και loss function “mse”. Ο autoencoder έτρεξε για 30 epochs με batch size 256.

```
autoencoder.fit(x_norm, x_norm,
                batch_size = 256, epochs = 30,
                shuffle = True, validation_split = 0.20);
```

```
Train on 556 samples, validate on 140 samples
Epoch 1/30
556/556 [==============================] - 4s 8ms/step - loss: 0.5640 - val_loss: 0.35
31
Epoch 2/30
556/556 [==============================] - 0s 106us/step - loss: 0.4737 - val_loss: 0.
2944
Epoch 3/30
556/556 [==============================] - 0s 113us/step - loss: 0.4259 - val_loss: 0.
2810
Epoch 4/30
556/556 [==============================] - 0s 117us/step - loss: 0.4055 - val_loss: 0.
2693
Epoch 5/30
556/556 [==============================] - 0s 152us/step - loss: 0.3876 - val_loss: 0.
2589
Epoch 6/30
556/556 [==============================] - 0s 105us/step - loss: 0.3714 - val_loss: 0.
```

Εικόνα 8: Compiling Autoencoder

Ο κώδικας προγραμματισμού του autoencoder σε γλώσσα Python παρουσιάζεται στο Παράρτημα Α.2

4.4.2 Variational Autoencoder (AE)

Η δημιουργία ενός variational autoencoder αποτελείται από τη δημιουργία ενός νευρωνικού δικτύου για τον κωδικοποιητή, ενός νευρωνικού δικτύου για τον αποκωδικοποιητή και του VAE ο οποίος τα συνδέει. Για τη δημιουργία του αλγορίθμου χρησιμοποιήσαμε τις παραμέτρους στην Εικόνα 9, με βάση δοκιμής και σύμφωνα με τις οδηγίες του Keras.org [18, 26, 27]. Ο optimizer που χρησιμοποιήσαμε για τον VAE είναι ο "Nadam"

```
Model: "encoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential_10 (Sequential)   (None, 8)                 498
_________________________________________________________________
dense_43 (Dense)             (None, 5)                 45
_________________________________________________________________
multivariate_normal_tri_l_5  ((None, 2), (None, 2))    0
=================================================================
Total params: 543
Trainable params: 543
Non-trainable params: 0
_________________________________________________________________
Model: "decoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential_11 (Sequential)   (None, 20)                334
_________________________________________________________________
dense_47 (Dense)             (None, 18)                378
_________________________________________________________________
independent_normal_5 (Indepe ((None, 9), (None, 9))    0
=================================================================
Total params: 712
Trainable params: 712
Non-trainable params: 0
_________________________________________________________________
Model: "vae_mlp"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
encoder_input (InputLayer)   [(None, 9)]               0
_________________________________________________________________
sequential_10 (Sequential)   (None, 8)                 498
_________________________________________________________________
dense_43 (Dense)             (None, 5)                 45
_________________________________________________________________
multivariate_normal_tri_l_5  ((None, 2), (None, 2))    0
_________________________________________________________________
decoder (Sequential)         (None, 9)                 712
=================================================================
Total params: 1,255
Trainable params: 1,255
Non-trainable params: 0
```

Κεφάλαιο 5
Αποτελέσματα πειράματος

5.1 Μέθοδος αξιολόγησης αποτελεσμάτων

Για την αξιολόγηση των μοντέλων ανίχνευσης εσωτερικών απειλών, χρησιμοποιήσαμε τα πιο κάτω μέτρα:

1. Precision είναι ο λόγος των ορθών θετικών περιπτώσεων επί του συνόλου των θετικών προβλέψεων

$$\text{Precision} = \frac{\text{True Positive}}{(\text{True Positive} + \text{False Positive})}$$

2. Recall είναι ο λόγος των ορθών θετικών προβλέψεων επί του συνόλου των θετικών προτύπων

$$\text{Recall} = \frac{\text{True Positive}}{(\text{True Positive} + \text{False Negative})}$$

3. Accuracy είναι ο λόγος των θετικών ορθών προβλέψεων, επί του συνόλου των προτύπων

$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{(\text{True Positive} + \text{False Positive} + \text{True Negative} + \text{False Negative})}$$

4. F1-Score είναι η μέση τιμή μεταξύ Recall και Precision η οποία δίνει καλύτερη μέτρηση των λανθασμένων περιπτώσεων από το Accuracy

$$\text{F1-score} = \left(\frac{\text{Recall}^{-1} + \text{Precision}^{-1}}{2}\right)^{-1} = 2 * \frac{(\text{Precision} * \text{Recall})}{(\text{Precision} + \text{Recall})}$$

5.2 Αποτελέσματα Autoencoder

Σύμφωνα με το μοντέλο του autoencoder πήραμε τα πιο κάτω αποτελέσματα.

Precision: 0.90 Recall: 0.95 Accuracy: 0.95 F1-Score: 0.92

```
train_x, val_x, train_y, val_y = train_test_split(rep_x, rep_y, test_size=0.25)
clf = LogisticRegression(solver="lbfgs").fit(train_x, train_y)
pred_y = clf.predict(val_x)

print ("")
print ("Classification Report: ")
print (classification_report(val_y, pred_y))

print ("")
print ("Accuracy Score: ", accuracy_score(val_y, pred_y))
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.95      1.00      0.97       178
         1.0       0.00      0.00      0.00        10

    accuracy                           0.95       188
   macro avg       0.47      0.50      0.49       188
weighted avg       0.90      0.95      0.92       188

Accuracy Score:  0.9468085106382979
```

Πίνακας 3:Autoencoder Classification Report

Βλέπουμε να αποτελέσματα της ανίχνευσης είναι αρκετά ψηλά και στα τέσσερα (4) μέτρα αξιολόγησης.

5.3 Αποτελέσματα Variational Autoencoder

Σε σχέση με το μοντέλο του Variational Autoencoder πήραμε τα πιο κάτω αποτελέσματα.

Precision: 0.92 Recall: 0.96 Accuracy: 0.96 F1-Score: 0.94

```
:
train_x, val_x, train_y, val_y = train_test_split(rep_x, rep_y, test_size=0.25)
clf = LogisticRegression(solver="lbfgs").fit(train_x, train_y)
pred_y = clf.predict(val_x)

print ("")
print ("Classification Report: ")
print (classification_report(val_y, pred_y))

print ("")
print ("Accuracy Score: ", accuracy_score(val_y, pred_y))
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.96      1.00      0.98       180
         1.0       0.00      0.00      0.00         8

    accuracy                           0.96       188
   macro avg       0.48      0.50      0.49       188
weighted avg       0.92      0.96      0.94       188


Accuracy Score:  0.9574468085106383
```

Βλέπουμε τα αποτελέσματα ανίχνευσης είναι αρκετά ψηλά και στα τέσσερα (4) μέτρα αξιολόγησης.

5.4 Σύγκριση αποτελεσμάτων με άλλες μεθοδολογίες

Δημιουργήσαμε τον πιο κάτω πίνακα για να συγκρίνουμε τα αποτελέσματα μας απέναντι σε άλλες μεθοδολογίες συγκρίνοντας τις τιμές Precision, Recall, Accuracy και F1-Score τις οποίες εξηγήσαμε πιο πάνω. Επιλέξαμε μελέτες οι οποίες χρησιμοποιήσανε το ίδιο dataset από το Cert.

	Precision	**Recall**	**Accuracy**	**F1-Score:**
Autoencoder (AE)	0.90	0.95	0.95	0.92
Variational Autoencoder (VAE)	0.92	0.96	0.96	0.94
CNN[6]	n/a	n/a	0.90	n/a
Mitigating Insider threats Using Bio-Inspired models[7](9 features)	n/a	1	n/a	0.07

Σε σχέση με τα μοντέλα Bio-inspired επιλέξαμε να συγκρίνουμε το μοντέλο που χρησιμοποιεί 9 features όπως και τα μοντέλα μας. Με βάση τα αποτελέσματα, μη έχοντας αποτελέσματα για τις υπόλοιπους μεθόδους αξιολόγησης είναι δύσκολο να έχουμε μία καθαρή εικόνα, αλλά εκ πρώτης όψεως, τα μοντέλα δείχνουν ικανά να μελετηθούν περεταίρω.

Κεφάλαιο 6
Επίλογος

6.1 Συμπεράσματα

Στόχος της παρούσας διατριβής ήταν υλοποιήσουμε και έπειτα να αξιολογήσουμε δύο (2) αλγόριθμους Τεχνητής Νοημοσύνης σε σχέση με την αποτελεσματικότητα τους. Επιλέξαμε τον Autoencoder και τον Variational Autoencoder διότι είναι αλγόριθμοι κατηγορίας Τεχνητών

Νευρωνικών Δικτύων, είναι αλγόριθμοι χωρίς επίβλεψη και χρησιμοποιούνται για ανίχνευση ανωμαλιών, που ήταν ο αρχικός μας στόχος.

	Precision	**Recall**	**Accuracy**	**F1-Score:**
AE	0.90	0.95	0.95	0.92
VAE	0.92	0.96	0.96	0.94

Επιλέξαμε να χρησιμοποιήσουμε τέσσερα (4) διαφορετικά μέτρα για να αξιολογήσουμε την αποτελεσματικότητα τους και τα αποτελέσματα ήταν πολύ κοντά. Αυτό πιθανώς έχει να κάνει και με τον τρόπο που δημιουργούνται αυτά τα μοντέλα, δηλαδή συνδυάζοντας παρόμοια νευρωνικά δίκτυα.

Μια από τις πιο χρονοβόρες διαδικασίες σε σχέση με των υλοποίηση των δύο (2) μοντέλων αφορά την προετοιμασία των δεδομένων για να χρησιμοποιηθούν από το μοντέλο. Αρχικά λόγω των διαφορετικών data types και την κανονικοποίηση των δεδομένων (normalization) απαιτεί ιδιαίτερη προετοιμασία των δεδομένων. Αυτό ίσως να μην μπορούσε να είναι το ίδιο αποτελεσματικό με συνεχής ροή δεδομένων. Επιπλέον για την υλοποίηση των μοντέλων χρησιμοποιήσαμε συνθετικά δεδομένα τα οποία αντιπροσωπεύουν διάφορα σενάρια κακόβουλων χρηστών. Η ανίχνευση απειλών σε έναν οργανισμό, απαιτεί τη χρήση πολλαπλών συστημάτων ασφαλείας , τα οποία συλλέγουν τα logs τους με διαφορετική μορφή που πιθανώς να απαιτεί τη δημιουργία parser πριν πάρουν την μορφή που χρησιμοποιήσαμε στο dataset.

6.2 Επόμενα βήματα

Σαν επόμενο βήμα θα ήταν καλό να χρησιμοποιηθεί το συγκεκριμένο dataset με διαφορετικούς αλγόριθμους ανίχνευσης ανωμαλιών και να γίνει σύγκριση των αποτελεσμάτων. Μπορούν να χρησιμοποιηθούν αλγόριθμοι με επίβλεψη όπως και συνδυασμός αλγορίθμων εάν είναι εφικτό.

Βιβλιογραφία

[1] G. B. Magklaras and S. M. Furnell, "A preliminary model of end user sophistication for insider threat prediction in IT systems," *Computers & Security,* pp. 371-380, 2005.

[2] B. Balakkrishman, "Insider Threat Mitigation Guidance," The SANS Institute, 2015.

[3] R. S. Kuheli, "Assessing insider threats to information security using technical, behavioural and organisational measures," Information Security Technical Report 15, pp. 112-133, 2010

[4] Papadaki. M. and Shiaeles. S. (2018). "Insider Threat: The forgotten, yet formidable foe", Chapter in Human Computer Interaction in Cyber-Security Handbook, CRC-Press, Taylor & Francis Group

[5] Collins, Matt., Michael Theis, Randall Trzeciak, Jeremy Strozer, Jeremy Clark, Daniel Costa, Tracey Cassidy, Michael Albrethsen, and Andrew Moore. Common Sense Guide to Mitigating Insider Threats, 5th Edition (Technical Report CMU/SEI-2016-TR-015). Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA (2016). Available at: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=484738 J. Graves, "How machine learning is catching up with the insider threat," Cyber Security: A Peer-Reviewed Journal, vol. 1, no. 2, pp. 127-133, 2017

[6] V. Koutsouvelis, S. Shiaeles, B. Ghita and G. Bendiab, "Detection of Insider Threats using Artificial Intelligence and Visualisation," 2020 6th IEEE Conference on Network Softwarization (NetSoft), Ghent, Belgium, 2020, pp. 437-443, doi: 10.1109/NetSoft48620.2020.9165337.

[7] Nicolaou, Andreas; Shiaeles, Stavros; Savage, Nick. 2020. "Mitigating Insider Threats Using Bio-Inspired Models." *Appl. Sci.* 10, no. 15: 5046.

[8] W. Li, W. Meng, L.-F. Kwok and H. H. IP, "Enhancing Collaborative Intrusion Detection Networks against Inside Attacks using Supervised Intrusion Sensitivity-Based Trust Management Model," *Network and Computer Applications,* pp. 135-145, 2017.

[9] S. M. Sarma, Y. Srinivas, V. M. Abhiram, L. Ullala, M. S. Prasanthi and J. R. Rao, "Insider Threat Detection with Face Recognition and KNN User Classification," in *International Conference on Cloud Computing in Emerging Markets (CCEM)*, Bangalore, India, 2017.

[10] M. Choras and R. Kozik, "Machine Learning Techniques for Threat Modeling and Detection," *Security and Resilience in Intelligent Data-Centric Systems and Communication Networks,* pp. 179-192, 2018

[11] A. Tuor, S. Kaplan, B. Hutchinson, N. Nichols and S. Robinson, "Deep Learning for Unsupervised Insider Threat Detection in Structured Cybersecurity Data Streams," pp. 1-9, 2017.

[12] L. Lin, S. Zhong, C. Jia and K. Chen, "Insider Threat Detection Based on Deep Belief Network Feature Representation," in *2017 International Conference on Green Informatics*, 2017.

[13] P. A. Legg, "Human-Machine Decision Support Systems for Insider Threat Detection," *Data Analytics and Decision Support for Cybersecurity*, pp. 46-66, April 2017.

[14] P. A. Legg, O. Buckley, M. Goldsmith and S. Creese, "Automated Insider Threat Detection System using User and Role-based Profile Assessment," *IEEE SYSTEMS JOURNAL,* pp. 503 - 512, 2017.

[15] Chen, J., Sathe, S., Aggarwal, C. and Turaga, D., 2017, June. Outlier detection with autoencoder ensembles. *SIAM International Conference on Data Mining*, pp. 90-98. Society for Industrial and Applied Mathematics.

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. http://www. deeplearningbook.org MIT Press, 2016.

[17] Mitchell, T. (1997). *Machine Learning*, McGraw Hill, *Machine Learning*, McGraw Hill, p.2

[18] D. P. Kingma and M. Welling. Auto-encoding variational Bayes. In *ICLR*, 2014.

[19] Wang, X., Du, Y., Lin, S., Cui, P., Shen, Y. and Yang, Y., 2019. adVAE: A self-adversarial variational autoencoder with Gaussian anomaly prior knowledge for anomaly detection. *Knowledge-Based Systems*.

[20] Y. Liu *et al.*, "Generative Adversarial Active Learning for Unsupervised Outlier Detection," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 8, pp. 1517-1528, 1 Aug. 2020, doi: 10.1109/TKDE.2019.2905606.

[21] Jupyter.org. (2020). Project Jupyter. [online] Available at: <https://jupyter.org/> [Accessed 12 September 2020].

[22] tensorflow.org. (2020).. [online] Available at: <https://jupyter.org/> [Accessed 12 September 2020].

[23] Zhao, Y., Nasrullah, Z. and Li, Z., 2019. PyOD: A Python Toolbox for Scalable Outlier Detection. Journal of machine learning research (JMLR), 20(96), pp.1-7.

[24] Glasser, J. and Lindauer, B. (2013). Bridging the Gap: A Pragmatic Approach to Generating Insider Threat Data. 2013 IEEE Security and Privacy Workshops (pp. 98-104). IEEE.

[25] Classify structured data with feature columns Available at <https://tensorflow.org/>[Accessed 2 October 2020]

[26] Zimmerer D., Isensee F., Petersen J., Kohl S., Maier-Hein K. (2019) Unsupervised Anomaly Localization Using Variational Auto-Encoders. In: Shen D. et al. (eds) Medical Image Computing and Computer Assisted Intervention – MICCAI 2019. MICCAI 2019. Lecture Notes in Computer Science, vol 11767. Springer, Cham. https://doi.org/10.1007/978-3-030-32251-9_32

[27] Building Autoencoders in Keras Available at < https://blog.keras.io/>[Accesed 4 October 2020]

Παράρτημα Α
Κώδικας Python

Α.1 Επεξεργασία Δεδομένων

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
import time as t
%matplotlib inline
from datetime import datetime, date, time
```

In [20]:

```
#create samples files for our experiment using 50000 rows per
csv file

#logins
sample_Logon = pd.read_csv('/Volumes/CERT-
DATA/r4.2/logon.csv',low_memory=False, nrows=50000)
sample_Logon.to_csv('/Volumes/CERT-DATA/r4.2/sample-
logon.csv', index=False)
#devices
sample_Device = pd.read_csv('/Volumes/CERT-
DATA/r4.2/device.csv', low_memory=False, nrows=50000)
sample_Device.to_csv('/Volumes/CERT-DATA/r4.2/sample-
device.csv', index=False)
```

```
#files
sample_File = pd.read_csv('/Volumes/CERT-DATA/r4.2/file.csv',
low_memory=False, nrows=50000)
sample_File.to_csv('/Volumes/CERT-DATA/r4.2/sample-file.csv',
index=False)
#http
sample_http = pd.read_csv('/Volumes/CERT-DATA/r4.2/http.csv',
low_memory=False, nrows=50000)
sample_http.to_csv('/Volumes/CERT-DATA/r4.2/sample-http.csv',
index=False)
#email
sample_email = pd.read_csv('/Volumes/CERT-
DATA/r4.2/email.csv', low_memory=False, nrows=50000)
sample_email.to_csv('/Volumes/CERT-DATA/r4.2/sample-
email.csv', index=False)
```

In [21]:

```
#load sample files into in panda dataframes using the desired
columns (features)

#load logins.csv into a pandas DataFrame
logins = pd.read_csv('/Volumes/CERT-DATA/r4.2/sample-
logon.csv', usecols=['date', 'user', 'pc', 'activity'])

#load devices.csv into a pandas DataFrame
devices = pd.read_csv('/Volumes/CERT-DATA/r4.2/sample-
device.csv', usecols=['date', 'user', 'pc',
'activity'])

#load files.csv into a pandas DataFrame
files = pd.read_csv('/Volumes/CERT-DATA/r4.2/sample-file.csv',
usecols=['date', 'user', 'pc'])

#load http.csv into a pandas DataFrame
http = pd.read_csv('/Volumes/CERT-DATA/r4.2/sample-http.csv',
usecols=['date', 'user', 'pc'])

#load email.csv into a pandas DataFrame
email = pd.read_csv('/Volumes/CERT-DATA/r4.2/sample-
email.csv', usecols=['date', 'user', 'pc'])
```

In [22]:

```
#create activity  for the email, files and http

email['activity'] = 5 #create activity for email
files['activity'] = 6 #create activity for files
http['activity'] = 7 #create activity for http
```

In [23]:

```
#Merge dataframes into one large dataframe

frames = [devices, files, logins, http, email]
df = pd.concat(frames)

#del initial data frames
```

```
del files, logins, devices, http, email
```

In [24]:

```
#convert all acticities in the dataset to integers

activities = ['Logon', 'Logoff', 'Connect', 'Disconnect']
i = 1
for activity in activities:
    df['activity'].replace(activity,i, inplace=True)
    i = i + 1
    del activity

del i, activities
```

In [25]:

```
#Perform One Hot Encoding

df['Logon'] = 0
df['Logoff'] = 0
df['Connect'] = 0
df['Disconnect'] = 0
df['email'] = 0
df['file'] = 0
df['http'] = 0

#Perform One Hot Encoding
df.loc[df.activity == 1,"Logon"] = 1
df.loc[df.activity == 2,"Logoff"] = 1
df.loc[df.activity == 3,"Connect"] = 1
df.loc[df.activity == 4,"Disconnect"] = 1
df.loc[df.activity == 5,"email"] = 1
df.loc[df.activity == 6,"file"] = 1
df.loc[df.activity == 7,"http"] = 1
```

In [27]:

```
#convert date column to day and time
https://pandas.pydata.org/pandasdocs/stable/reference/api/pand
as.to_datetime.html
df['date'] =
pd.to_datetime(df['date'],infer_datetime_format=True)

#Split date column in day and time | #PARSE the date from
strings
days , times = zip(*[(d.date().weekday(),
float((d.time().minute/60)+d.time().hour)) for d in
df['date']])
df = df.assign(day = days , time = times)

del days, times #save resources
```

In [28]:

```
#load the usernames of the malicious insiders according to the
scenarios
insiders = [ "AAF0535", "ABC0174", "AKR0057", "CCL0068",
"CEJ0109", "CQW0652", "DIB0285", "DRR0162", "EDB0714",
```

```
            "EGD0132", "FSC0601", "HBO0413", "HXL0968",
"IJM0776", "IKR0401", "IUB0565", "JJM0203", "KRL0501",
            "LCC0819", "MDH0580", "MOS0047", "NWT0098",
"PNL0301", "PSF0133", "RAR0725", "RHL0992", "RMW0542",
            "TNM0961", "VSS0154", "XHW0498"]
```

In [29]:

```
#create the column insider

df['insider'] = 0
#if the user is insider set feature to 1
df.loc[(df['user'].isin(insiders)),"insider"] = 1
```

In [30]:

```
#sort dataset by date

df.sort_values('date', axis=0, inplace=True)
```

In [35]:

```
#create samples of the first 1000 rows of the dataframe
header = ["day", "time", "Logon", "Logoff", "Connect",
"Disconnect", "email", "file", "http"]
df_insiderthreat = df.head(1000)
```

In [32]:

```
df_insiderthreat.to_csv('/Volumes/CERT-
DATA/r4.2/insider_dataset.csv')
```

A.2 Autoencoder (AE)

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import numpy as np

import tensorflow as tf
from tensorflow.python import tf2
if not tf2.enabled():
    import tensorflow.compat.v2 as tf
    tf.enable_v2_behavior()
    assert tf2.enabled()

# import tensorflow_datasets as tfds
import tensorflow_probability as tfp
```

```
tfk = tf.keras
tfkl = tf.keras.layers
tfpl = tfp.layers
tfd = tfp.distributions

from tensorflow.keras.utils import plot_model
from tensorflow.keras.callbacks import EarlyStopping,
ModelCheckpoint
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report,
accuracy_score, roc_auc_score, roc_curve
from sklearn.preprocessing import MinMaxScaler
from sklearn.manifold import TSNE
import os
from keras.layers import Input, Dense
from keras.models import Model, Sequential
from keras import regularizers
from sklearn.model_selection import train_test_split

from sklearn import preprocessing
import seaborn as sns
sns.set(style="whitegrid")
np.random.seed(203)
%matplotlib inline
np.random.seed(0)
tf.random.set_seed(0)
```

In [ ]:

```
### Utility Functions
## Plots


def tsne_plot(x1, y1, name="graph.png"):
    tsne = TSNE(n_components=2, random_state=0)
    X_t = tsne.fit_transform(x1)
#     plt.figure(figsize=(12, 8))
    plt.scatter(X_t[np.where(y1 == 0), 0], X_t[np.where(y1 ==
0), 1], marker='o', color='g', linewidth='1', alpha=0.8,
label='Non Fraud', s=2)
    plt.scatter(X_t[np.where(y1 == 1), 0], X_t[np.where(y1 ==
1), 1], marker='o', color='r', linewidth='1', alpha=0.8,
label='Fraud', s=2)

    plt.legend(loc='best');
    plt.savefig(name);
    plt.show();

# Plot Keras training history
def plot_loss(hist):
    plt.plot(hist.history['loss'])
```

```
    plt.plot(hist.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.yscale('log',basey=10)
    plt.show()
```

In [ ]:

```
# import data file and read the data, split the data into
training set and test set.


raw_data = pd.read_csv("insider_dataset.csv")

del raw_data["date"]
del raw_data["user"]
del raw_data["pc"]
del raw_data["day"]
del raw_data["time"]

data, data_test = train_test_split(raw_data, test_size=0.25)


non_fraud = data[data['insider'] == 0].sample(600)
fraud = data[data['insider'] == 1]

df =
non_fraud.append(fraud).sample(frac=1).reset_index(drop=True)
X = df.drop(['insider'], axis = 1).values
Y = df["insider"].values
```

In [ ]:

```
## input layer
input_layer = Input(shape=(X.shape[1],))

## encoding part
encoded = Dense(100, activation='tanh',
activity_regularizer=regularizers.l1(10e-5))(input_layer)
encoded = Dense(50, activation='relu')(encoded)

## decoding part
decoded = Dense(50, activation='tanh')(encoded)
decoded = Dense(100, activation='tanh')(decoded)

## output layer
output_layer = Dense(X.shape[1], activation='relu')(decoded)
```

In [ ]:

```
autoencoder = Model(input_layer, output_layer)
autoencoder.compile(optimizer="adadelta", loss="mse")
```

In [ ]:

```
x = data.drop(["insider"], axis=1)
y = data["insider"].values
```

```
x_scale = preprocessing.MinMaxScaler().fit_transform(x.values)
x_norm, x_fraud = x_scale[y == 0], x_scale[y == 1]

x_norm_sample = x_norm[np.random.randint(x_norm.shape[0],
size=600), :]
x_norm_train_sample, x_norm_val_sample =
train_test_split(x_norm_sample, test_size=0.2)
```

In [ ]:

```
vae.load_weights('bestmodel.h5', by_name=True)
```

In [ ]:

```
reconstruct_samples_n = 100

def reconstruction_log_prob(eval_samples,
reconstruct_samples_n):
    encoder_out = encoder(eval_samples)
    encoder_samples =
encoder_out.sample(reconstruct_samples_n)
    return
np.mean(decoder(encoder_samples).log_prob(eval_samples),
axis=0)
```

In [ ]:

```
autoencoder.fit(x_norm, x_norm,
                batch_size = 256, epochs = 30,
                shuffle = True, validation_split = 0.20);
```

In [ ]:

```
hidden_representation = Sequential()
hidden_representation.add(autoencoder.layers[0])
hidden_representation.add(autoencoder.layers[1])
hidden_representation.add(autoencoder.layers[2])
```

In [ ]:

```
norm_hid_rep = hidden_representation.predict(x_norm)
fraud_hid_rep = hidden_representation.predict(x_fraud)
```

In [ ]:

```
train_x, val_x, train_y, val_y = train_test_split(rep_x,
rep_y, test_size=0.25)
clf = LogisticRegression(solver="lbfgs").fit(train_x, train_y)
pred_y = clf.predict(val_x)

print ("")
print ("Classification Report: ")
print (classification_report(val_y, pred_y))

print ("")
print ("Accuracy Score: ", accuracy_score(val_y, pred_y))
```

A.3 Variational Autoencoder (VAE)

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import numpy as np

import tensorflow as tf
from tensorflow.python import tf2
if not tf2.enabled():
    import tensorflow.compat.v2 as tf
    tf.enable_v2_behavior()
    assert tf2.enabled()

# import tensorflow_datasets as tfds
import tensorflow_probability as tfp

tfk = tf.keras
tfkl = tf.keras.layers
tfpl = tfp.layers
tfd = tfp.distributions

from tensorflow.keras.utils import plot_model

from tensorflow.keras.callbacks import EarlyStopping,
ModelCheckpoint
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report,
accuracy_score, roc_auc_score, roc_curve
from sklearn.preprocessing import MinMaxScaler
from sklearn.manifold import TSNE
import os

%matplotlib inline
np.random.seed(0)
tf.random.set_seed(0)
print(tf.__version__)
print(tfp.__version__)
```

In [ ]:

```
from keras.layers import Input, Dense
from keras.models import Model, Sequential
from keras import regularizers
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report,
accuracy_score
from sklearn.manifold import TSNE
```

```
from sklearn import preprocessing
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
sns.set(style="whitegrid")
np.random.seed(203)
```

In [ ]:

```
# Utility Functions
# Plots
# Plot Feature Projection

def tsne_plot(x1, y1, name="graph.png"):
    tsne = TSNE(n_components=2, random_state=0)
    X_t = tsne.fit_transform(x1)
#     plt.figure(figsize=(12, 8))
    plt.scatter(X_t[np.where(y1 == 0), 0], X_t[np.where(y1 ==
0), 1], marker='o', color='g', linewidth='1', alpha=0.8,
label='Non Fraud', s=2)
    plt.scatter(X_t[np.where(y1 == 1), 0], X_t[np.where(y1 ==
1), 1], marker='o', color='r', linewidth='1', alpha=0.8,
label='Fraud', s=2)

    plt.legend(loc='best');
    plt.savefig(name);
    plt.show();

# Plot Keras training history
def plot_loss(hist):
    plt.plot(hist.history['loss'])
    plt.plot(hist.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.yscale('log',basey=10)
    plt.show()
```

In [ ]:

```
raw_data = pd.read_csv("insider_dataset.csv")
del raw_data["date"]
del raw_data["user"]
del raw_data["pc"]
del raw_data["day"]
del raw_data["time"]
data, data_test = train_test_split(raw_data, test_size=0.25)
```

In [ ]:

```
raw_data.head()
```

In [ ]:

```
raw_data.info()
```

In [ ]:

```
raw_data.shape
```

In [ ]:

```
non_fraud = data[data['insider'] == 0].sample(600)
fraud = data[data['insider'] == 1]

df =
non_fraud.append(fraud).sample(frac=1).reset_index(drop=True)
X = df.drop(['insider'], axis = 1).values
Y = df["insider"].values

#tsne_plot(X, Y, "original.png")
```

In [ ]:

```
def dense_layers(sizes):
    return tfk.Sequential([tfkl.Dense(size,
activation=tf.nn.leaky_relu) for size in sizes])

original_dim = X.shape[1]
input_shape = X[0].shape
intermediary_dims = [20, 10, 8]
latent_dim = 2
batch_size = 128
max_epochs = 1000

# prior = tfd.Independent(tfd.Normal(loc=tf.zeros(latent_dim),
scale=1),
#                         reinterpreted_batch_ndims=1)

prior = tfd.MultivariateNormalDiag(
        loc=tf.zeros([latent_dim]),
        scale_identity_multiplier=1.0)

encoder = tfk.Sequential([
    tfkl.InputLayer(input_shape=input_shape,
name='encoder_input'),
    dense_layers(intermediary_dims),

tfkl.Dense(tfpl.MultivariateNormalTriL.params_size(latent_dim)
, activation=None),
    tfpl.MultivariateNormalTriL(latent_dim,

activity_regularizer=tfpl.KLDivergenceRegularizer(prior)),
], name='encoder')

encoder.summary()
#plot_model(encoder, to_file='vae_mlp_encoder.png',
show_shapes=True)

decoder = tfk.Sequential([
    tfkl.InputLayer(input_shape=[latent_dim]),
    dense_layers(reversed(intermediary_dims)),

tfkl.Dense(tfpl.IndependentNormal.params_size(original_dim),
activation=None),
    tfpl.IndependentNormal(original_dim),
```

```
], name='decoder')

decoder.summary()
#plot_model(decoder, to_file='vae_mlp_decoder.png',
show_shapes=True)

vae = tfk.Model(inputs=encoder.inputs,
                outputs=decoder(encoder.outputs[0]),
                name='vae_mlp')

negloglik = lambda x, rv_x: -rv_x.log_prob(x)

vae.compile(optimizer=tf.keras.optimizers.Nadam(),
            loss=negloglik)

vae.summary()
#plot_model(vae,to_file='vae_mlp.png', show_shapes=True)
```

In [ ]:

```
x = data.drop(["insider"], axis=1)
y = data["insider"].values

x_scale = preprocessing.MinMaxScaler().fit_transform(x.values)
x_norm, x_fraud = x_scale[y == 0], x_scale[y == 1]
```

In [ ]:

```
norm_hid_rep = hidden_representation.predict(x_norm)
fraud_hid_rep = hidden_representation.predict(x_fraud)
```

In [ ]:

```
rep_x = np.append(norm_hid_rep, fraud_hid_rep, axis = 0)
y_n = np.zeros(norm_hid_rep.shape[0])
y_f = np.ones(fraud_hid_rep.shape[0])
rep_y = np.append(y_n, y_f)
#tsne_plot(rep_x, rep_y, "latent_representation.png")
```

In [ ]:

```
train_x, val_x, train_y, val_y = train_test_split(rep_x,
rep_y, test_size=0.25)
clf = LogisticRegression(solver="lbfgs").fit(train_x, train_y)
pred_y = clf.predict(val_x)

print ("")
print ("Classification Report: ")
print (classification_report(val_y, pred_y))

print ("")
print ("Accuracy Score: ", accuracy_score(val_y, pred_y))
```

In [ ]:

In [ ]: