Technical Report ML assignment

1.Understanding RAW DATA

Εδώ θεωρούμε ως raw data το NSL-KDD99 dataset. Αυτό το dataset έχει προκύψει από το KDD99 CUP dataset με κάποιες διορθώσεις όπως απαλοιφή των διπλών εγγραφών και αναλογία όσον αφορά τα entries με δυσκολία. Η δυσκολία περιγράφεται από ένα column που δίνει με μέγιστο το 21 πόσοι classifiers αναγνώρισαν σωστά το datapoint.
Το dataset περιέχει στοιχεία από normal κίνηση καθώς και 4 διαφορετικά είδη επιθέσεων.
Επεξήγηση για τα διαφορετικά features υπάρχει εδώ 1563006133_9702-etarjome-English.pdf (e-tarjome.com) .
Τα τέσσερα διαφορετικά είδη επιθέσεων είναι και οι υποκατηγορίες της κάθε κατηγορίας είναι οι παρακάτω

TABLE VII : MAPPING OF ATTACK CLASS WITH ATTACK TYPE

Attack Class	Attack Type
DoS	Back, Land, Neptune, Pod, Smurf,Teardrop,Apache2, Udpstorm, Processtable, Worm (10)
Probe	Satan, Ipsweep, Nmap, Portsweep, Mscan, Saint (6)
R2L	Guess_Password, Ftp_write, Imap, Phf, Multihop, Warezmaster, Warezclient, Spy, Xlock, Xsnoop, Snmpguess, Snmpgetattack, Httptunnel, Sendmail, Named (16)
U2R	Buffer_overflow, Loadmodule, Rootkit, Perl, Sqlattack, Xterm, Ps (7)

Έχουμε συνολικά 41 features σε κάθε datapoint τα οποία αφορούν 3 κατηγορίες όπως περιγράφονται παρακάτω.

TABLE VI: ATTRIBUTE VALUE TYPE

Type	Features
Nominal	Protocol_type(2), Service(3), Flag(4)
Binary	Land(7), logged_in(12), root_shell(14), su_attempted(15), is_host_login(21),, is_guest_login(22)
umeric	Duration(1), src_bytes(5), dst_bytes(6), wrong_fragment(8), urgent(9), hot(10), num_failed_logins(11), num_compromised(13), num_root(16), num_file_creations(17), num_shells(18), num_access_files(19), num_outbound_cmds(20), count(23) srv_count(24), serror_rate(25), srv_serror_rate(26), rerror_rate(27), srv_rerror_rate(28), same_srv_rate(29) diff_srv_rate(30), srv_diff_host_rate(31), dst_host_count(32), dst_host_srv_count(33), dst_host_same_srv_rate(34), dst_host_diff_srv_rate(35), dst_host_same_src_port_rate(36), dst_host_srv_diff_host_rate(37), dst_host_serror_rate(38), dst_host_srv_serror_rate(39), dst_host_rerror_rate(40), dst_host_srv_rerror_rate(41)

Το dataset που χρησιμοποιούμε ως raw data είναι χωρισμένο σε train και test. Δεν γνωρίζουμε με ποια κριτήρια έχει χωριστεί από το αρχικό KDD Cup 1999 Data (uci.edu) το οποίο για κάποιο λόγω δεν ανοίγει. Επειδή η κλάση U2R έχει πολύ λίγα samples δημιουργεί πρόβλημα διότι θέλουμε ίδιο αριθμό samples ανά κλάση στο dataset που θα δημιουργήσουμε. Αναλυτικά οι κατανομές των κλάσεων σε train/test είναι οι παρακάτω.

TABLE VIII: DETAILS OF NORMAL AND ATTACK DATA IN DIFFERENT TYPES OF NSL-KDD DATA SET

Data Set Type	Total No. of					
	Records	Normal Class	DoS Class	Probe Class	U2R Class	R2L Class
KDD Train+ 20%	25192	13449	9234	2289	11	209
		53.39 %	36.65 %	9.09%	0.04 %	0.83 %
KDD Train+	125973	67343	45927	11656	52	995
		53.46 %	36.46	9.25%	0.04 %	0.79 %
KDD Test+	22544	9711	7458	2421	200	2754
		43.08 %	33.08 %	10.74 %	0.89 %	12.22 %

Μέθοδος δημιουργίας του dataset

Αρχικά θα δημιουργήσουμε ένα καινούριο dataset που θα αποτελείται από 5000 samples 1000 από κάθε κλάση. Προσπαθήσαμε εκτός από τις highlevel κλάσεις που πρέπει να είναι balanced να είναι balanced και οι υποκλάσεις στο βαθμό που είναι δυνατόν. Έτσι επιλέξαμε όλα τα samples των minority και με αναλογία όλα τα υπόλοιπα διαθέσιμα για την περίπτωση που κάνουμε δεύτερο classification στις υποκατηγορίες κάθε highlevel κλάσης. Το dataset που προέκυψε έχει μέγεθος 4500 samples τα οποία θα γίνουν 5000 όταν θα χρησιμοποιήσουμε smote για να παράγουμε νέα samples για την κλάση U2R .Στην συνέχεια θα δοκιμάσουμε την μέθοδο weights για να αντιμετωπίσουμε το πρόβλημα των imbalanced κλάσεων. Αν δεν έχουμε καλά αποτελέσματα με αυτό θα δοκιμάσουμε smote.
Με βάση τον τελευταίο πίνακα βλέπουμε ότι στο αρχείο KDDtrain+ έχουμε αρκετά samples ανά κλάση για να δημιουργήσουμε το νέο dataset συνεπώς μπορούμε να χρησιμοποιήσουμε αυτό το csv. Επίσης επειδή θέλουμε όλα τα samples από την U2R και για ευκολία θα ενώσουμε τα δύο CSVs και θα κάνουμε το sampling από την ένωση και θα δημιουργήσουμε ένα νέο csv με το οποίο θα συνεχίσουμε.

- Επιλογή samples από τις 4 κλάσεις. Για την U2R δεν έχουμε επιλογή θα κρατήσουμε όλα τα διαθέσιμα. Για την DoS,Probe,R2L και Normal θα επιλέξουμε με τυχαίο τρόπο. Πρώτα θα δημιουργήσουμε ένα dataframe για κάθε κλάση όπου θα έχουμε όλα τα datapoints που έχουν το αντίστοιχο label. Στην συνέχεια θα επιλέξουμε αν είναι δυνατόν ίδιο αριθμό samples ανά υποκλάση.

- Αφού δημιουργήσουμε το dataframe θα αφαιρέσουμε τα columns που δεν χρειάζονται δηλαδή αυτό της δυσκολίας και θα δημιουργήσουμε ακόμα ένα αυτό με την high level class που ανήκει το κάθε datapoint.
- Τέλος θα σώσουμε το dataframe σε csv με το οποίο θα συνεχίσουμε.

Το script που έτρεξε είναι το **scrip1** και αυτό που κάνει είναι να ενώσει τα δύο datasets και να διαλέξει τυχαία τα 5000 samples αφού έχει προηγηθεί annotation της κάθε highlevel κλάσης με βάση τις υποκλάσεις.

2.DATA PREPROCESSING

Εδώ επεξεργαζόμαστε το csv που δημιουργήσαμε στο προηγούμενο με σκοπό να φέρουμε όλα τα attributes σε μορφή κατάλληλη για τον αλγόριθμο. Αρχικά πρέπει να αντιμετωπίσουμε τα nominal attributes. Υπάρχουν πολλοί τρόποι να γίνει αυτό. Για το feature protocol και flag τα οποία παίρνουν λίγες τιμές(3 και 10) θα μπορούσαμε να κάνουμε one hot encoding για το protocol ,buissness logic για το flag και για το service το οποίο παίρνει 40 τιμές θα πρέπει να βρούμε άλλο τρόπο. Ο μόνος τρόπος που καταφέραμε να βρούμε είναι το feature hashing scheme. Αυτός ο αλγόριθμος μπορεί να έχει ως είσοδο strings και να παράγει vectors με προκαθορισμένο μέγεθος, έτσι δεν θα έχουμε διπλασιασμό των features όπως θα γινόταν με το one hot. Μία εναλλακτική θα ήταν να εφαρμόσουμε buisness logic και να ενοποιήσουμε τις κατηγορίες .Έτσι για το protocol που παίρνει μόνο 3 τιμές θα κάνουμε onehot. Για το flag θα χρησιμοποιήσουμε buissness logic αφού παρατηρούμε από την επεξήγηση των πιθανών τιμών καθώς και από τα counts που παρουσιάζονται στο dataset οι τιμές δείχνουν συνδέσεις που έχουν επιτυχία και συνδέσεις σε κάποιο στάδιο δεν φαίνονται ενεργές/συνηθισμένες. Εδώ κάποιες τιμές είναι πιο αντιπροσωπευτικές της κατάστασης και δίνουν μεγαλύτερη πληροφορία όπως πχ η S0 με την S1 οπότε θα μπορούσαμε να ερμηνεύσουμε την κατηγορία αυτή ως ordinal categorical και να χρησιμοποιήσουμε την πληροφορία από τα flags με ανάλογο τρόπο ώστε να είναι συγκρίσιμες οι τιμές τους. Για ευκολία αυτό το attribute θα το μετατρέψουμε σε binary όπου η μία τιμή θα δείχνει αν το flag είναι ύποπτο ή όχι. Στην πρώτη κατηγορία θα είναι το flag SF και στην άλλη όλα τα υπόλοιπα. Για ευκολία το flag OTH(other) θα το βάλουμε στην κατηγορία με τα ύποπτα αντί να δημιουργήσουμε μία κατηγορία μόνο για αυτό. Αυτό το δικαιολογούμε με το σκεπτικό ότι για να μην είναι στην κατηγορία με την συνηθισμένη δραστηριότητα μάλλον θα είναι ύποπτο. Οι επεξηγήσεις για τα flags είναι οι παρακάτω.

STATUS FLAG OF THE CONNECTION

State	Meaning
SF	Normal SYN/FIN completion
REJ	Connection rejected, Initial SYN elicited a RST in reply
S0	State 0: initial SYN seen but no reply
S1	State 1: connection established (SYN's exchanged), nothing further seen
S2	State 2: connection established, initiator has closed their side
S3	State 3: connection established, responder has closed their side
RSTO	Connection reset by the originator
RSTR	Connection reset by the responder
OTH	Other, a state not contemplated here.

Τέλος για το attribute service θα κάνουμε feature hashing με παράμετρο 5.Δεν βρήκα guidelines/rule of thumb για το πώς να επιλέξω το μέγεθος του vector με βάση το σύνολο των στοιχείων της κλάσης. Πρόκειται για σημείο που θέλει πειραματισμό ως προς τα collisions που πιθανόν να προκύψουν οπότε είναι trial/error διαδικασία. Στην συνέχεια έχουμε numeric attributes. Μεγάλο μέρος τους δείχνουν ρυθμό και έχουν εύρος [0,1] οπότε όσα αφορούν τέτοια παρατήρηση θα μείνουν ως έχουν. Για τα υπόλοιπα θα εφαρμόσουμε normalization ώστε να βρεθούν και αυτά στο [0,1].Παρατηρούμε ότι το num_outbound_cmds δίνει NaN μετά το normalization, αυτό συμβαίνει διότι όλες οι τιμές είναι 0 συνεπώς δεν δίνει πληροφορία και στην συνέχεια θα το κάνουμε drop.

Το script που έτρεξε είναι το **script2** και κάνει normalization,hash trick,buissness logic. Τέλος σώζουμε το csv με το οποίο και θα συνεχίσουμε.

3.Επιλογή Αλγορίθμου και CLASS IMBALANCE

Με βάση αυτά που είπαμε στο μάθημα και το use case που έχουμε, δηλαδή υψηλό dimensionality αρκετά υψηλό αριθμό samples και την απαίτηση το testing time της τελικής λύσης να είναι χαμηλό διότι θα πρέπει να επεξεργάζεται μεγάλο όγκο πληροφορίας σε πραγματικό χρόνο επιλέγουμε να πειραματιστούμε με τους αλγορίθμους classification SVC και DT. Επιλέγουμε τον SVC διότι έχει πολύ καλή απόδοση σε σχέση με το accuracy και το testing time δεν είναι απαγορευτικά υψηλό. Αντίστοιχα επιλέγουμε DT διότι έχει πολύ μικρό testing time και αρκετά καλή απόδοση στο accuracy επιπλέον με αυτόν τον αλγόριθμο και την υλοποίηση του έχουμε ένα score για κάθε feature που μας λέει πόσο σημαντικό είναι πράγμα που θα χρειαστούμε στο επόμενο βήμα. Στο τέλος θα γίνει σύγκριση και του accuracy/precision/recall και του testing time.
Σε αυτή την φάση θα τρέξουμε τους δύο αλγορίθμους με το dataset που δημιουργήσαμε στο προηγούμενο βήμα για να έχουμε ένα baseline απόδοσης. Τρέχοντας για DT βλέπουμε ότι τα πάει πολύ καλά και δίνει accuracy 95% υψηλό recall/precision με την κλάση U2R να μένει πίσω κάτι που το περιμέναμε. Να σημειωθεί ότι μας ενδιαφέρει κυρίως το recall. Για τον SVC το training time είναι κατά πολύ μεγαλύτερο και η απόδοση σχετικά με τα metrics χειρότερη από του DT. Στην συνέχεια θα δοκιμάσουμε να τρέξουμε τους αλγορίθμους με το flag class_weight='balanced' και να κάνουμε smote. Ανάλογα με τα αποτελέσματα θα αποφασίσουμε αν θα συνεχίσουμε με smote. Το script που χρησιμοποιούμε σε αυτό το βήμα είναι το **script3** το οποίο δίνει όλα τα metrics για τα 3 προηγούμενα σενάρια.

- **Σενάριο 1 No class weights No smote**

```
Precision for DT: [0.97826087 0.92567568 0.98039216 0.93413174 0.66666667]
Recall for DT: [0.97826087 0.93835616 0.96463023 0.92857143 0.74074074]

Precision for SVM: [0.90697674 0.82474227 0.894081   0.85833333 0.8       ]
Recall for SVM: [0.84782609 0.82191781 0.92282958 0.91964286 0.44444444]
```

- **Σενάριο 2 class weights**

```
Precision with class weights for DT: [0.97597598 0.95714286 0.96226415 0.90881459 0.89285714]
Recall with class weights for DT: [0.99085366 0.90540541 0.96226415 0.9522293  0.78125   ]

Precision with class weights for SVM: [0.88311688 0.83391003 0.89908257 0.86322188 0.57142857]
Recall with class weights for SVM: [0.82926829 0.81418919 0.9245283  0.9044586  0.625     ]
```

- **Σενάριο 3 Smote**

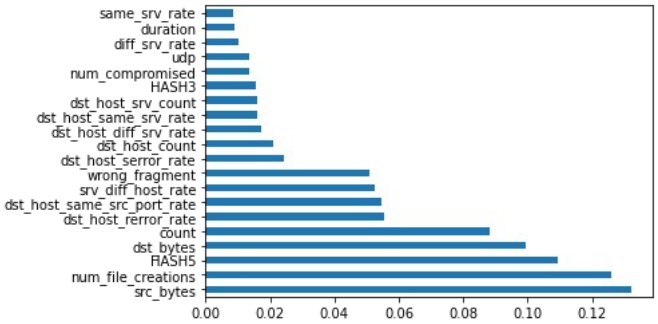
```
Precision with SMOTE for DT: [0.95924765 0.92604502 0.97029703 0.9376947  0.76470588]
Recall with SMOTE for DT: [0.97452229 0.9442623  0.95765472 0.9204893  0.74285714]

Precision with SMOTE for SVM: [0.8538961  0.84488449 0.88178914 0.87537092 0.77777778]
Recall with SMOTE for SVM: [0.83757962 0.83934426 0.8990228  0.90214067 0.6       ]
```

Με βάση τα αποτελέσματα βλέπουμε ότι το Recall για το DT και στα τρία πειράματα είναι σχεδόν το ίδιο ενώ για το SVM βελτιώνεται σημαντικά και με smote και με class weights. Συνολικά η καλύτερη λύση με βάση και το precision και το recall είναι αυτή του smote. Οπότε θα συνεχίσουμε

με αυτό. Να σημειωθεί ότι συνεχίζοντας με smote χάνουμε οριστικά την πληροφορία για τις υποκλάσεις της κλάσης U2R διότι να νέα samples δεν μπορούν να κατηγοριοποιηθούν με κάποιον τρόπο. Θα έπρεπε να κάνουμε smote σε κάθε υποκλάση. Το **script4** κάνει smote και σώζει το csv με το οποίο θα συνεχίσουμε.

4.FEATURE SELECTION AND EVALUATION

Εδώ θα προσπαθήσουμε να μειώσουμε το dimensionality εντοπίζοντας features που μπορούμε να παραλείψουμε για να πετύχουμε 2 στόχους. Ο πρώτος είναι να βελτιώσουμε τα metrics συνεχίζοντας αυτό που κάναμε στο προηγούμενο με το SMOTE. Ο δεύτερος είναι να γίνει πιο απλό το μοντέλο. Κοιτώντας τα δεδομένα βλέπουμε ότι έχουμε πολλά features που δίνουν count και άλλα αντίστοιχα που δίνουν rate. Περιμένουμε μετά το normalization αυτά τα ζεύγη να είναι συσχετισμένα δηλαδή να περιέχουν την ίδια πληροφορία πράγμα που σημαίνει ότι είναι πλεονάζοντα. Το πρώτο που θα δοκιμάσουμε είναι να εξάγουμε τα καλύτερα features κατευθείαν από το fit στο DT. Το δεύτερο θα είναι να κάνουμε correlation matrix στα features μόνο καθώς οι κλάσεις είναι categorical άρα δεν μπορούμε να δείξουμε συσχέτιση μεταξύ features/classes.
Για το DT βλέπουμε ότι το score των 20 σημαντικότερων features αθροίζει σε 0.93/1 συνεπώς έχουμε το μεγαλύτερο μέρος πληροφορίας(ισχύει??) στις μισές διαστάσεις. Τα αποτελέσματα είναι τα παρακάτω



Παρατηρούμε ότι δεν βρίσκονται όλα τα HASH features στην λίστα παρόλο που αντιπροσωπεύουν το ίδιο μέρος πληροφορίας. Γενικά πολλές απορίες για το hash trick. Αποφασίζουμε αυθαίρετα ότι δεν θα κρατήσουμε όλα τα hash features επειδή δεν είναι στην λίστα με τα 20.
Στην συνέχεια θα κάνουμε correlation matrix ώστε να βρούμε τις συσχετίσεις και να δούμε αν δίνει τα ίδια αποτελέσματα με την παραπάνω μέθοδο. Εδώ δεν μας είναι ξεκάθαρο το πως πρέπει να συνεχίσουμε. Φυσικά θα δημιουργήσουμε το correlation matrix αλλά μετά πώς θα επιλέξουμε ποιο feature να κρατήσουμε. Θα τα συγκρίνουμε ανά ζεύγη?Η θα υπολογίσουμε την συνολική συσχέτιση κάθε feature με όλα τα άλλα? Επειδή δεν είναι σαφές θα προχωρήσουμε με τα features που έδωσε η προηγούμενη μέθοδος αλλά θα δούμε και τι αποτελέσματα δίνει αυτή. Από θεωρία ξέρουμε ότι ψάχνουμε για την ελάχιστη συσχέτιση μεταξύ των features δηλαδή κοντά στο μηδέν. Άρα θα υπολογίσουμε για κάθε γραμμή ή στήλη του πίνακα το άθροισμα των συσχετίσεων και θα κρατήσουμε αυτά που έχουν το ελάχιστο. Δέν δίνουμε το heatmap σε αυτό το report διότι είναι δύσκολο να δεί κάποιος τις τιμές.

```
src_bytes                       1.394796
su_attempted                    1.521412
dst_bytes                       1.756959
is_host_login                   2.013514
num_failed_logins               2.776029
land                            3.478003
duration                        3.694490
is_guest_login                  3.960335
wrong_fragment                  4.450178
urgent                          4.711958
num_access_files                4.857532
hot                             5.128452
num_root                        5.143469
num_shells                      5.318806
srv_diff_host_rate              5.706270
srv_count                       5.721651
num_compromised                 5.813018
dst_host_srv_diff_host_rate     5.885422
num_file_creations              6.018610
HASH2                           6.222304
```

Παρατηρούμε ότι δεν έχει πολλά κοινά στοιχεία με αυτά που έχουμε από το προηγούμενο, πράγμα που μας κάνει να πιστεύουμε ότι είναι λάθος αλλά θα τα κρατήσουμε για πειραματισμό στο επόμενο βήμα. Σώζουμε τα δύο dataframes για το επόμενο βήμα.

Στην συνέχεια τρέχουμε πάλι τον DT/SVC στα CSVs που δημιουργήσαμε για να δούμε τι αποτελέσματα δίνουν. Το **script5** βρίσκει τα καλύτερα features που δίνει ο DT και σώζει το νέο csv. Το **script6** κάνει το ίδιο με correlation matrix. Το **scrip7** τρέχει τους DT/SVC με τα προηγούμενα δύο csvs(βάλτε το comment από το δεύτερο url). Εδώ το **script7** για το δεύτερο dataset δηλαδή αυτό που δημιουργήθηκε από τα features που διαλέξαμε με βάση το correlation matrix δίνει τα παρακάτω αποτελέσματα.

- Dataset with features from corr matrix

```
Precision for DT: [0.98095238 0.90056818 0.84210526 0.94909091 0.96273292]
Recall for DT: [0.94207317 0.92961877 0.95597484 0.83121019 0.95679012]


Precision for SVC: [0.95454545 0.58695652 0.59375    0.45790554 0.49082569]
Recall for SVC: [0.32012195 0.63343109 0.41823899 0.71019108 0.66049383]
```

Βλέπουμε ότι ο DT βελτιώνει την απόδοση του σε σχέση με το πρώτο πείραμα όπου δοκιμάσαμε με class weights και smote παρόλο που δεν χρησιμοποιήσαμε τα features που θεωρούσε σημαντικότερα. Ο SVC βλέπουμε ότι έχει πολύ κακή απόδοση και κάνει πολύ ώρα να εκτελεστεί(περίπου 1 ώρα). Για το dataset που προέκυψε με βάση τα features που θεωρεί πιο σημαντικά το DT έχουμε.

- Dataset with features from DT

```
Precision for DT: [0.98327759 0.94303797 0.97910448 0.93696275 0.9601227 ]
Recall for DT: [0.96710526 0.94904459 0.98498498 0.93965517 0.9601227 ]

Precision for SVC: [0.89090909 0.81730769 0.7581864  0.81184669 0.72616137]
Recall for SVC: [0.64473684 0.81210191 0.9039039  0.66954023 0.91104294]
```

Βλέπουμε ότι ο SVC δεν παέι καθόλου καλά και παρόλο που το recall για την minority class U2R(η τελευταία) έχει καλά αποτελέσματα οι κλάσεις DDoS και R2L(η πρώτη και η τρίτη) δεν έχουν πλέον καλά αποτελέσματα στο dataset που προέκυψε από το DT. Για το dataset που προέκυψε από το corr matrix έχουμε ακόμα χειρότερα αποτελέσματα. Να σημειωθεί εδώ ότι οι κλάσεις εμφανίζονται πάντα με την σειρά DDoS,Normal,Probe,R2L,U2R.
Για τον DT τρέχουμε και για τα δύο datasets πολύ καλά metrics για τα features που επιλέχθηκαν με βάση τον DT στο προηγούμενο βήμα,κάτι που είναι αναμενόμενο. Για το δεύτερο dataset που επιλέχθηκε με βάση το correlation βλέπουμε αποτελέσματα κάπου λιγότερο από 5% κάτω σε σχέση με το πρώτο dataset. Παρατηρούμε ότι η απόδοση ναι μεν έχει διαφορά αλλά όχι πολύ μεγάλη πράγμα που μας δίνει τη δυνατότητα να συμπεράνουμε ότι η μέθοδος που ακολουθήθηκε σχετικά με το correlation δεν είναι τελείως λάθος. Παρόλα αυτά θα συνεχίσουμε με τα features από τον αλγόριθμο DT γιατί δίνουν καλύτερα αποτελέσματα με βάση την προηγούμενη σύγκριση και γιατί ο SVC είναι πολύ αργός στο training και τα αποτελέσματα μετά το feature selection δεν είναι καθόλου καλά. Να σημειωθεί ότι εδώ θα μπορούσαμε να πάμε στο προηγούμενο βήμα να μειώσουμε επιπλέον τα features και να ξανατρέξουμε το πείραμα.

5.MODEL VALIDATION HYPERPARAMETERS TUNING

Εδώ θα κάνουμε το τελικό evaluation του μοντέλου. Με βάση τα αποτελέσματα από τα προηγούμενα θα επιλέξουμε το dataset με τα features που προέκυψαν από τον αλγόριθμο DT και ο αλγόριθμος που θα χρησιμοποιήσουμε είναι ο DT. Δέν θα δοκιμάσουμε τον SVC διότι αποδείχθηκε πολύ αργός στο training και τα metrics που είδαμε δεν ήταν καλά.
Οι βασικές hyperparameters για τον DT που θα εξετάσουμε είναι

- criterion gini/entropy
- splitter best/random
- max depth
- min samples split

Με αυτές τις παραμέτρους θα πειραματιστούμε χρησιμοποιώντας τον αλγόριθμο GridSearch. Οι δύο πρώτες παράμετροι δεν έχουν κάποια δυσκολία είναι απλά δύο τιμές. Για το max_depth από το script6 βλέπουμε ότι προκύπτουν τιμές γύρω στο 20 οπότε θα πειραματιστούμε με ένα εύρος τιμών γύρω από το 20. Μετά τον πειραματισμό έχουμε πλέον το καλύτερο μοντέλο και το σώζουμε για να το χρησιμοποιήσουμε στο επόμενο βήμα. Το script που έτρεξε είναι το **script8.**

6.MODEL EVALUATION AND RESULT EXPLANATION

Εδώ έχουμε κρατήσει από τα προηγούμενα βήματα ένα csv με μερικά datapoint για δοκιμή του αλγορίθμου. Το script που φορτώνει το μοντέλο και κάνει predict είναι το **script9** . Τέλος θα τρέξουμε το μοντέλο για όλο το dataset δηλαδή 5000 samples για να δούμε σε πόση ώρα επεξεργάζεται κάθε sample. Αυτό το κάνει το **script10.**Περίπου 1/1000 secs για 5000 samples σε τετραπύρηνο i5 χωρίς να υπολογίζουμε τον χρόνο που χρειάζεται για να μετατραπούν τα raw samples στην μορφή που δέχεται το μοντέλο που έχουμε σώσει, δηλαδή να γίνουν normalized τα columns, na γίνει το onehot και το feature hashing.

7.CONCLUSIONS

Θα μπορούσαμε να πειραματιστούμε περεταίρω σε πολλά στάδια της διαδικασίας που περιγράφηκε στα παραπάνω. Περισσότεροι αλγόριθμοι και πιο αναλυτική και στοχευμένη διαδικασία όσον αφορά το feature selection και evaluation του κάθε σεναρίου που θα εξετάζαμε. Στην βιβλιογραφία και επειδή αυτό το dataset είναι ένα από τα KDDCUP datasets έχουν πετύχει υψηλές αποδώσεις με κάτω από 10 features. Επίσης θα άξιζε να δούμε το σενάριο όπου κάνουμε δεύτερο classification στις υποκλάσεις αλλά θα έπρεπε να κάνουμε smote σε πολλά samples ώστε να πετύχουμε αυτό που πετύχαμε με τις high-level κλάσεις. Τα σημεία που κρατάμε είναι ότι καταφέραμε να αντιμετωπίσουμε το πρόβλημα class imbalance με χρήση smote και το feature engineering για τα categorical features όπου εφαρμόσαμε expert knowledge και hash trick.

REFERENCES

- https://stackabuse.com/cross-validation-and-grid-search-for-model-selection-in-python/
- https://www.nbshare.io/notebook/312837011/Decision-Tree-Regression-With-Hyper-Parameter-Tuning-In-Python/#:~:text=%20Decision%20Tree%20Regression%20With%20Hyper%20Parameter%20Tuning,sets.%20Feature%20selection%20by%20ExtraTreesRegressor%20%28model...%20More%20
- https://machinelearningmastery.com/one-hot-encoding-for-categorical-data/
- https://www.datacamp.com/community/tutorials/random-forests-classifier-python
- https://www.analyticsvidhya.com/blog/2021/05/how-to-perform-one-hot-encoding-for-multi-categorical-variables/
- https://www.geeksforgeeks.org/ml-one-hot-encoding-of-datasets-in-python/
- https://en.wikipedia.org/wiki/Feature_hashing#Feature_vectorization_using_the_hashing_trick
- https://www.analyticsvidhya.com/blog/2020/08/types-of-categorical-data-encoding/
- https://www.analyticsvidhya.com/blog/2015/11/easy-methods-deal-categorical-variables-predictive-modeling/?utm_source=blog&utm_medium=Categorical_data_encoding
- http://www.cs.columbia.edu/~hgs/teaching/security/slides/intrusion_detection_guest_lecture.pdf
- https://elitedatascience.com/imbalanced-classes
- https://towardsdatascience.com/feature-selection-techniques-in-machine-learning-with-python-f24e7da3f36e
- https://towardsdatascience.com/data-exploration-understanding-and-visualization-72657f5eac41
- https://towardsdatascience.com/feature-selection-correlation-and-p-value-da8921bfb3cf
- https://towardsdatascience.com/feature-selection-with-pandas-e3690ad8504b