

Relazione progetto per il corso di Big Data Analytics & Machine Learning

CLUSTERING DI NOMI DI DOMINIO GENERATI A PARTIRE DA DGA

ASCANI CHRISTIAN, SALVONI SIMONE, TISENI LORENZO

INDICE

INTRODUZIONE	3
Contesto.....	3
Obiettivo.....	5
MATERIALI E METODI.....	6
Descrizione del dataset	6
ETL	6
Embedding.....	8
Metodi e metriche.....	9
RISULTATI Sperimentali	13
Configurazione sperimentale.....	13
Primo esperimento	15
Analisi sulla miglior segmentazione	19
Analisi di medie e varianze delle feature	28
Analisi senza famiglie correttamente segmentate.....	36
CONCLUSIONI	39

INDICE DELLE FIGURE

Figura 1: Estratto di un feed del Bambenek Consulting	6
Figura 2: Estratto del dataset da segmentare	8
Figura 3: Risultati silhouette media primo esperimento con epochs=10	16
Figura 4: Risultati silhouette media primo esperimento con epochs=15	16
Figura 5: Risultati silhouette media primo esperimento con epochs=20	17
Figura 6: Risultati silhouette media con epochs=20, dim=3 e eps=2.75	18
Figura 7: Risultati omogeneità primo esperimento con epochs=20	18
Figura 8: Risultati completezza primo esperimento con epochs=20	19
Figura 9: Matrice di confusione con dim=4 minPoints=189 eps=3.0	20
Figura 10: Precision con dim=4 minPoints=180 eps=3.0	21
Figura 11: Recall con dim=4 minPoints=189 eps=3.0	21
Figura 12: Matrice di confusione con dim=3 minPoints=142 eps=2.75	22
Figura 13: Precision con dim=3 minPoints=142 eps=2.75	23
Figura 14: Recall con dim=3 minPoints=142 eps=2.75	23
Figura 15: Matrice di confusione con dim=2 minPoints=95 eps=2.25	24
Figura 16: Precision con dim=2 minPoints=95 eps=2.25	25
Figura 17: Recall con dim=2 minPoints=95 eps=2.25	25
Figura 18: Distribuzioni dei valori della silhouette per il cluster_noise	26
Figura 19: Distribuzioni dei valori della silhouette per il cluster_0	27
Figura 20: Distribuzione dei valori della silhouette per la famiglia bazarbackdoor	27
Figura 21: Distribuzioni dei valori della silhouette per i cluster contenenti elementi monerodownloader	28
Figura 22: Media e Varianza di "dyre" con dim=2	29
Figura 23: Media e Varianza di "dyre" con dim=3	30
Figura 24: Media e Varianza di "monerodownloader" con dim=2	31
Figura 25: Media e Varianza di "monerodownloader" con dim=3	31
Figura 26: Media e Varianza di "monerodownloader" con dim=4	32
Figura 27: Media e Varianza di "bazarbackdoor" con dim=3	33
Figura 28: Media e Varianza di "bazarbackdoor" con dim=4	33
Figura 29: Media e Varianza di "zeus-newgoz" con dim=2	34
Figura 30: Media e Varianza di "zeus-newgoz" con dim=3	35
Figura 31: Media e Varianza di "zeus-newgoz" con dim=4	35
Figura 32: Matrice di confusione dim=2 minPoints=95 eps=2.25	36
Figura 33: Matrice di confusione dim=3 minPoints=142 eps=2.75	37
Figura 34: Matrice di confusione dim=4 minPoints=189 eps=3.0	38

INTRODUZIONE

Contesto

Le grandi potenzialità offerte da Internet non sono esenti da controindicazioni e pericoli. Con l'aumentare del numero delle persone connesse alla rete, è aumentato anche il numero di bersagli che un attaccante può andare a minacciare. Infatti, il numero di attacchi nell'ultimo periodo è significativamente cresciuto, non solo alle macchine private, ma anche a sistemi appartenenti ad enti pubblici e governativi. Questo fenomeno ha dato la possibilità agli hacker di creare facilmente delle reti di computer zombie, che potessero essere sfruttate per compiere azioni criminose, e interrompere alcuni servizi cruciali. Oltretutto, è molto difficile individuare queste reti, poiché spesso i possessori delle macchine si accorgono troppo tardi di essere stati vittima di un attacco.

Oggi giorno ogni server, o servizio, della rete Internet è identificato da un indirizzo univoco (l'indirizzo IP) che permette all'utente di localizzarlo sul web, fargli delle richieste, e ottenere in risposta delle informazioni. Tuttavia, richiedere all'utente di internet l'utilizzo diretto degli indirizzi IP sarebbe fin troppo complesso. Per rendere la navigazione un processo più agevole è stato quindi ideato il *Domain Name System*, o DNS, il quale è un sistema utilizzato per assegnare nomi ai nodi della rete, che diventano identificatori alternativi all'IP. Più nello specifico, il DNS è un database distribuito in cui ad ogni indirizzo IP viene associato un nome di dominio, nel caso più semplice. I nomi di dominio, generalmente, sono delle stringhe separate da punti e organizzate a livelli. Tra questi il livello più importante è rappresentato dalla stringa che si incontra partendo da destra, che viene detta per questo *Top Level Domain* o TLD. Segue poi il dominio di secondo livello, che deve rispettare delle regole formali imposte, come avere una determinata lunghezza e contenere solo dei caratteri appartenenti ad un certo set. Il terzo livello a sua volta è detto dominio di terzo livello e nel caso generale può essere seguito anche da livelli superiori. Quando un utente digita il nome di dominio assegnato a un nodo della rete che vuole raggiungere per effettuare una richiesta, avviene un processo detto Risoluzione DNS, tramite il quale l'utente ottiene l'indirizzo IP associato a quel nome di dominio, a cui può quindi collegarsi.

Con l'uso sempre maggiore della rete internet, negli anni si sono diffusi codici malevoli capaci di danneggiare PC usati per lo più da utenti non

esperti. Se tutte queste azioni rimangono confinate ad un singolo PC, nel peggiore dei casi queste nuociono solo alla vittima, portando alla perdita totale o parziale dei suoi dati. Oltre a questo genere di attacchi, però, esiste anche un'altra tecnica: invece di infettare un solo computer, si cerca di infettare contemporaneamente più computer. Questo tipo di attacco porta il pirata informatico a costituire delle vere e proprie reti di "computer zombie", dette *botnet*, che possono essere sfruttate per compiere attività illecite anche contro terzi, e non direttamente contro i proprietari dei computer infettati. Colui che si occupa di gestire la *botnet*, che viene chiamato *botmaster*, invia dei comandi attraverso il cosiddetto C&C (*Command and Control*) Channel. Questa componente della *botnet* è fondamentale, in quanto permette all'attaccante di comunicare con le macchine infette e ricevere dati da esse. Per questo motivo senza tale componente la *botnet* diventa inutile, poiché il *botmaster* perde il contatto con le macchine controllate, le quali non sono più in grado di eseguire attacchi autonomamente. Perciò, chiudere il canale di controllo della *botnet* significa essenzialmente neutralizzarla. Un'altra caratteristica rilevante delle *botnet* è la loro struttura, da cui dipende anche il modo in cui il *botmaster* comunica con gli altri nodi. Essa può essere:

- Centralizzata, quando il server di comando e controllo è unico, e tutte le macchine appartenenti alla *botnet* sono connesse a questo unico nodo da cui ricevono i comandi.
- Decentralizzata, quando ogni *bot* che fa parte della *botnet* può essere utilizzato sia come client che come server di comando e controllo.
- Ibrida, quando combina i due approcci.

Il dominio per contattare il C&C server della *botnet* può essere inserito in maniera *hard-coded* all'interno del codice del *malware* oppure in dei file di configurazione che vengono scaricati nei *bot* insieme al *malware*. Questo però fa in modo che, se quel nodo da contattare cade, non è possibile per la *botnet* aggiornarsi per contattare un nuovo nodo. Per prevenire questo inconveniente, l'idea è stata quella di fare in modo che fossero i *bot* stessi a generare periodicamente un insieme di nomi di dominio, tra i quali si troverà quello che permetterà di raggiungere il C&C server. Gli algoritmi con cui i *bot* riescono a generare questi domini sono detti *Domain Generation Algorithm* (DGA). Con questi algoritmi viene generata una lista di nomi di dominio pseudo-casuali, i quali saranno validi per un certo intervallo di tempo. Allo stesso tempo il *botmaster*, che sa in anticipo come vengono generati i nomi, ne userà uno in quel lasso di

tempo per il suo C&C server; scaduto questo intervallo di tempo avviene la rigenerazione dei domini, e il processo riparte. Tuttavia, i DGA hanno anche degli svantaggi, tra cui il fatto che la generazione dei domini non è completamente casuale. Come ogni processo che in informatica genera un output casuale, anche la generazione dei domini dipenderà da un "seme", o comunque da un meccanismo pseudo-casuale. Perciò, facendo *reverse engineering* dell'algoritmo di generazione dei domini all'interno di un *bot*, è possibile cercare di prevedere in anticipo i nomi che saranno generati, compiendo delle azioni preventive per oscurare in anticipo i domini.

5

Obiettivo

L'obiettivo del lavoro qui presentato è studiare come sono strutturati i nomi di dominio prodotti da varie famiglie di DGA. Per fare ciò si è deciso di seguire l'approccio dell'apprendimento non supervisionato, in particolare il clustering. L'idea è cercare di capire se esistono delle somiglianze tra i nomi di dominio di alcune famiglie di malware, in maniera tale da poter accorpare le famiglie in un unico insieme e semplificare l'individuazione delle intrusioni.

Il lavoro svolto ha seguito i seguenti passaggi:

- Identificazione delle fonti da usare e conseguente costruzione del dataset da studiare
- Studio dei campioni in modo da comprendere come è formato il dataset
- Pulizia del rumore o di dati non interessanti per l'analisi
- Scelta della tecnica di *embedding* più consona fra quelle disponibili
- Scelta dell'algoritmo di clustering che soddisfi i criteri definiti in precedenza.
- Progettazione ed esecuzione dell'analisi vera a propria.

Grazie a tale analisi, saranno disponibili nuove informazioni sulle varie famiglie, per cui si potrà studiare singolarmente le classi di interesse in modo da capire il loro comportamento al variare dei parametri.

MATERIALI E METODI

Descrizione del dataset

Il dataset utilizzato per il task di clusterizzazione è stato costruito a partire da delle liste di nomi di dominio malevoli ottenuti da *Bambenek Consulting*. Questa organizzazione fornisce giornalmente una lista di nomi di dominio, prodotti tramite DGA, rilevati in rete e che sono classificati in base alla famiglia di DGA che li ha generati. Per ciascun file è inoltre fornita una stima temporale della validità dei nomi di dominio in esso presenti. Infatti, come descritto nell'introduzione, i nomi di dominio prodotti da DGA hanno una vita limitata, scaduta la quale questi vengono sostituiti. Per questo motivo il *Bambenek Consulting* inserisce un nome di dominio all'interno del feed per cinque giorni consecutivi, da due giorni prima che venga generato fino a tre gironi dopo la generazione. In questa maniera il feed permette a chi lo ottiene sia di registrare attacchi in corso, sia di analizzare un attacco avvenuto nei giorni precedenti. Questa particolarità dei *feed* forniti da *Bambenek* viene gestita in fase di ETL. I feed sono strutturati come in Figura 1 e sono stati raccolti per il periodo che va dal 20/07/2022 al 5/10/2022. Tutti questi file sono stati poi letti per ottenere i dati accumulati necessari per il dataset.

```
pmffjddxxjgoa.com,Domain used by Cryptolocker - Flashback DGA for 25 Jul 2022,2022-07-25,http://osint.bambenekconsulting.com/manual/cl.txt  
dakujicluwap.net,Domain used by Cryptolocker - Flashback DGA for 25 Jul 2022,2022-07-25,http://osint.bambenekconsulting.com/manual/cl.txt  
udhssgwiauom.biz,Domain used by Cryptolocker - Flashback DGA for 25 Jul 2022,2022-07-25,http://osint.bambenekconsulting.com/manual/cl.txt  
vfmcrlndlwggg.ru,Domain used by Cryptolocker - Flashback DGA for 25 Jul 2022,2022-07-25,http://osint.bambenekconsulting.com/manual/cl.txt  
vujuewogoscr.org,Domain used by Cryptolocker - Flashback DGA for 25 Jul 2022,2022-07-25,http://osint.bambenekconsulting.com/manual/cl.txt  
wwotaculautl.co.uk,Domain used by Cryptolocker - Flashback DGA for 25 Jul 2022,2022-07-25,http://osint.bambenekconsulting.com/manual/cl.txt  
wllhgbngdbok.info,Domain used by Cryptolocker - Flashback DGA for 25 Jul 2022,2022-07-25,http://osint.bambenekconsulting.com/manual/cl.txt  
xnqccgtlodge.com,Domain used by Cryptolocker - Flashback DGA for 25 Jul 2022,2022-07-25,http://osint.bambenekconsulting.com/manual/cl.txt  
xdnjerferycp.net,Domain used by Cryptolocker - Flashback DGA for 25 Jul 2022,2022-07-25,http://osint.bambenekconsulting.com/manual/cl.txt  
yfsiawljdbtj.biz,Domain used by Cryptolocker - Flashback DGA for 25 Jul 2022,2022-07-25,http://osint.bambenekconsulting.com/manual/cl.txt  
bseiiwoicyedn.ru,Domain used by Cryptolocker - Flashback DGA for 25 Jul 2022,2022-07-25,http://osint.bambenekconsulting.com/manual/cl.txt  
ogjtdwvjvadb.org,Domain used by Cryptolocker - Flashback DGA for 25 Jul 2022,2022-07-25,http://osint.bambenekconsulting.com/manual/cl.txt  
dngjefamlmtq.co.uk,Domain used by Cryptolocker - Flashback DGA for 25 Jul 2022,2022-07-25,http://osint.bambenekconsulting.com/manual/cl.txt  
qbluknnntiitk.info,Domain used by Cryptolocker - Flashback DGA for 25 Jul 2022,2022-07-25,http://osint.bambenekconsulting.com/manual/cl.txt  
dbiwwjyackdf.com,Domain used by Cryptolocker - Flashback DGA for 25 Jul 2022,2022-07-25,http://osint.bambenekconsulting.com/manual/cl.txt  
qonidrmhygy.net,Domain used by Cryptolocker - Flashback DGA for 25 Jul 2022,2022-07-25,http://osint.bambenekconsulting.com/manual/cl.txt  
fvkxeaqkosto.biz,Domain used by Cryptolocker - Flashback DGA for 25 Jul 2022,2022-07-25,http://osint.bambenekconsulting.com/manual/cl.txt  
sjpkierloti.ru,Domain used by Cryptolocker - Flashback DGA for 25 Jul 2022,2022-07-25,http://osint.bambenekconsulting.com/manual/cl.txt  
jjmixdpthwfb.org,Domain used by Cryptolocker - Flashback DGA for 25 Jul 2022,2022-07-25,http://osint.bambenekconsulting.com/manual/cl.txt  
klrhlfvveiq.co.uk,Domain used by Cryptolocker - Flashback DGA for 25 Jul 2022,2022-07-25,http://osint.bambenekconsulting.com/manual/cl.txt
```

Figura 1: Estratto di un feed del Bambenek Consulting

ETL

A partire dai *feed* di *Bambenek*, per costruire il dataset richiesto è stato necessario processare tutti i feed raccolti nel tempo in maniera tale da ottenere un dataset in cui ogni riga contenesse un nome di dominio e l'etichetta di classe associata. Per la lettura e gestione dei dati sono state utilizzate le funzionalità di *PySpark*, l'implementazione di *Spark* sul linguaggio di programmazione *Python*.

Per prima cosa, dovendo lavorare con molti feed collezionati in un arco di tempo molto lungo, si è pensato di implementare un processo di costruzione incrementale del dataset. Infatti, mano a mano che si collezionavano i dati dal *Bambenek Consulting*, ad intervalli di tempo prestabiliti, questi venivano processati e ripuliti, in maniera tale che potessero entrare a far parte del dataset finale per il clustering. Quindi, ogni cinque giorni si è rilanciato periodicamente il processo di pulizia dei dati all'interno dei cinque feed, per poi accodarli a quelli già processati in precedenza in un unico file.

Mano a mano che i domini venivano accodati per formare il dataset complessivo, si rendeva necessario eliminare i duplicati che potevano esistere tra i feed collezionati, per avere un dataset privo di ripetizioni. Per questo si eliminavano i duplicati esistenti nei cinque feed considerati al momento del processing, data la possibile presenza di uno stesso nome di dominio in più feed di *Bambenek* successivi. Successivamente, ottenuta la lista di nomi di dominio senza ripetizioni, associata ai cinque feed processati, si è proceduto ad eliminare da questa i nomi di dominio già presenti nel file unico. Infatti, nell'ultimo feed della serie processata allo step precedente non si può escludere che siano presenti nomi appena inseriti, che quindi saranno inseriti anche nei feed processati allo step corrente. In questa maniera il dataset finale non presenta elementi duplicati.

Per quanto riguarda il processing del singolo feed, per prima cosa viene assegnata a ciascun nome di dominio un'etichetta di classe che corrisponde alla famiglia estratta dalla descrizione del nome. In questo procedimento si tiene conto del fatto che le famiglie di DGA possono avere più nomi, che sono tra loro degli alias; ad esempio, *wiki25* è un alias di *cryptolocker*, mentre *ptgoz* è un alias di *zeus-newgoz*. In questi casi, le etichette di classe sono state modificate per avere nel dataset una corrispondenza uno ad uno tra etichetta di classe e famiglia. Successivamente, si sono scartate le colonne relative alla descrizione del nome e al riferimento al manuale, conservando solo nome di dominio ed etichetta di classe. A questo punto, ogni nome di domino è stato sostituito dai bigrammi ricavati dalla stringa formata dal second level domain e dal top level domain, concatenati senza spazio. Ad esempio, *mionome.com* viene trasformato in *mionomecom* e successivamente viene sostituito con la lista di bigrammi [*mi*, *io*, *on*, *no*, *om*, *me*, *ec*, *co*, *om*].

Il dataset così costruito, in cui ogni riga contiene due campi, ovvero l'etichetta di classe e la lista di bigrammi, è salvato in formato csv e strutturato come in Figura 2. Esso è composto da tre milioni di record circa.

```
cryptolocker,ly yk kk kg gn nf fm mp pv vb br ra ao or rg
zeus-newgoz,1b bw wb b6 6w wf fr rf fy yo ov ve e1 1i iv va af fy yy y3 3u uj jt tj je eo or rg
dyre,f3 36 69 91 14 43 36 6e e5 52 23 37 73 35 54 4f f1 18 8f f9 98 84 4f f3 35 51 15 56 62 29 9a a4 40 0i in
zeus-newgoz,1d dc cj jw wi iq qe em mr rf f1 1l 11 1s s1 lh hd dc c7 71 1m mw w2 2f ff fm mn ne et
zeus-newgoz,2v vj jp p7 7q qs ss sn n7 7z zj 7n n3 3j jj jd d3 3x xa ah h1 15 5c co om
dyre,18 89 9f f8 8e ee e8 83 33 38 8c c5 54 4d d2 22 20 02 22 2d dc c8 83 3e ec c2 20 0a ad d5 50 09 99 9w ws
zeus-newgoz,5b by yd du u0 0d d4 4o op p4 4y y3 30 0h ht tv vs su uq qn nd d3 3n nn ne et
zeus-newgoz,43 3z zy ye ec co ot tt tg g9 9u u1 1n n1 1o om mx x7 71 1p pr ru u1 1q q1 io or rg
zeus-newgoz,19 9x xv ve ed dy y1 16 68 8t tk kf fz z1 1o ow wh hj jj jd dm my yw wz za a6 6n ne et
zeus-newgoz,1x xd di ii i8 8n ni ik kd dw wt tr r4 4v v8 8p pm mk k2 2i ie ed d6 6n ne et
geodo,xn nu ui io ow wa ar ry yw ww wh hw wc ct tj je eu
symmi,di ic cu uu ur rl le ed dd dn ns sn ne et
zeus-newgoz,ns sn nj jq qr rd d8 80 06 6c c5 5f f9 9g g2 2p po o1 1x xy y7 7e ek kc cc co om
```

Figura 2: Estratto del dataset da segmentare

Embedding

Per poter eseguire il clustering, è necessario uno strato di *embedding* che converta gli input in vettori che l'algoritmo sia in grado di processare. Infatti, l'*embedding* consiste in una mappatura dell'input, tramite la quale si rappresentano delle variabili discrete (in questo caso nomi di dominio) come vettori continui che possono essere collocati in uno spazio vettoriale.

Ci sono diversi tipi di *embedding*, tra cui quello randomico, in cui si associano in maniera casuale i caratteri presenti in tutto il dataset ad un numero casuale e poi si trasformano tutti i nomi di dominio presenti nel dataset in vettori numerici, con il mapping definito. Tuttavia, anche se tale modalità è molto semplice e veloce, le prestazioni spesso non sono ottimali. Per questo è stata scelta un'altra tipologia di *embedding*, ovvero FastText, che è un'estensione di un'architettura, *Word2Vec*, rilasciata da Facebook AI Research nel 2016. Esso è una libreria open-source il cui obiettivo è apprendere da solo, dato un grande corpus di frasi, le rappresentazioni vettoriali continue per le parole contenute nel testo.

Rispetto ad altre tecniche, FastText è noto in quanto, per creare il vettore da associare ad una parola, considera la rappresentazione interna della parola stessa, dividendola in n-grammi. Infatti, presa una parola, dopo averla divisa in n-grammi, ricava le rappresentazioni vettoriali degli n-grammi che la formano, per poi combinarle al fine di ottenere un vettore continuo che rappresenta una parola nello spazio. In questa maniera si ottengono rappresentazioni migliori delle parole e si riescono a costruire rappresentazioni di parole mai viste nel dataset, come somma di

n-grammi. Oltre a questo, FastText possiede il vantaggio di poter personalizzare la fase di training scegliendo vari parametri e il corpus di addestramento, il che si è rivelato molto utile nell’ambito dei domain name. Per questi, infatti, non è possibile rilevare un contesto sensato da sfruttare per la vettorizzazione, rendendo inutilizzabili altre tecniche di *embedding* pre-addestrate e word-based.

Il risultato dell’addestramento di FastText è un modello contenente tutte le rappresentazioni delle parole, trovate nel corpus di training, come vettori a n componenti. Tali modelli possono essere usati in fase di *embedding* per tradurre le parole in rappresentazioni vettoriali continue. Inoltre, FastText fornisce due diversi modelli per calcolare le rappresentazioni delle parole: *skipgram* e *cbow*. Il modello *skipgram* impara a prevedere una parola target grazie ad una parola vicina. Invece, il modello *cbow* prevede la parola target in base al suo contesto, che è rappresentato come un contenitore di parole, all’interno di una finestra con dimensioni fisse, attorno alla parola target.

Nell’esperimento, quindi, è stato utilizzato FastText come tecnica di *embedding*, e non si sono usate le parole intere, bensì i bigrammi, creando un corpus di addestramento, con tutti i vettori di bigrammi associati ai nomi nel dataset. Il corpus ha tante righe quanti sono i nomi del dataset, e ogni riga contiene la rappresentazione in bigrammi del nome di dominio. Come modello di addestramento è stato scelto lo *skipgram*, perché non esistono sostanziali differenze per tempo di esecuzione con il modello *cbow*. Inoltre, non è molto coerente utilizzare il *cbow*, perché non è possibile individuare un contesto sensato per i bigrammi, all’interno di un nome di dominio. In seguito all’addestramento, ogni parola viene rappresentata dal vettore risultante dalla concatenazione dei singoli vettori associati ai bigrammi che la compongono. Questo vettore, per via dell’isomorfismo fra $\mathbb{R}^{m \cdot n}$ e $M_{m \times n}(\mathbb{R})$, è equivalente alla matrice che ha come colonne i vettori associati ai vari bigrammi. Data la maggiore efficienza nei calcoli da eseguire con la rappresentazione vettoriale, si è scelto di concatenare i vettori piuttosto che lavorare con una rappresentazione matriciale.

Metodi e metriche

Fra le varie tecniche di clustering, è stato selezionato il *Density-Based Spatial Clustering of Applications with Noise* (DBSCAN). Il DBSCAN è una metodologia basata su densità: il cluster è definito dai punti che ne fanno parte, e non da un punto particolare che funge da prototipo, come in altri

algoritmi (per esempio, il K-Means). È stato scelto un metodo di clustering basato su densità per due motivi:

- Avendo collezionato dati provenienti dalla rete, non è noto a priori il numero di classi all'interno del periodo di osservazione, il che rende scorretto utilizzare un algoritmo in cui si specifica a priori il numero di cluster.
- La distribuzione dei punti costruiti dall'*embedding* di FastText è ignota, ed è più corretto affidarsi ad un algoritmo che non fa ipotesi a priori sulla forma dei cluster nello spazio, come il DBSCAN.

L'algoritmo DBSCAN basa la costruzione dei cluster su due parametri fondamentali, che vengono solitamente definiti ***minPoints*** ed ***eps* (ϵ)**. Generalmente, nel DBSCAN si classificano i punti come ***core point***, ***border point*** e ***noise point***. I *core point* sono quei punti che hanno nel loro intorno di raggio *eps* almeno altri *minPoints* punti. I *border point* sono quei punti che si trovano nell'intorno di almeno un *core point*, ma non sono a loro volta *core point*. Infine, i *noise point* sono i punti rimanenti (e quindi outlier).

Il funzionamento dell'algoritmo è quindi il seguente:

- Si parte da un punto qualunque del dataset. La scelta è totalmente arbitraria e non influenza né il risultato né l'efficienza dell'algoritmo.
- Si controlla l'intorno di raggio *eps* del punto scelto.
 - Se non ci sono almeno *minPoints* punti, allora si passa al punto successivo.
 - Se invece è presente un numero di punti maggiore o uguale a *minPoints*, allora si è trovato un nuovo cluster e il punto viene quindi classificato come *core point*. Si esegue quindi lo stesso controllo su tutti i punti trovati nell'intorno del *core point*. Se nel loro intorno si trovano almeno *minPoints* punti, allora quelli sono a loro volta *core point* e questo processo continua con i punti nel loro intorno. Altrimenti, il punto è classificato come *border point*. In ogni caso, tutti questi punti contribuiscono alla formazione del cluster individuato. Il processo si completa quando non si hanno più nuovi *core point*, ottenendo un cluster.
- Si passa ad un nuovo punto non precedentemente controllato, e si ripete il processo fino ad aver controllato tutti i punti. Tutti quei punti che con questo processo non vengono classificati né come *core*

point né come *border point* vengono classificati come *noise point*, e non faranno propriamente parte di nessun cluster.

Nell'algoritmo di clustering, come metrica per la misura della distanza tra i vettori, associati alle parole, viene utilizzata la distanza euclidea, poiché è quella computazionalmente più immediata e di più facile comprensione.

Per la valutazione del clustering, sono state utilizzate metriche apposite, sia supervisionate sia non supervisionate, sfruttando funzionalità messe a disposizione dalla libreria *sklearn*. Le tre metriche supervisionate sono:

- L'**omogeneità**, o **homogeneity**: una segmentazione è perfettamente omogenea se ogni cluster ha campioni appartenenti ad una sola classe. Tale misura viene sfruttata perché tiene conto della distribuzione degli elementi nelle classi e dell'entropia.
- La **completezza**, o **completeness**: una segmentazione è perfettamente completa se tutti i campioni appartenenti ad una certa classe si trovano nello stesso cluster. Come la precedente anche questa tiene conto della distribuzione delle classi e dell'entropia.
- La **v-measure**: rappresenta la media armonica tra omogeneità e completezza

La metrica non supervisionata scelta è la *silhouette*, che esprime, dato un punto x del dataset, se questo è stato assegnato al cluster corretto, oppure dovrebbe essere assegnato a un altro cluster. Tale metrica dà per ogni punto una misura della bontà della segmentazione. Essa per ogni punto calcola quanto è vicino agli elementi appartenenti al suo stesso cluster, e quanto è lontano dagli elementi di altri cluster. La *silhouette*, definita con $a(x)$ la distanza media intra-cluster riferita al punto x e con $b(x)$ la distanza media inter-cluster riferita al punto x , si calcola come:

$$\frac{b(x) - a(x)}{\max(a(x), b(x))}$$

In generale la *silhouette*, per un dato punto x , deve essere prossima al valore 1, che indica che il punto x è molto vicino agli elementi del suo cluster e molto lontano dagli elementi di altri. In caso contrario, se è negativa, la clusterizzazione è errata e, se è nulla, non si può dire se la clusterizzazione sia corretta o meno. Per la valutazione degli esperimenti, specialmente quelli iniziali, si utilizzerà la relativa funzione di *sklearn* che calcola il valore medio della *silhouette* per tutti i punti, compresi quelli nel cluster rumore.

Per la valutazione delle singole segmentazioni, sono state considerate anche altre due metriche supervisionate:

- La *precision*, che valuta quanto un cluster è singolarmente omogeneo. Essa è data dal rapporto tra gli elementi della famiglia j nel cluster i e tutti gli elementi del cluster i .
- La *recall*, che misura la capacità del cluster di rappresentare una famiglia. Essa è data dal rapporto tra gli elementi di famiglia j nel cluster i e quelli appartenenti alla famiglia j .

RISULTATI Sperimentali

Configurazione sperimentale

Definita la tecnica di *embedding* per ricavare un vettore di feature per ogni bigramma nel dataset considerato, e scelto l'algoritmo di clustering DBSCAN, si è definita una griglia di esperimenti sulla base dei parametri da impostare sia nell'*embedding* sia nell'algoritmo di clustering. I parametri liberi, di cui si rendeva necessario fare il tuning al fine di individuare la migliore clusterizzazione, sono i seguenti:

- *epochs*: il numero di epoche per l'addestramento dello strato di *embedding* basato su FastText.
- *dim*: la dimensione del vettore di feature associato a un bigramma del corpus di addestramento.
- *minPoints*: il numero minimo di punti che devono appartenere all'intorno di un punto in esame nell'algoritmo DBSCAN, affinché tale punto sia considerato core point.
- *eps*: il raggio dell'ipersfera che rappresenta l'intorno di un punto nell'algoritmo DBSCAN.

Basandosi sulla documentazione di FastText e sulle best practice da essa suggerite, si è ipotizzato di condurre gli esperimenti facendo assumere al parametro *epochs* i valori dieci, quindici e venti, e al parametro *dim* i valori [100, 150, 200, 250, 300]. Per quanto riguarda il parametro *minPoints*, detto n il numero di features, facendo riferimento alla regola empirica che impone:

- $\text{minPoints} \geq n + 1$ per dataset con rumore moderato
- $\text{minPoints} \geq 2 \cdot n$ per dataset molto rumoroso

si è deciso di valutare valori di *minPoints* compresi nell'intervallo $[n, \frac{6}{5} \cdot n, \frac{7}{5} \cdot n, \frac{8}{5} \cdot n, \frac{9}{5} \cdot n, 2 \cdot n]$. Il numero di features n dipende dalla dimensione dei vettori estratti da FastText, *dim*, e dalla lunghezza massima delle stringhe presenti nel dataset, *maxLen*, secondo la relazione $n = \text{dim} \cdot \text{maxLen}$. L'idea alla base della scelta dell'intervallo, non sapendo a priori quale fosse il rumore o la sparsità dei punti del dataset, è cercare di esplorare uno spazio di ricerca in cui ci siano soluzioni il più possibile robuste al rumore. Infine, si è deciso di determinare il valore di *eps* per ogni combinazione dei tre parametri precedenti, utilizzando una tecnica empirica, in maniera tale da ridurre la

grandezza dello spazio di ricerca. Tale tecnica consiste nel calcolare le distanze tra tutti i punti del dataset e selezionare per ciascuno di essi il k-esimo elemento più vicino, supposto $k = minPoints$. Le distanze dal k-esimo elemento di ogni punto vengono quindi graficate in ordine crescente (o decrescente). L'eps più adatto per quel $minPoints$ si va a ricercare nella regione nella quale il grafico presenta un cambio repentino di curvatura.

Definito questo setup sperimentale, si sono analizzate la cardinalità del dataset e la dimensione dei vettori di features ottenuti per ogni parola all'interno del dataset. Vista l'onerosità computazionale dell'algoritmo DBSCAN, pari a $O(n^2)$, si è constatato che non si aveva a disposizione una macchina abbastanza potente per portare a termine gli esperimenti. Per questo, si è rivisto lo spazio di ricerca determinando nuovi intervalli dei parametri liberi, adatti per i limiti di memoria e potenza di calcolo di una macchina *general purpose* con 12 GB di RAM. Quindi, si è ridefinito l'intervallo di variazione del parametro dim , come $[2, 4, 6, 8, 10]$ e si è determinata una serie di intervalli di variazione per il parametro eps , tutti con forma $\left[\frac{1}{5}eps_{max}, \frac{2}{5}eps_{max}, \frac{3}{5}eps_{max}, \frac{4}{5}eps_{max}, eps_{max}\right]$, dove:

- Per $dim = 2, eps_{max} = 2.25$
- Per $dim = 4, eps_{max} = 3$
- Per $dim = 6, eps_{max} = 4$
- Per $dim = 8, eps_{max} = 4.5$
- Per $dim = 10, eps_{max} = 5$

Nel precedente intervallo il valore eps_{max} rappresenta il valore massimo di eps collegato a una certa dimensione, e viene scelto in maniera tale da rispettare i limiti computazionali delle macchine a disposizione. Il motivo per cui si è optato per scegliere diversi intervalli di eps , al variare del valore del parametro dim , dipende dal fatto che variando dim varia il numero di features di ogni elemento del dataset. Se dim aumenta, cresce anche il numero di feature degli elementi del dataset, il che porta ad un aumento delle distanze tra i punti esistenti nello spazio \mathbb{R}^n . Quindi, scegliere lo stesso intervallo per ogni dimensione avrebbe potuto causare una degradazione dei risultati, in quanto l'utilizzo di valori troppo piccoli o troppo grandi di eps porta a situazioni degeneri. In questi casi o si verifica che tutto il dataset viene inserito nel cluster noise oppure è individuato un unico cluster corrispondente con tutto il dataset.

Oltre alla definizione dei nuovi intervalli di variazione dei parametri liberi, si è operato anche un ridimensionamento della cardinalità del dataset, riducendolo da tre milioni di elementi a circa centomila elementi. Tale operazione non è stata fatta solo per ridurre il carico delle strutture dati in memoria, ma anche per adeguare la dimensione del dataset al parametro *minPoints*. Infatti, riducendo il numero di features dei vettori estratti da FastText per ogni bigramma, si riduce anche il parametro *minPoints*, che però allo stesso tempo deve avere sempre un valore tale da essere almeno di due ordini di grandezza inferiore alla cardinalità del dataset. Quindi, nella riduzione della cardinalità del dataset si è considerato anche questo aspetto, operando quindi un *undersampling*, in maniera tale da rispettare la distribuzione iniziale della popolazione, e soddisfacendo così sia i limiti computazionali sia il vincolo su *minPoints*.

Terminata la definizione degli intervalli di variazione dei parametri liberi, si è sviluppato un algoritmo *Python* che realizzasse il processo di clustering dei nomi di dominio. Tale algoritmo comprende tre fasi:

- **Pre-processamento dei dati:** si colleziona il dataset, si estraggono tutte le etichette di classe e si assegna ad ognuna di esse un numero intero. Successivamente si rileva la lunghezza massima delle stringhe nel dataset.
- **Embedding:** si applica il processo di *embedding*, definito nella sezione omonima, e, ottenuti i vettori di features per ogni parola, si applica un *padding* di zeri ad ogni vettore, in maniera da fornire all'algoritmo di clustering vettori di uguale lunghezza.
- **Clustering e calcolo delle metriche:** si applica l'algoritmo di clustering DBSCAN e poi si calcolano le metriche per la valutazione della segmentazione, ovvero omogeneità, completezza, v-measure e silhouette media.

Primo esperimento

Definite la configurazione sperimentale e la serie di esperimenti da condurre, si è lanciato l'algoritmo progettato per tutti i valori dei parametri scelti, ottenendo i risultati delle metriche presentati in [Figura 3](#), [Figura 4](#), [Figura 5](#). Le figure sono delle mappe di calore in cui il blu rappresenta i valori più alti, il verde quelli più bassi, e le caselle grigie dei valori mancanti dovuti a casi degeneri. Confrontando i risultati ottenuti nelle tre figure per i tre valori del parametro *epochs*, si rileva subito che all'aumentare del parametro non c'è un significativo incremento dei risultati, ma c'è

sicuramente un incremento del tempo di esecuzione. Questo aumento non incide per le dimensioni del dataset in oggetto, però per dataset grandi diventa significativo. Al variare del parametro dim , invece, si rilevano delle variazioni significative per tutte e tre le metriche considerate, ovvero omogeneità, completezza e silhouette media dei cluster.

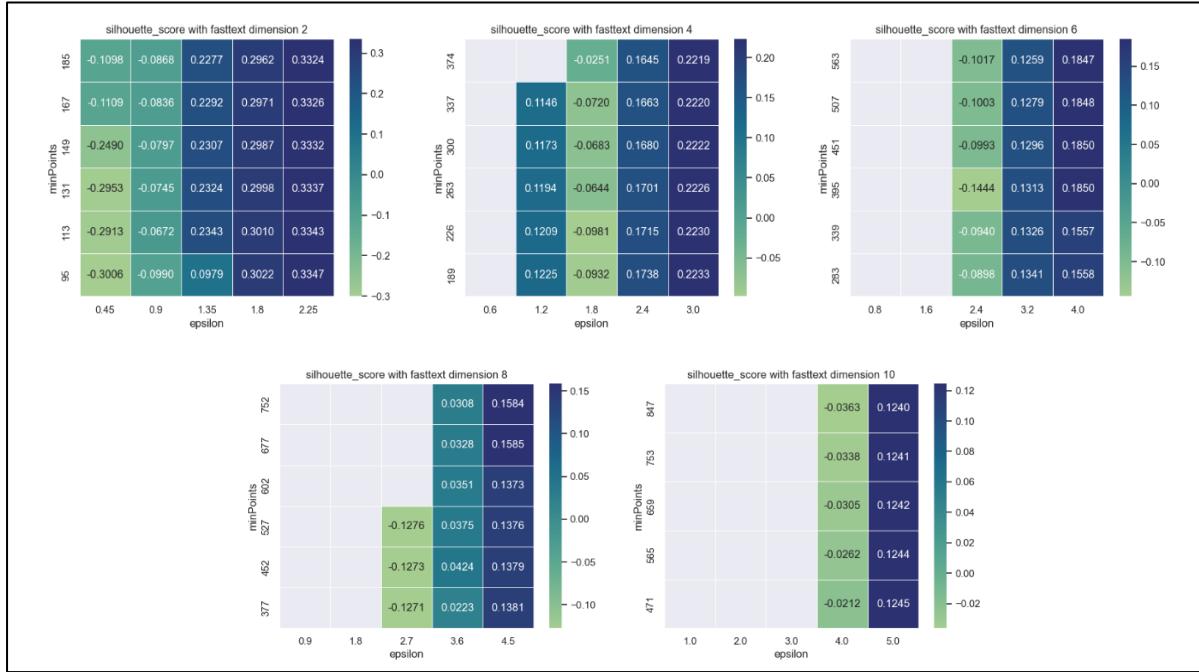


Figura 3: Risultati silhouette media primo esperimento con epochs=10

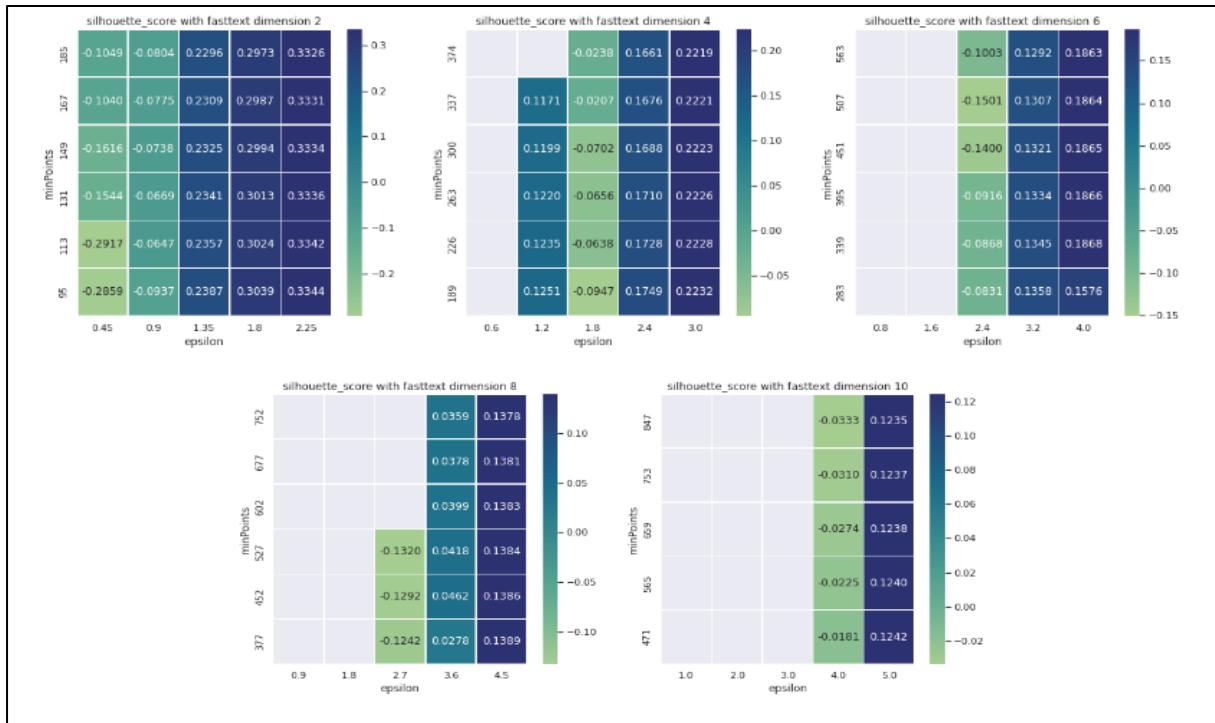


Figura 4: Risultati silhouette media primo esperimento con epochs=15

Analizzando solo la *silhouette* media, si può constatare che, all'aumentare del numero di features estratte da FastText, per ogni bigramma, la qualità media dei cluster peggiora a parità di valori dei parametri *minPoints* ed *eps*. Inoltre, considerando un singolo valore del parametro *dim* alla volta, si può osservare come in generale la qualità della clusterizzazione migliori all'aumentare del valore del parametro *eps*. Invece, al variare del parametro *minPoints*, non si riesce dalle mappe in oggetto a registrare una tendenza chiara verso un miglioramento o un peggioramento della *silhouette* media.

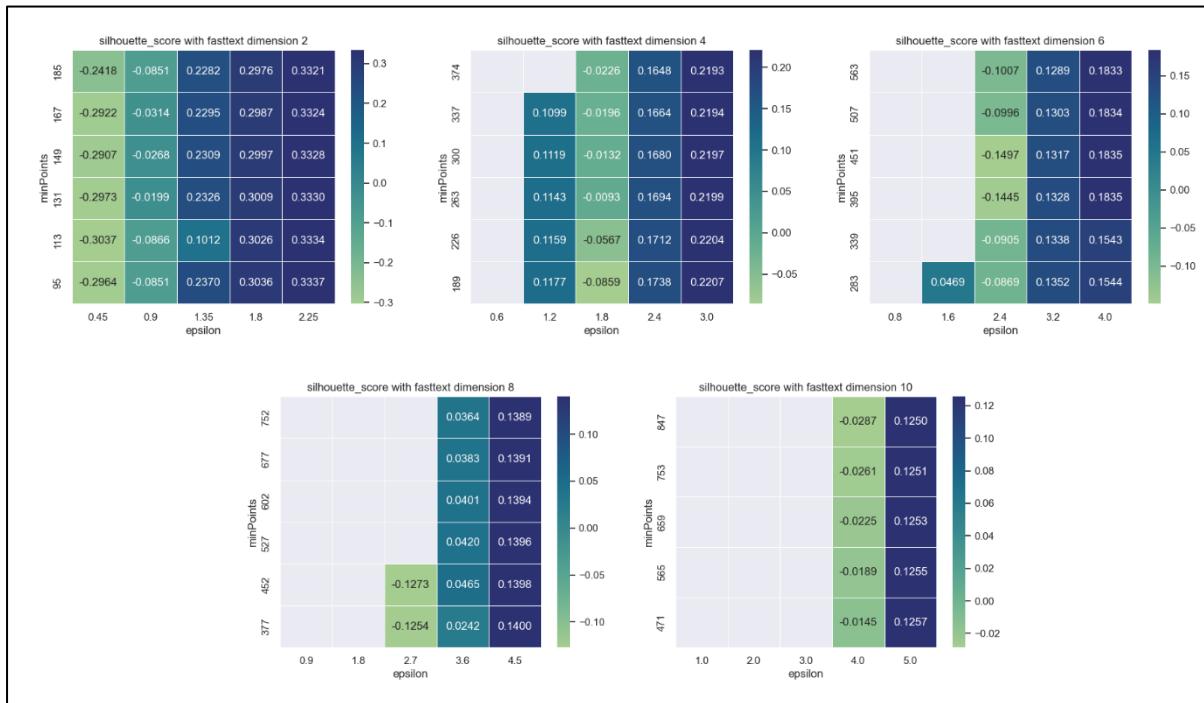
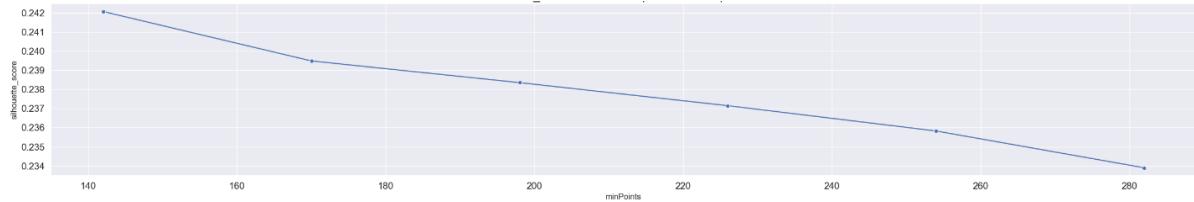


Figura 5: Risultati silhouette media primo esperimento con epochs=20

Per quanto detto prima, il valore relativamente basso della *silhouette* nei diversi casi, che indica una mediocre qualità della segmentazione, è dovuto al fatto che i punti finiti all'interno del cluster rumore sono stati mal clusterizzati.

Osservando i risultati ottenuti, si è notato che tra i valori 2 e 4 per il parametro *dim* si avevano i migliori valori di silhouette possibili per tutti gli esperimenti svolti, accettabili alla luce delle considerazioni fatte sul rumore. Quindi, si è deciso di approfondire lo studio nell'intervallo composto dai valori interi [2,3,4]. Innanzitutto, tenendo conto delle tendenze registrate con i primi esperimenti, si è rieseguito l'algoritmo per il valore del parametro *dim* pari a tre, considerando solo il valore di *eps* più alto che si possa avere, e valori di *minPoints* nello stesso intervallo

utilizzato in precedenza. Il processo ha fornito i risultati della silhouette media riportati nella [Figura 6](#):



18

Figura 6: Risultati silhouette media con epochs=20, dim=3 e eps=2.75

Nella figura è chiaramente osservabile che all'aumentare del valore del parametro *minPoints*, la silhouette media ha una leggera diminuzione. Invece, globalmente si conferma la tendenza della silhouette a calare, all'aumentare del numero di features estratte da FastText per ogni bigramma. In questo caso non si è reso necessario indagare tutti i possibili valori di *eps*, per la dimensione tre, in quanto esso rappresentava l'unico valore intero presente tra due e quattro, e la tendenza era sufficientemente chiara dai precedenti risultati.

Individuate le migliori clusterizzazioni, si è proceduto a valutare per tutti i valori dei parametri liberi considerati le metriche supervised, ovvero omogeneità e completezza, riportate all'interno delle mappe in [Figura 7](#) e [Figura 8](#). In esse sono riportate le metriche solo per *epoch*=20, in quanto per gli altri valori esse non variano di molto.

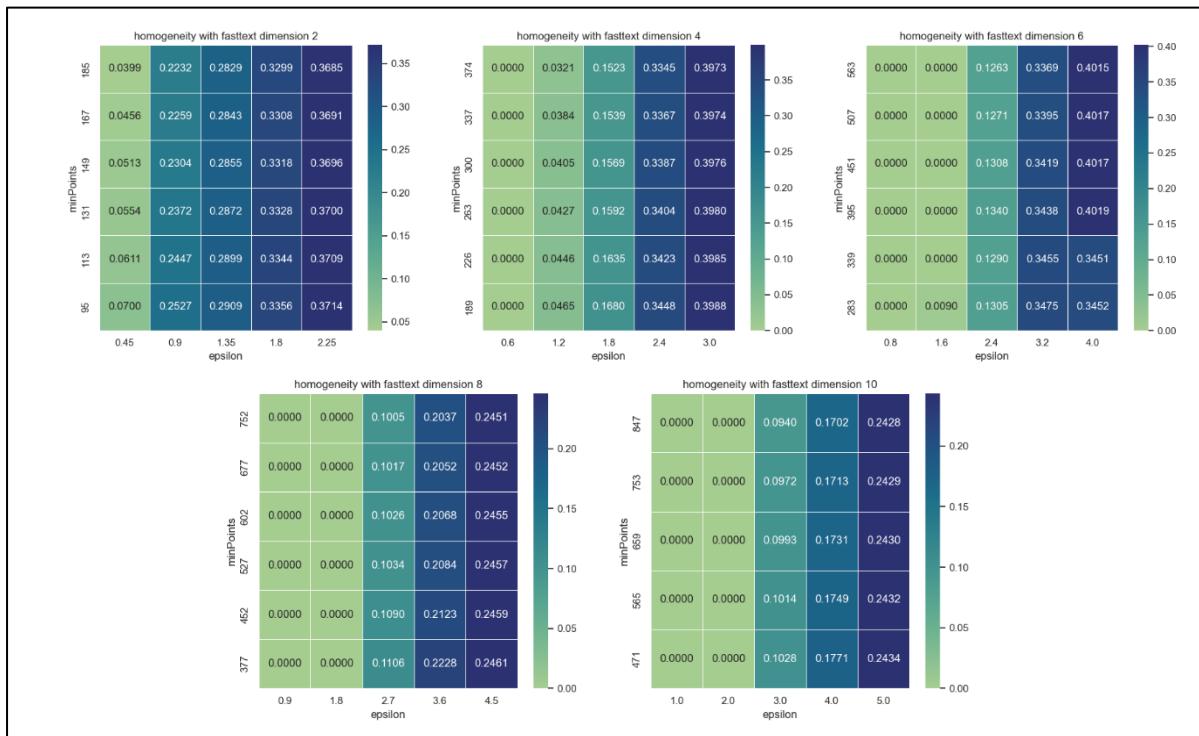


Figura 7: Risultati omogeneità primo esperimento con epochs=20

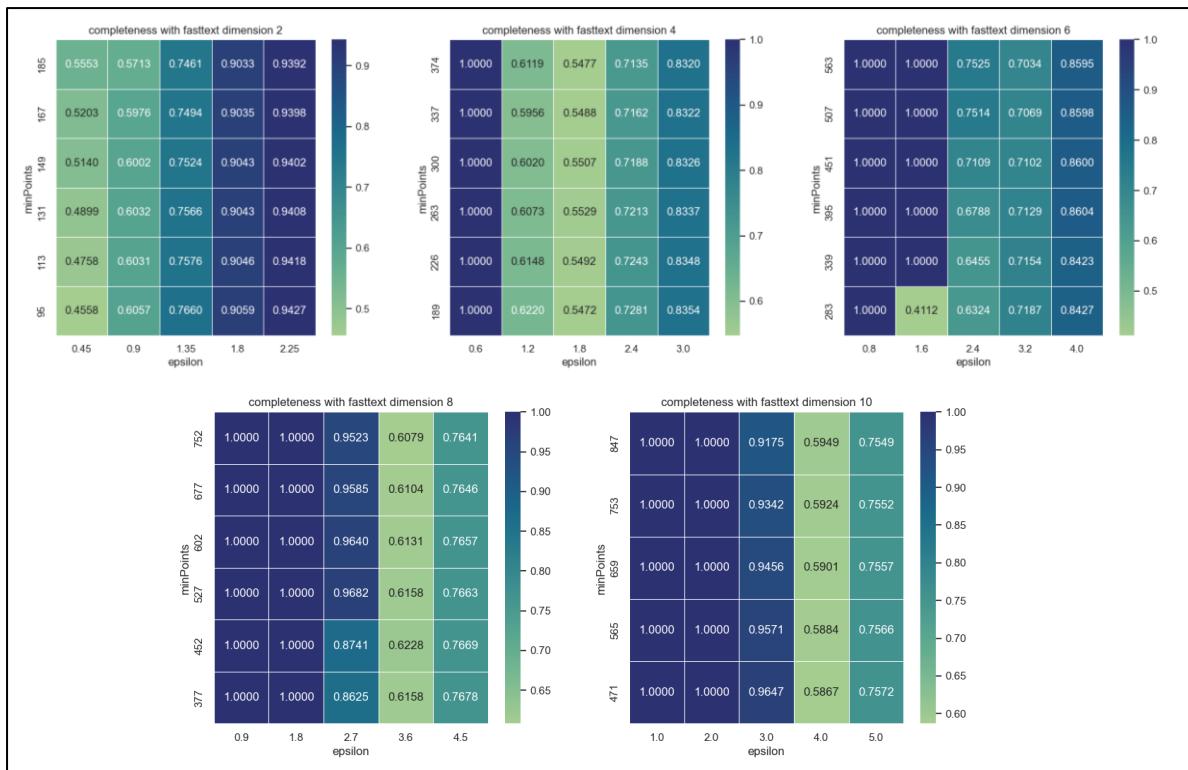


Figura 8: Risultati completezza primo esperimento con epochs=20

Osservando i valori di omogeneità e completezza ottenuti, si nota che il miglior trade-off tra le due metriche si ottiene in corrispondenza dei valori di *dim* che vanno da due a sei, mentre per i valori successivi si ottengono risultati globalmente peggiori.

Tenendo conto del fatto che la miglior segmentazione si ottiene per i valori di *dim*, *minPoints* ed *eps* definiti in precedenza, e che per gli stessi valori si ottiene un buon compromesso per le metriche supervised, si è deciso di condurre analisi più precise per questi valori dei parametri.

Analisi sulla miglior segmentazione

Individuati gli intervalli, o i valori dei parametri liberi, in cui si ottengono i migliori valori della silhouette media, si è deciso di condurre di nuovo gli esperimenti per quei valori, calcolando alcune metriche più approfondite. Tra queste si hanno la precision e il recall per ogni coppia cluster-classe, la matrice di confusione per il clustering e la silhouette di ogni campione. In particolare, le matrici di confusione si sono rivelate molto utili per individuare alcuni fenomeni rilevanti che si osservano negli esperimenti.

Considerando il primo esperimento, condotto con i valori dei parametri liberi $dim = 4$, $epochs = 20$, $eps = 3.0$, $minPoints = 189$, si è determinata la matrice di confusione della clusterizzazione visibile in Figura 9, in cui

sono rilevati sei cluster, oltre al *cluster_noise*. La prima cosa importante da sottolineare è che tutti gli elementi della classe “*bazarbackdoor*” sono isolati all'interno del *cluster_5*.

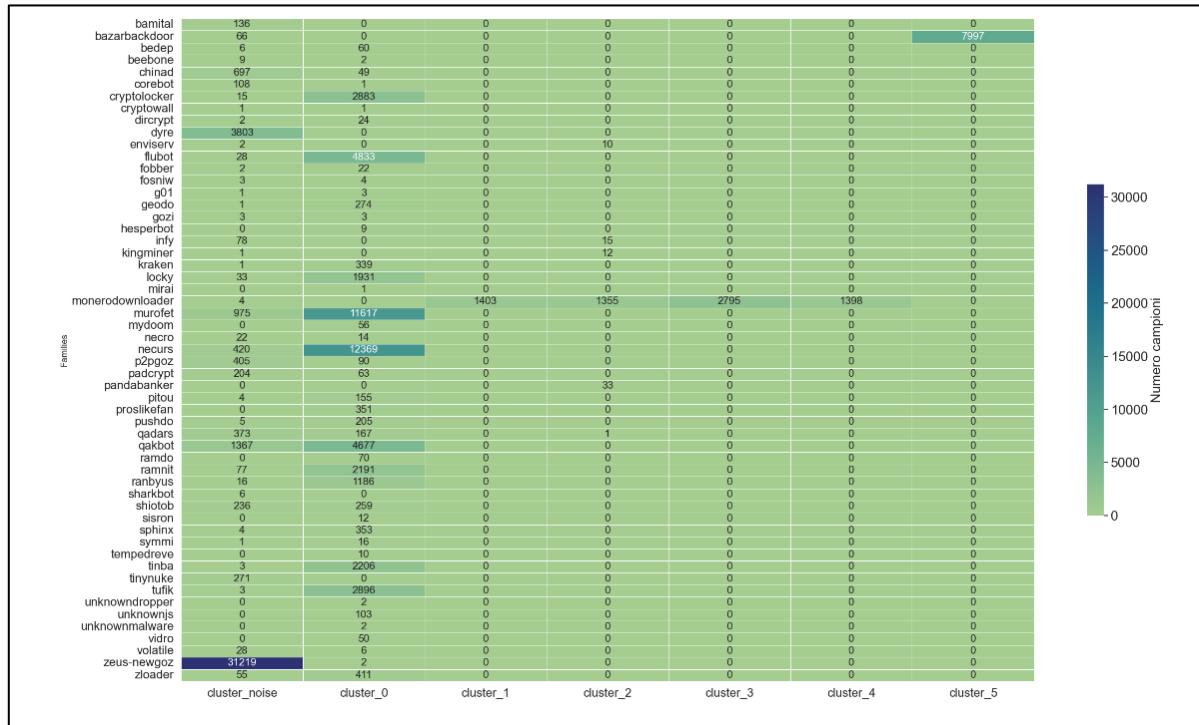


Figura 9: Matrice di confusione con dim=4 minPoints=189 eps=3.0

Questo significa che il *cluster_5* è rappresentativo della classe in oggetto, come testimoniato anche dagli alti valori di precision e recall rispetto alla famiglia di malware *bazarbackdoor*, visibili rispettivamente in [Figura 10](#), e in [Figura 11](#). Invece, considerando la classe *monerodownloader* si rileva che tutti gli elementi si distribuiscono equamente all'interno del *cluster_1*, del *cluster_2*, del *cluster_3* e del *cluster_4*. Questi cluster contengono per lo più solo elementi della famiglia di malware in esame, e per questo hanno altissimi valori della precision rispetto ad essa, configurandosi come cluster molto omogenei. D'altro canto, poiché i campioni si distribuiscono in più cluster, si ottiene che questi cluster non sono rappresentativi della classe *monerodownloader*, generando dei valori di recall non elevati. Per le altre famiglie si osserva che tutti gli elementi ad esse appartenenti, confluiscono in unico cluster, ovvero il *cluster_0*. Esso risulta essere molto completo rispetto alle singole classi, perché ne contiene la maggior parte dei campioni, ma poco omogeneo, perché non è rappresentativo di una singola famiglia, come osservabile in [Figura 10](#) e in [Figura 11](#). Eccezioni rispetto all'ultimo caso commentato sono le famiglie *dyre* e *zeus-newgoz*, i cui campioni sono prevalentemente inseriti

all'interno del *cluster_noise*, insieme a pochi campioni delle altre classi. Oltre a questi nel *cluster_noise* sono contenuti tutti gli elementi appartenenti alle famiglie con una cardinalità nel dataset inferiore al parametro *minPoints*.

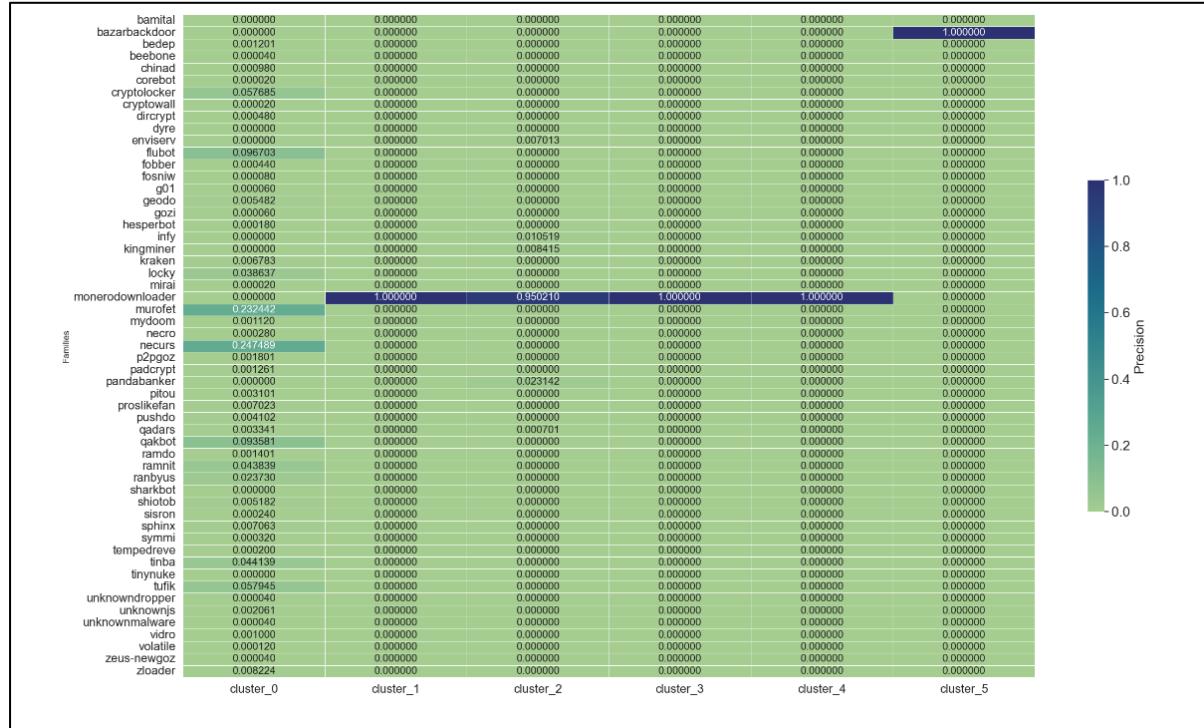


Figura 10: Precision con dim=4 minPoints=180 eps=3.0

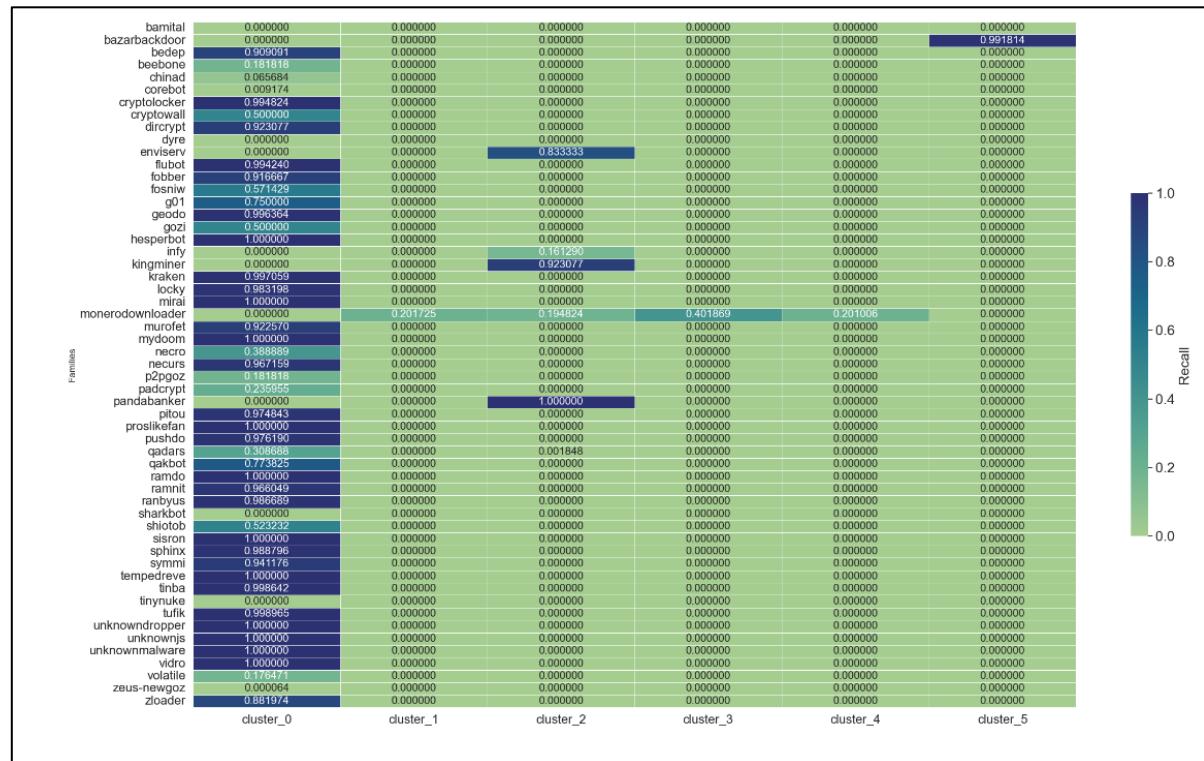


Figura 11: Recall con dim=4 minPoints=189 eps=3.0

Proseguendo, il secondo esperimento è stato condotto con i valori dei parametri liberi $dim = 3, epochs = 20, eps = 2.75, minPoints = 142$, calcolando sempre i valori di *precision* e *recall* per ogni coppia cluster-classe e la matrice di confusione della clusterizzazione visibile in [Figura 12](#). Tali risultati sono simili a quelli precedenti, in quanto si è ottenuto lo stesso numero di cluster, con qualche differenza rispetto alle famiglie segmentate.

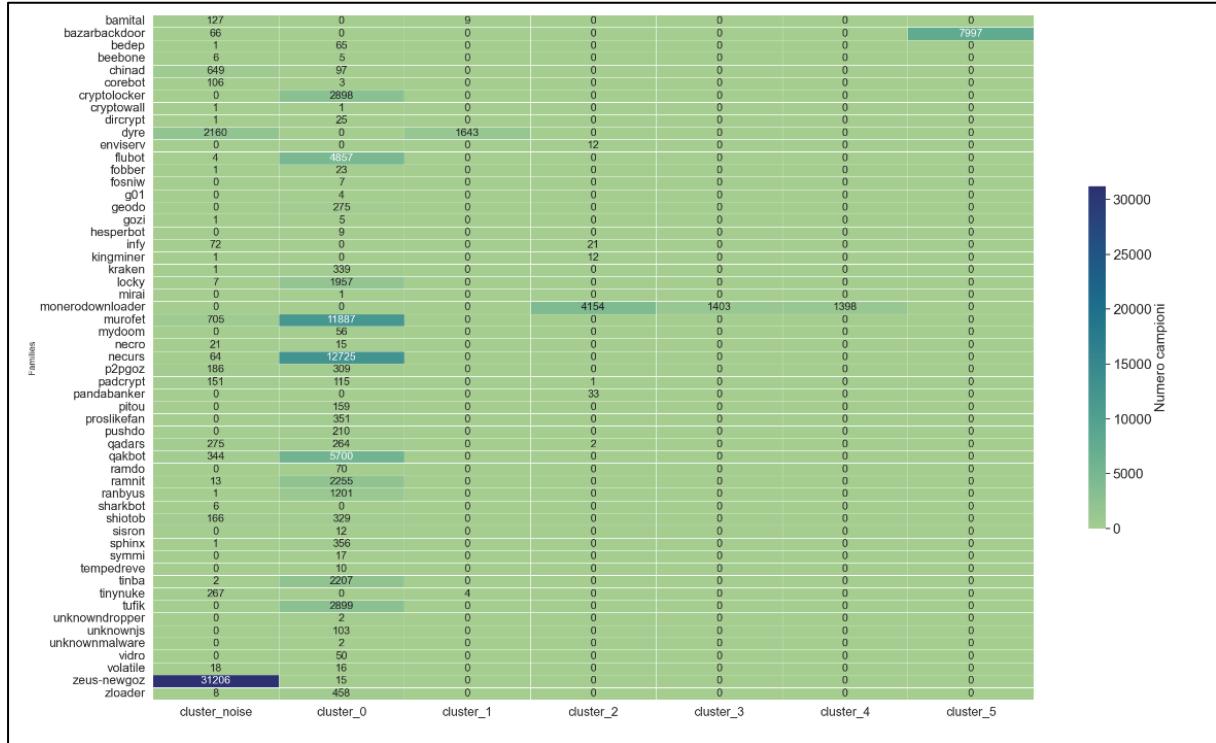


Figura 12: Matrice di confusione con $dim=3$ $minPoints=142$ $eps=2.75$

Come prima la famiglia *bazarbackdoor* viene isolata interamente all'interno di un unico cluster, mentre gli elementi della famiglia *monerodownloader* vengono divisi in più cluster. A differenza di prima i nomi di dominio *monerodownloader* sono divisi solo in tre cluster, ovvero *cluster_4*, *cluster_3* e *cluster_2*, di cui quest'ultimo ha una cardinalità maggiore rispetto agli altri due. Infatti, mentre i valori di precision sono alti per tutti e tre, come visibile in [Figura 13](#), sicuramente si può notare che il valore della *recall* rispetto a tale famiglia è molto più alto per il *cluster_2*, rispetto agli altri, come mostrato in [Figura 14](#). A differenza dell'esperimento precedente si nota che il *cluster_1* è molto omogeneo rispetto alla famiglia *dyre*, contenendo solo campioni della stessa. Allo stesso tempo però il cluster non è rappresentativo della famiglia, in quanto contiene solo la metà degli elementi, generando un valore di *recall* non elevato. Infine, come nel caso precedente si rileva che il resto delle

famiglie si colloca prevalentemente all'interno del *cluster_0*, fatta eccezione sempre per la famiglia *zeus-newgoz*, i cui elementi sono classificati come rumore, e per alcuni elementi della famiglia *dyre* che confluiscano nel *cluster_noise*.

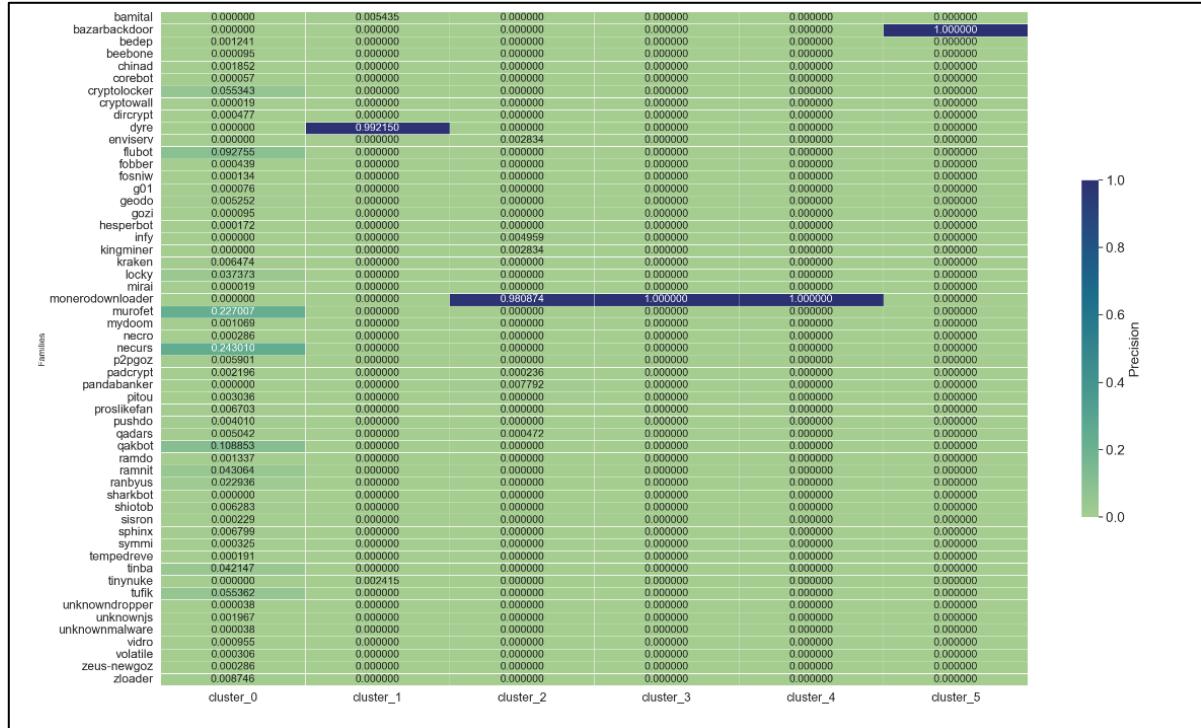


Figura 13: Precision con dim=3 minPoints=142 eps=2.75

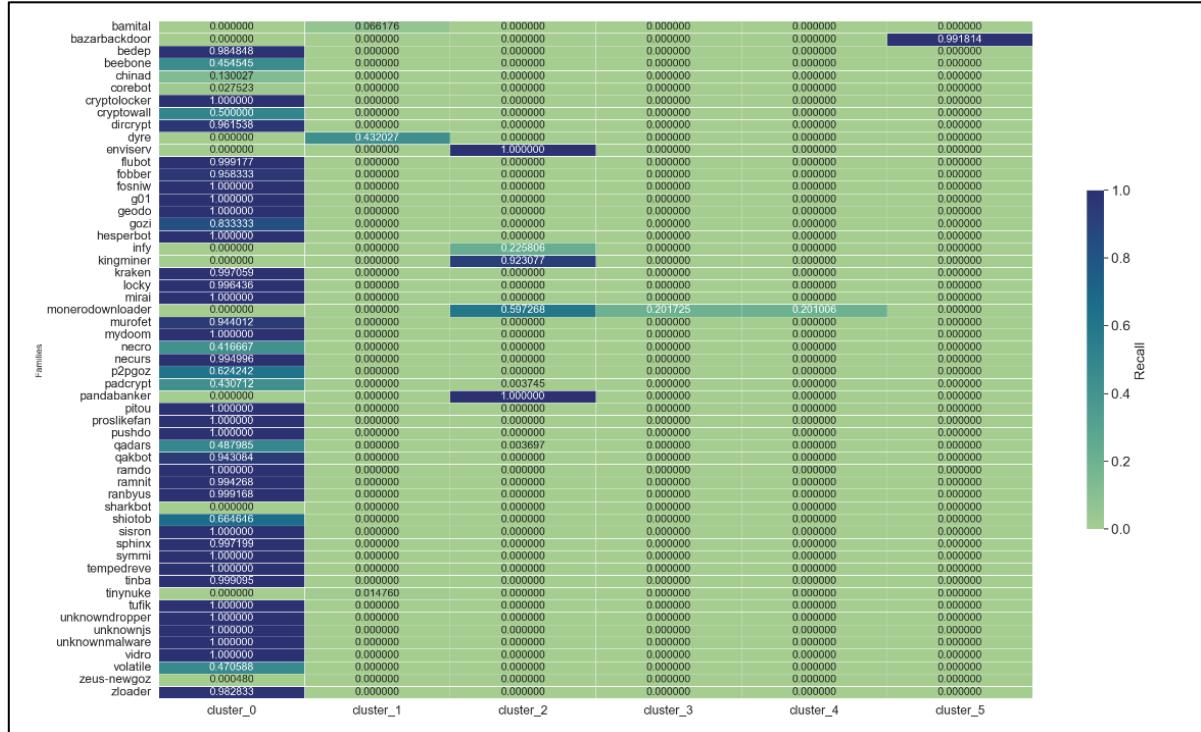


Figura 14: Recall con dim=3 minPoints=142 eps=2.75

Poi, per l'ultimo esperimento si sono impostati i valori dei parametri liberi $dim = 2$, $epochs = 20$, $eps = 2.25$, $minPoints = 95$, e si sono calcolate le stesse metriche dei precedenti esperimenti. Per questo esperimento si ottengono risultati leggermente differenti dagli altri due, come riportato anche nella matrice di confusione in [Figura 15](#). Innanzitutto, la prima cosa che si può notare, al livello macroscopico, è che il numero di cluster individuati passa da sei a tre.

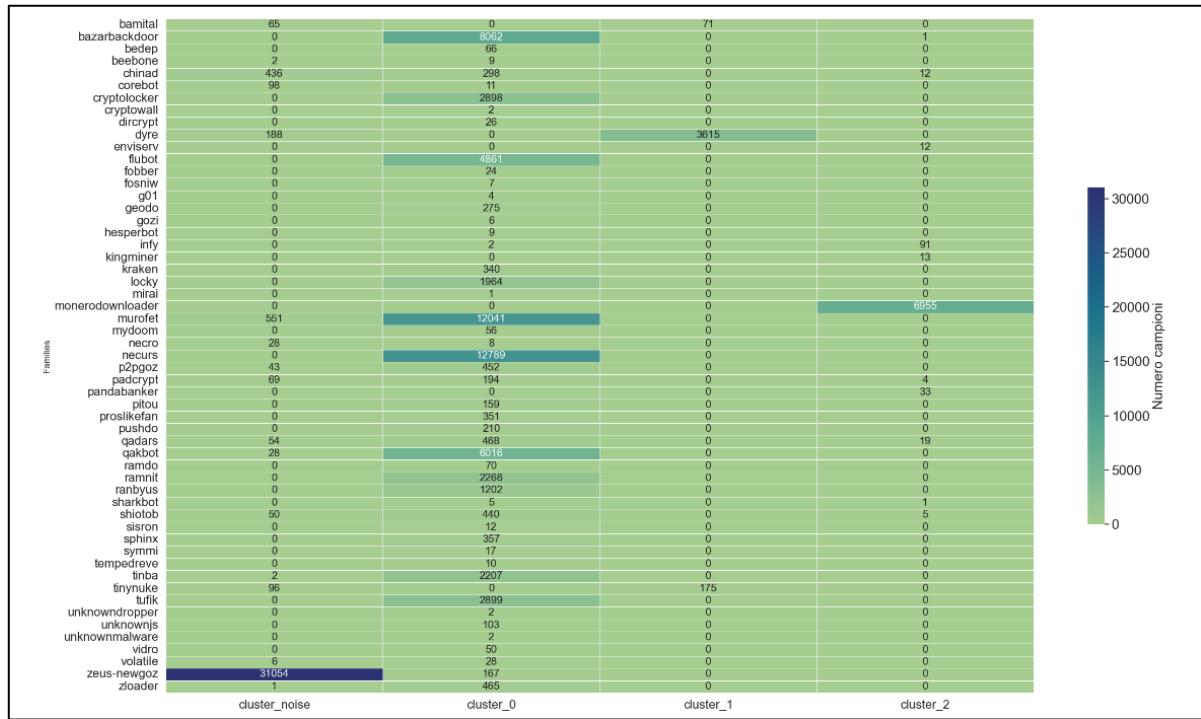


Figura 15: Matrice di confusione con dim=2 minPoints=95 eps=2.25

In dettaglio, il *cluster_1* contiene quasi tutti gli elementi della famiglia *dyre*, risultando molto omogeneo e completo rispetto alla famiglia, come dimostrato anche dagli alti valori di *precision* e *recall* visibili in [Figura 16](#) e in [Figura 17](#). Allo stesso modo il *cluster_2* è molto rappresentativo della famiglia *monerodownloader*, che invece precedentemente veniva scomposta in tre cluster diversi. Tale cluster, inoltre, assorbe anche tutti gli elementi di altre famiglie come *enviserv*, *infy*, *kingminer* e *pandabanker*. Invece, in questo caso, i campioni della famiglia *bazarbackdoor* rientrano nel *cluster_0*, al cui interno confluiscono quasi tutte le famiglie di malware nel dataset, anche alcune molto numerose come *flubot*, *murofet*, *necurs* e *qakbot*. Infine, come in precedenza, anche nell'esperimento corrente spicca il caso della famiglia *zeus-newgoz*, che viene inserita completamente nel *cluster_noise*. Anche se questo potrebbe sembrare un fattore negativo a prima vista, un'altra chiave di lettura potrebbe essere quella di considerare il *cluster_noise* come un

cluster alternativo, per cui tale famiglia verrebbe segmentata in maniera discreta.

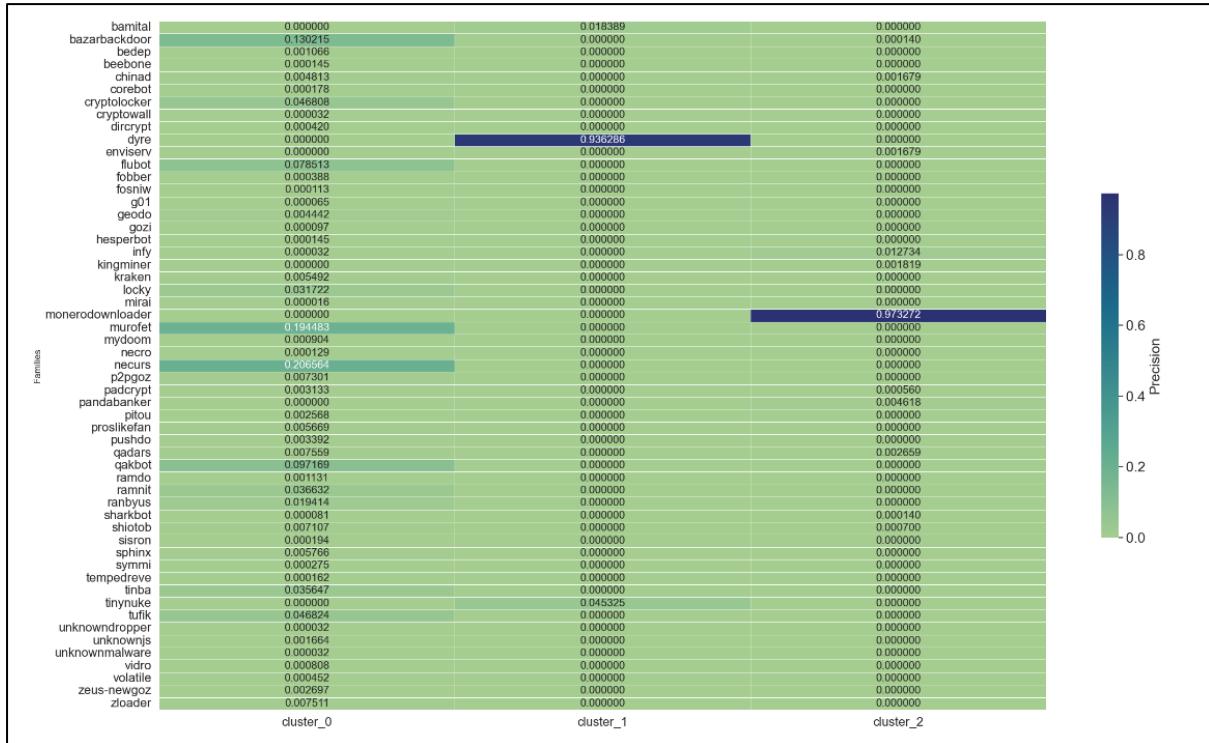


Figura 16: Precision con dim=2 minPoints=95 eps=2.25

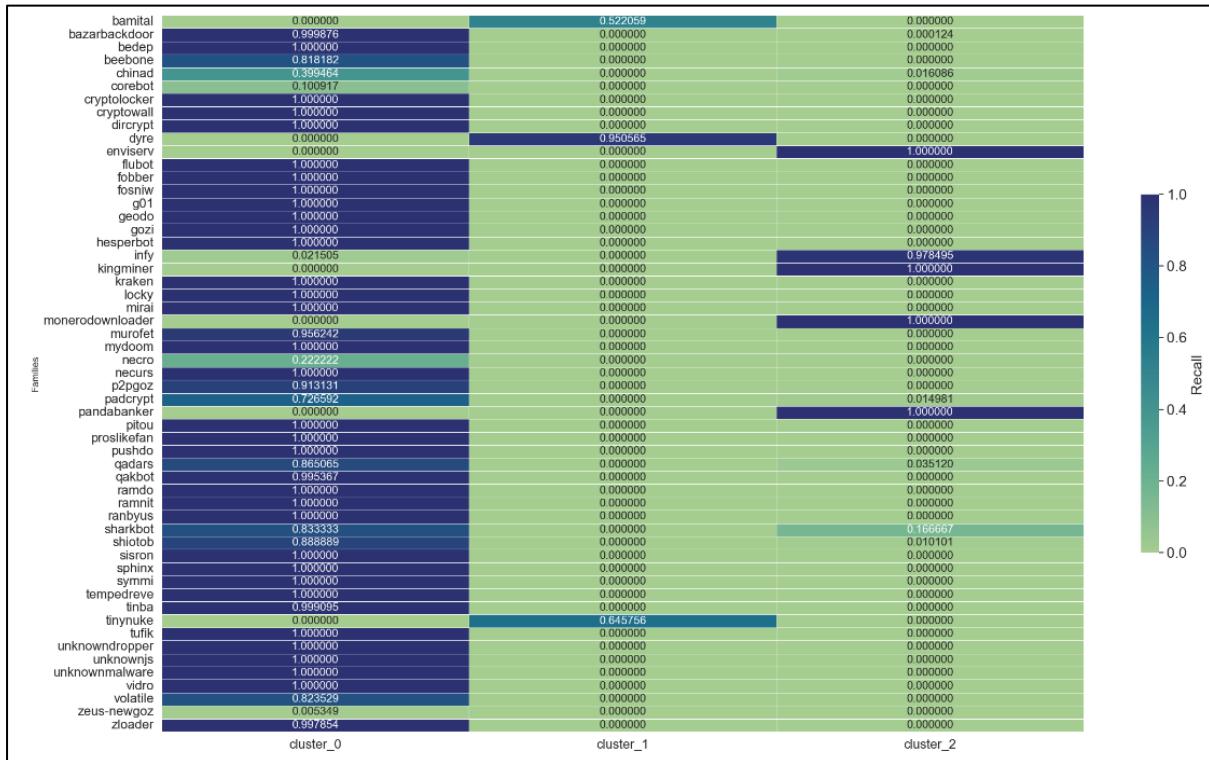


Figura 17: Recall con dim=2 minPoints=95 eps=2.25

Infine, per tutti gli esperimenti si sono calcolate le silhouette di ogni campione presente nel dataset, per stabilire più precisamente la qualità della segmentazione, sia considerando elementi di una singola famiglia, sia considerando interi cluster. Infatti, indagando più a fondo le silhouette di ogni campione, si può vedere in maniera chiara l'effetto che hanno le silhouette degli elementi nel dataset che confluiscano nel *cluster_noise*. Come riportato nella [Figura 18](#) si vede che per tutti gli esperimenti la maggior parte degli elementi appartenenti al *cluster_noise*, possiede una silhouette tra -0.2 e 0.1, con un numero ridotto di elementi che tende ad avere una silhouette prossima a -0.5.

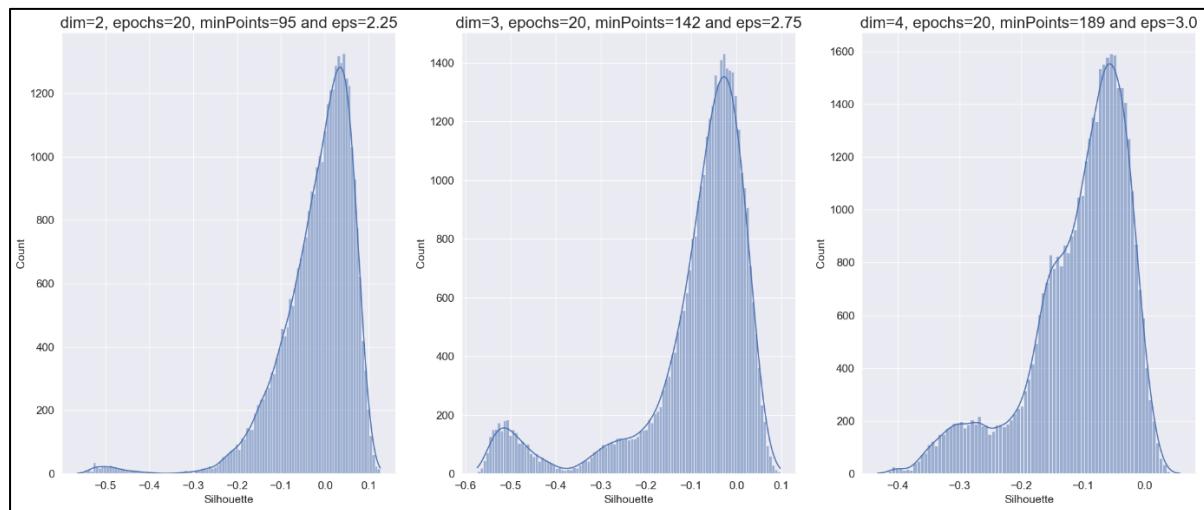
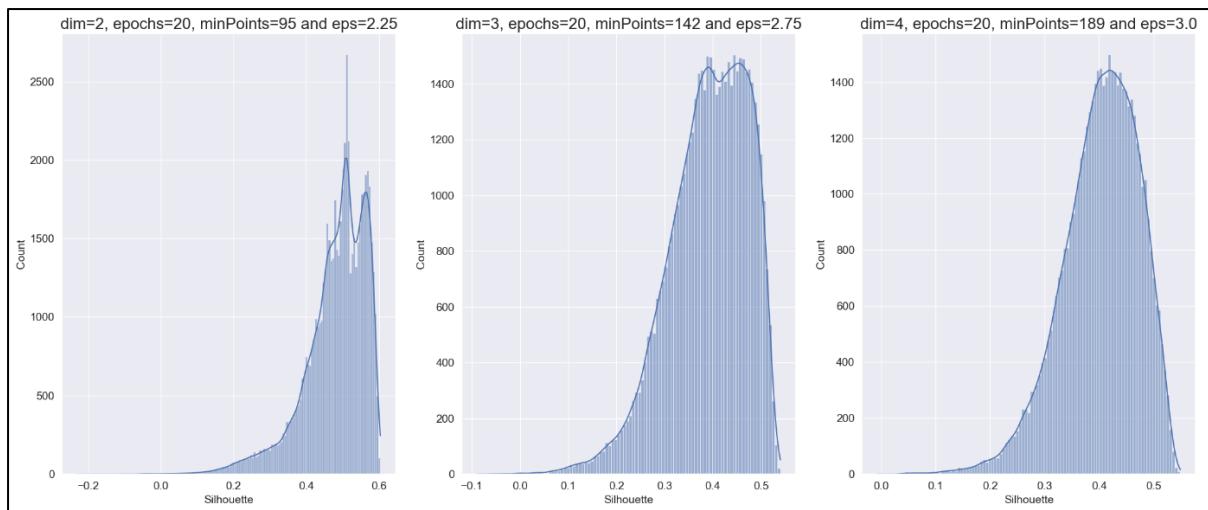


Figura 18: Distribuzioni dei valori della silhouette per il *cluster_noise*

Quindi, la maggior parte degli elementi appartenenti al *cluster_noise* possiede un valore della silhouette o molto basso, oppure tale per cui non si può indicare se questi siano stati inseriti nel cluster corretto o meno. Inoltre, poiché all'interno del *cluster_noise* finisce circa un terzo degli elementi nel dataset, questi valori delle silhouette tendono ad avere una forte influenza sul valore medio che si riduce in maniera sensibile. Al contrario considerando il *cluster_0*, che in tutti gli esperimenti si è dimostrato quello che assorbe la maggior parte di elementi all'interno del dataset, si verifica che i valori delle silhouette associati a ogni elemento sono soddisfacenti. Infatti, come mostrato in [Figura 19](#) si può vedere che la maggior parte degli elementi all'interno del *cluster_0* è associato a un valore di silhouette pari in media a 0.5, con pochissimi outlier che hanno un valore negativo o nullo. Quindi, nonostante in esso confluiscano tantissimi elementi di famiglie diverse, la qualità del cluster in esame è discreta per i valori dei parametri considerati.



27

Figura 19: Distribuzioni dei valori della silhouette per il cluster_0

Oltre ai singoli cluster, si sono analizzate anche le silhouette degli elementi appartenenti a delle singole famiglie di malware, in particolare *bazarbackdoor* e *monerodownloader*. Cominciando dalla prima, come visibile a sinistra in [Figura 20](#), si nota che la maggior parte degli elementi ha associati dei valori di silhouette più che soddisfacenti, a testimoniare che la qualità del *cluster_5*, in cui questi elementi sono raccolti, è molto alta. Non tutti i campioni *bazarbackdoor* presenti nel dataset confluiscono nel *cluster_5*, in quanto 66 elementi sono inseriti nel *cluster_noise*, come si vede in [Figura 20](#) a destra, ai quali sono associati valori della silhouette negativi che testimoniano una segmentazione errata.

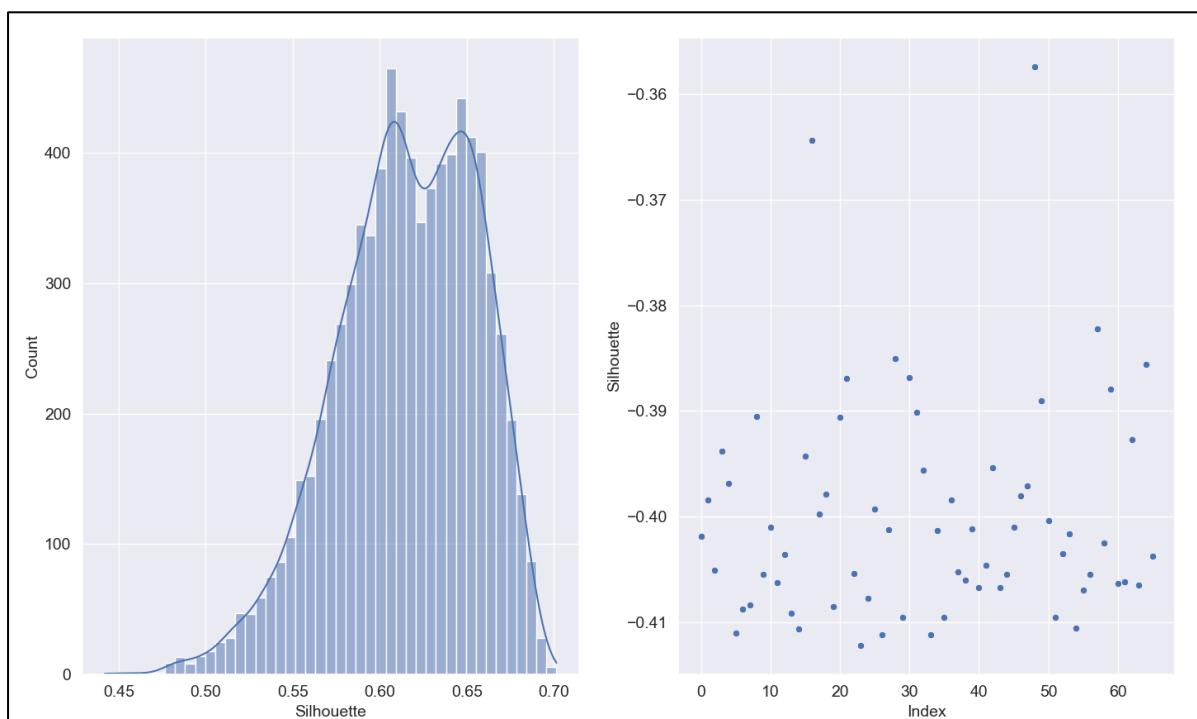


Figura 20: Distribuzione dei valori della silhouette per la famiglia bazarbackdoor

Nel caso della famiglia *monerodownloader*, nell'esperimento eseguito con $dim = 4, epochs = 20, eps = 3.0, minPoints = 189$, gli elementi del dataset si distribuiscono in quattro cluster, per cui la distribuzione dei valori della silhouette è riportata in Figura 21. Nonostante per tutti i cluster la maggior parte dei campioni in essa contenuta sia associata a valori positivi della silhouette, mediamente la silhouette non è molto alta. La maggior parte degli elementi contenuti in questi cluster possiede un valore della silhouette intorno allo 0.4, con alcuni che si assestano anche attorno allo 0.3. Questo testimonia che probabilmente, anche se gli elementi all'interno dei cluster sono molto vicini, comunque i quattro cluster individuati non sono molto distanti tra loro nello spazio, facendo riferimento ad elementi della stessa famiglia.

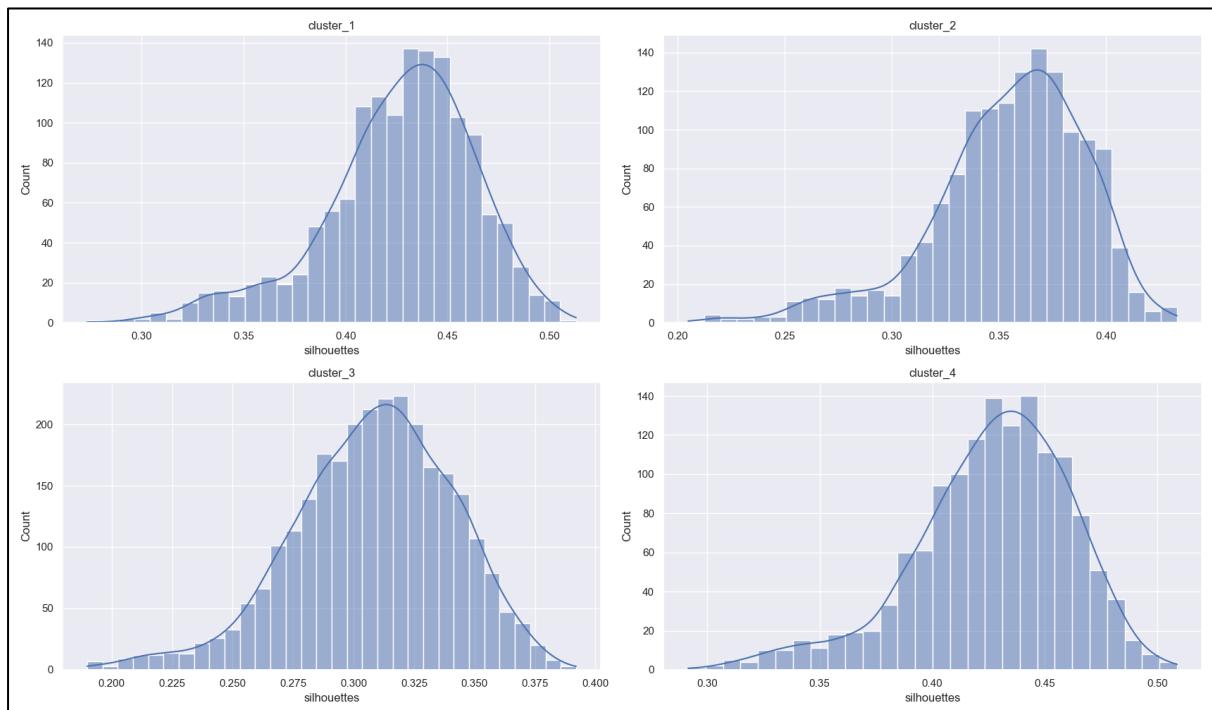


Figura 21: Distribuzioni dei valori della silhouette per i cluster contenenti elementi *monerodownloader*

Analisi di medie e varianze delle feature

Avendo individuato le famiglie ben segmentate per ogni dimensione, si può procedere con il loro studio al fine di rilevare eventuali pattern. Nello specifico, si possono analizzare le medie e varianze delle famiglie prese in considerazione, al variare della dimensione. Per poter visualizzare efficacemente media e varianza verranno considerate separatamente le feature dei vettori media e varianza. In base anche alla dimensione considerata, ogni feature relativa ad un bigramma avrà il suo colore nei

grafici presentati successivamente. La prima feature verrà rappresentata di colore rosso, la seconda di verde, la terza di blue e la quarta di ciano.

Per primo, è necessario sottolineare che, applicando alle parole un padding di zeri per egualare la lunghezza dei vettori indipendentemente dalla lunghezza delle parole di partenza, i valori delle feature sono sempre, da un certo punto in poi, pari a 0. In aggiunta, un fattore rilevante per l'andamento di media e varianza è la lunghezza dei nomi di dominio. Se per una famiglia questa è molto variabile, infatti, si avranno dei nomi con più feature non nulle rispetto ad altri, e ciò influisce inevitabilmente sul calcolo di media e varianza per quelle specifiche feature. Questi effetti sono visibili negli andamenti statistici di varie famiglie esaminate in seguito.

Considerando la famiglia *dyre*, si può notare un andamento regolare di media e varianza, eccetto per i primi e gli ultimi valori. In dettaglio, per la dimensione 2 ([Figura 22](#)), la varianza delle prime feature (circa 0.10) è più alta di quella delle seconde (circa 0.02), ed entrambe rimangono stabili. Nel caso della dimensione 3 ([Figura 23](#)), si notano alcuni cambiamenti; in particolare, le varianze delle prime e delle seconde feature si assestano su valori molto simili (attorno a 0.05), mentre le terze hanno valori più alti (sopra 0.10), mantenendo tutte comunque una regolarità.

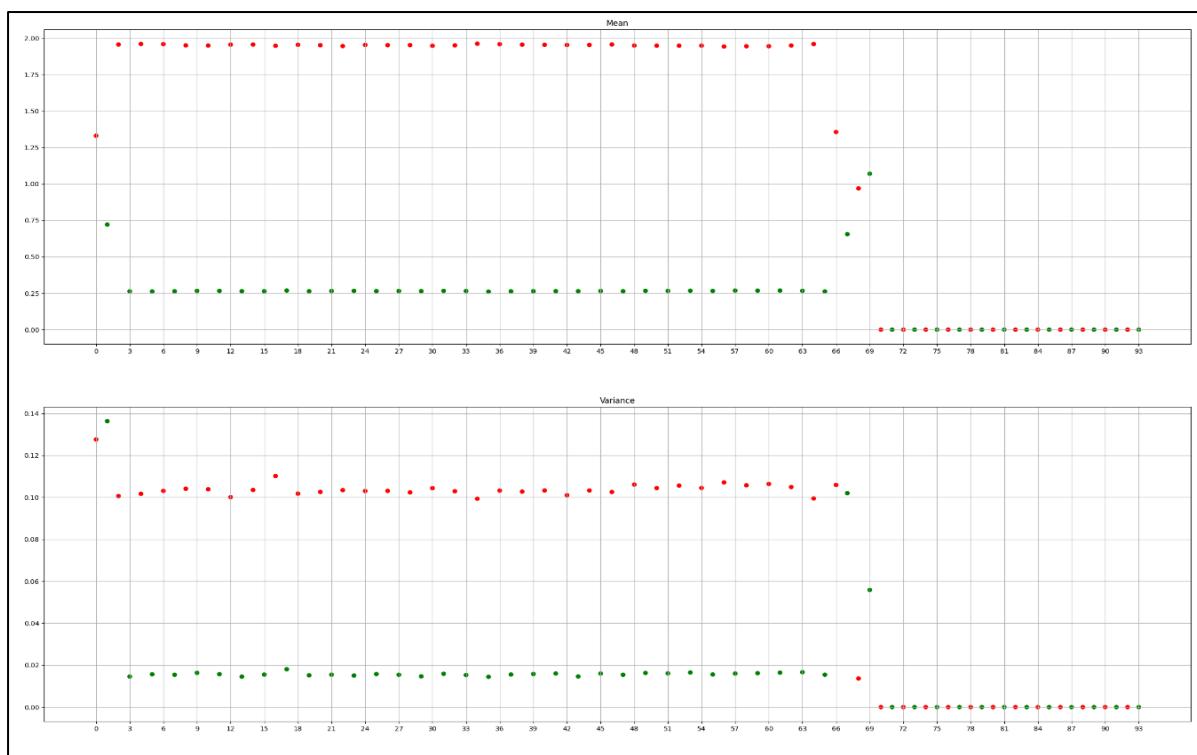


Figura 22: Media e Varianza di "dyre" con dim=2

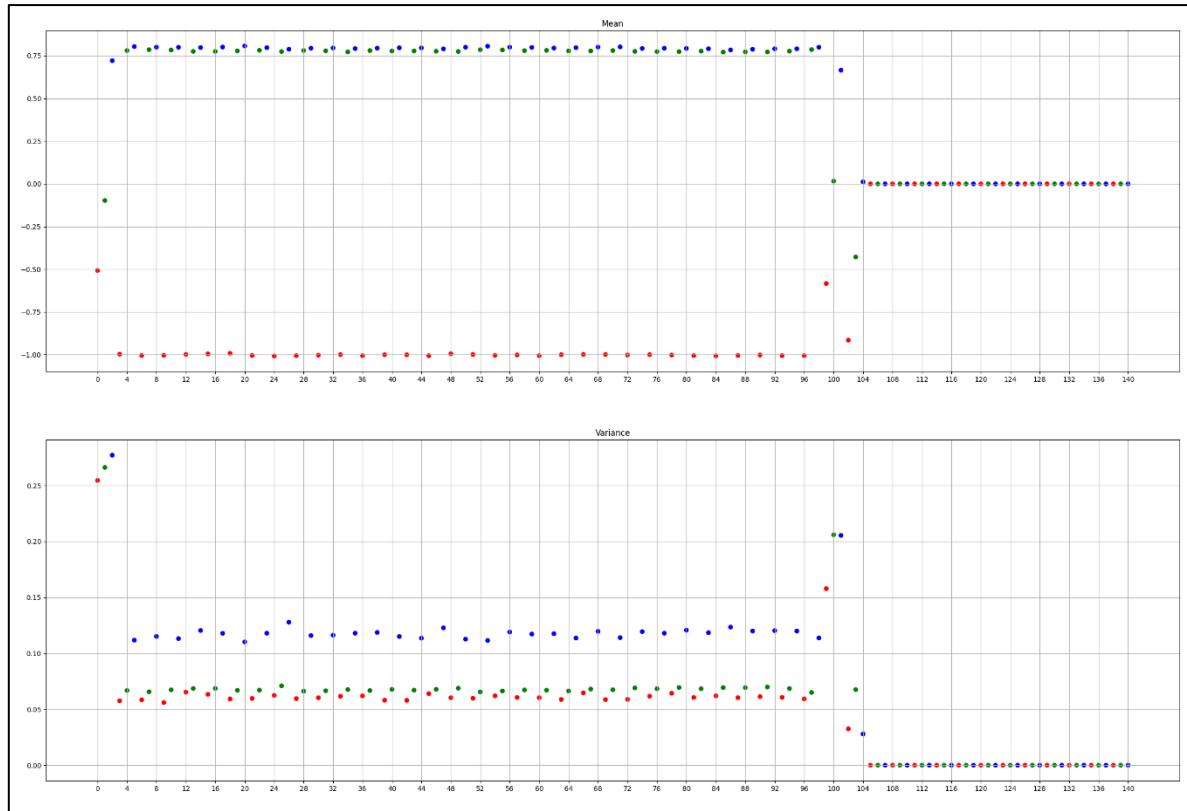


Figura 23: Media e Varianza di "dyre" con dim=3

È interessante notare anche che le medie delle seconde e delle terze feature hanno valori molto simili fra loro. Per quanto riguarda la lunghezza delle parole, nella maggior parte dei casi queste contengono 35 bigrammi. Quindi, questa famiglia di DGA, costruisce nomi di dominio di lunghezza poco variabile. Con $dim = 2$, gli elementi rientrano interamente in un unico cluster, in quanto questi si trovano probabilmente vicini fra di loro. Tuttavia, aumentando la dimensione, aumenta la sparsità dei punti e, di conseguenza, la segmentazione peggiora e non si riesce ad individuare un cluster rappresentativo.

La famiglia *monerodownloader* ha un comportamento differente, in quanto si ottiene un cluster rappresentativo in corrispondenza della dimensione 2 ([Figura 24](#)), mentre negli altri casi i campioni si distribuiscono su più cluster. Considerando la segmentazione migliore, i valori di media e varianza delle prime feature sono nettamente superiori alle seconde, ma dalla seconda metà (dopo la 23° feature) i valori perdono di regolarità. Tale fenomeno si verifica anche con la dimensione 3 ([Figura 25](#)) e con la dimensione 4 ([Figura 26](#)), probabilmente a causa della grande variabilità di lunghezza delle parole. Infatti, si è visto che i nomi di dominio di questa famiglia hanno quattro possibili lunghezze, motivo per cui i punti relativi si distribuiscono in quattro cluster differenti.

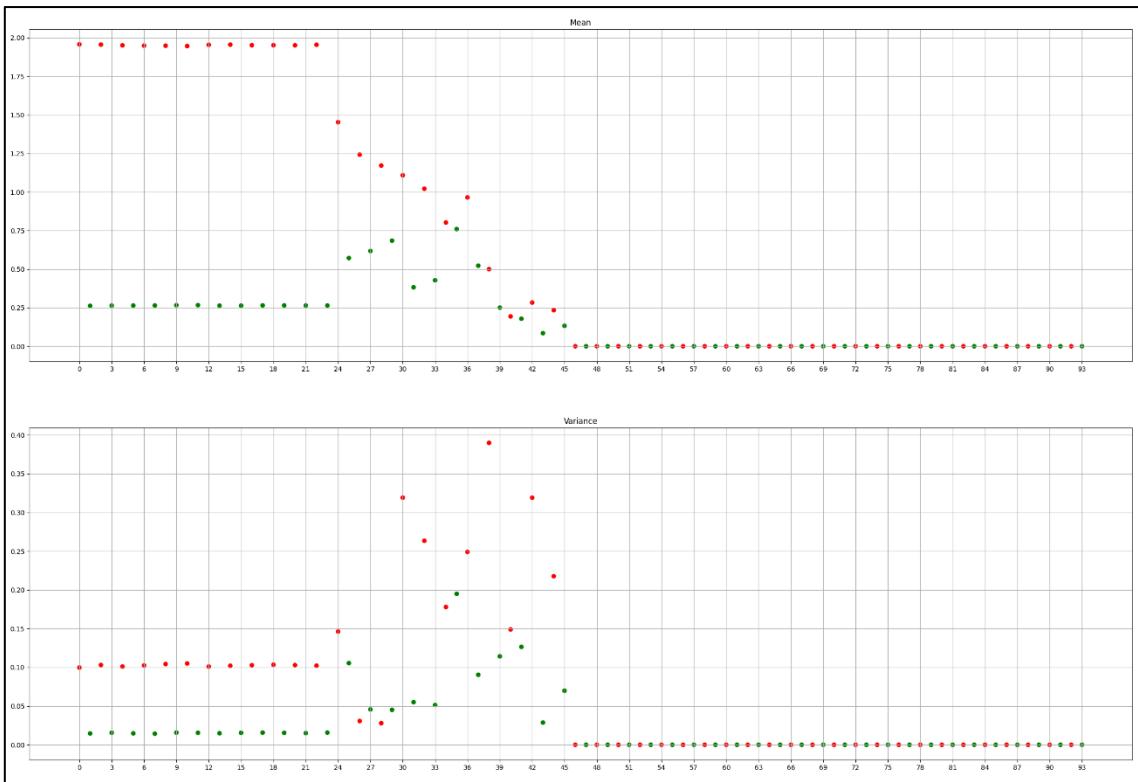


Figura 24: Media e Varianza di "monerodownloader" con $\text{dim}=2$

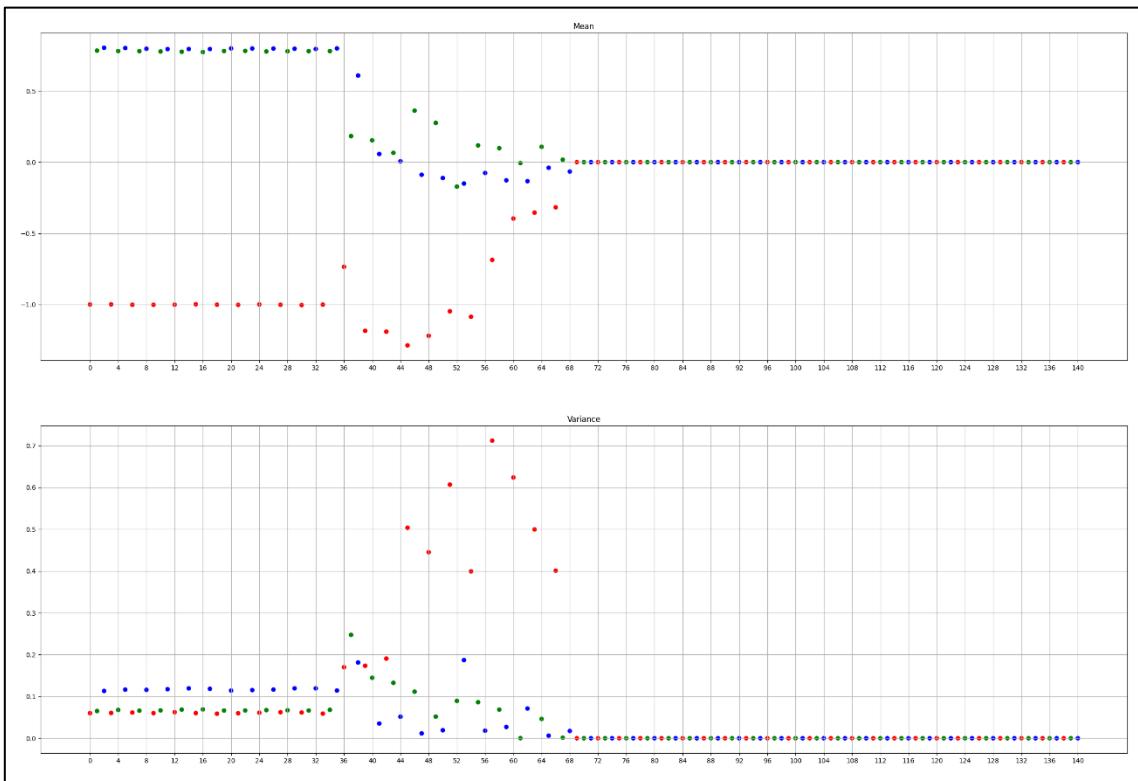


Figura 25: Media e Varianza di "monerodownloader" con $\text{dim}=3$

All'aumentare delle dimensioni cresce il numero di feature per ogni nome di dominio, e quindi il maggior contenuto informativo e la diversa lunghezza dei nomi fanno sì che questi si distribuiscano in più cluster.

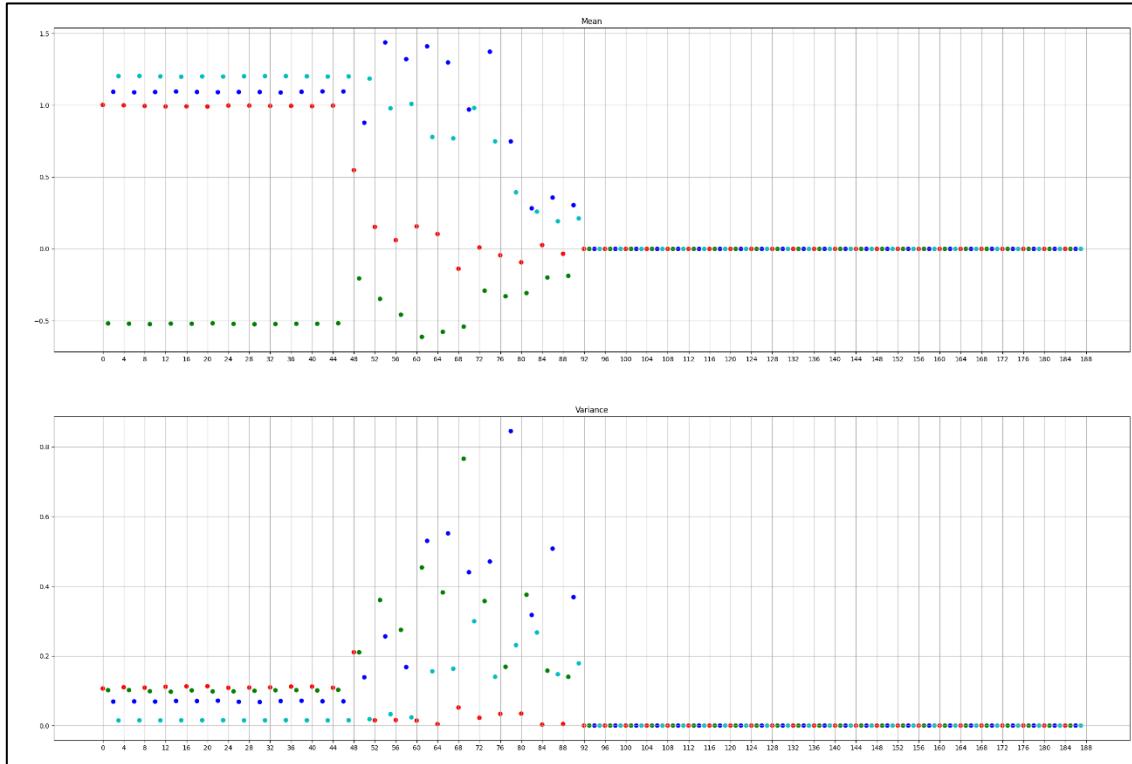


Figura 26: Media e Varianza di "monerodownloader" con dim=4

Passando alla famiglia *bazarbackdoor*, che viene correttamente segmentata sia con dimensione 3 sia con dimensione 4, si presenta un andamento totalmente differente dai casi precedenti. La media e la varianza non si stabilizzano su un valore, ma cambiano in maniera importante il loro valore al variare delle feature. Focalizzando l'attenzione sulla dimensione 3 ([Figura 27](#)), le medie delle prime e delle terze feature diminuiscono con la stessa tendenza, mentre la media delle seconde inizialmente è stabile, successivamente ha un picco verso lo 0, poi decresce verso i valori delle terze (-1.0). Invece, tutte le varianze tendono a 0, anche se sono presenti degli outlier. Per quanto riguarda la dimensione 4 ([Figura 28](#)), le medie delle prime, terze e quarte feature sono regolari, anche se gli ultimi valori variano maggiormente. Invece, la media delle seconde, pur seguendo inizialmente le prime, non segue un andamento regolare, in quanto ci sono sempre oscillazioni evidenti. Tuttavia, i valori delle varianze tendono a diminuire, come in precedenza, e allo stesso tempo sono di un ordine di grandezza più piccole rispetto alle medie.

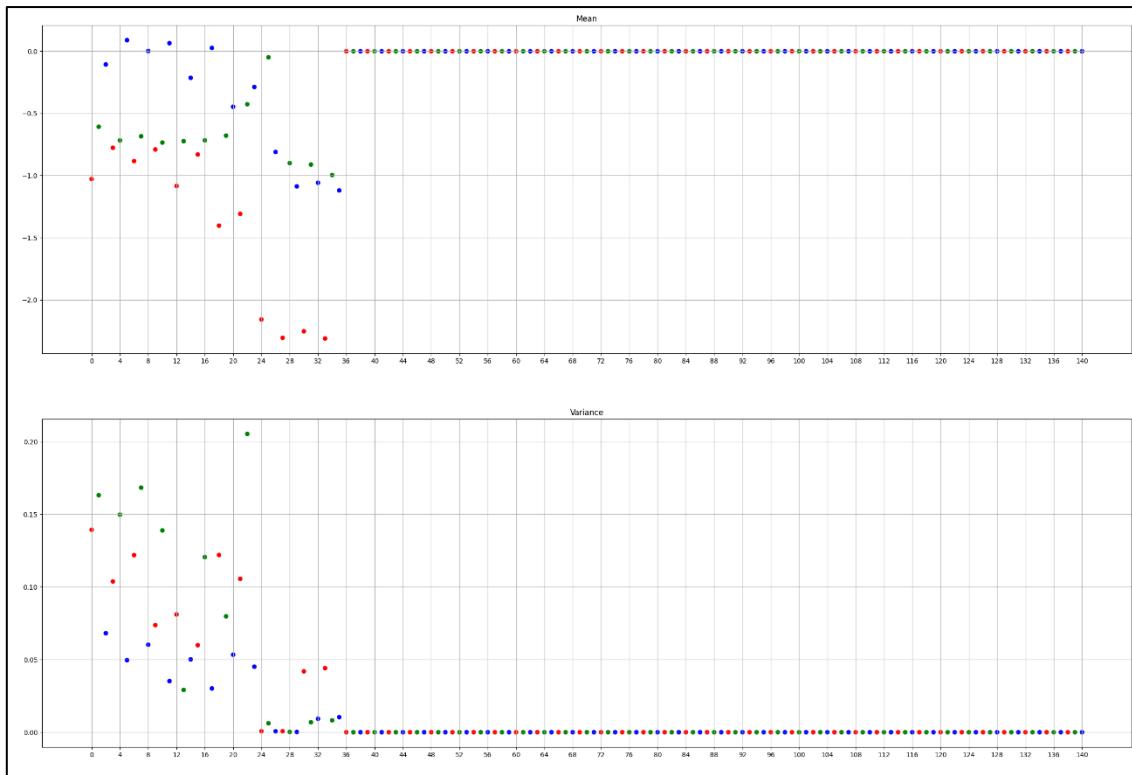


Figura 27: Media e Varianza di "bazarbackdoor" con dim=3

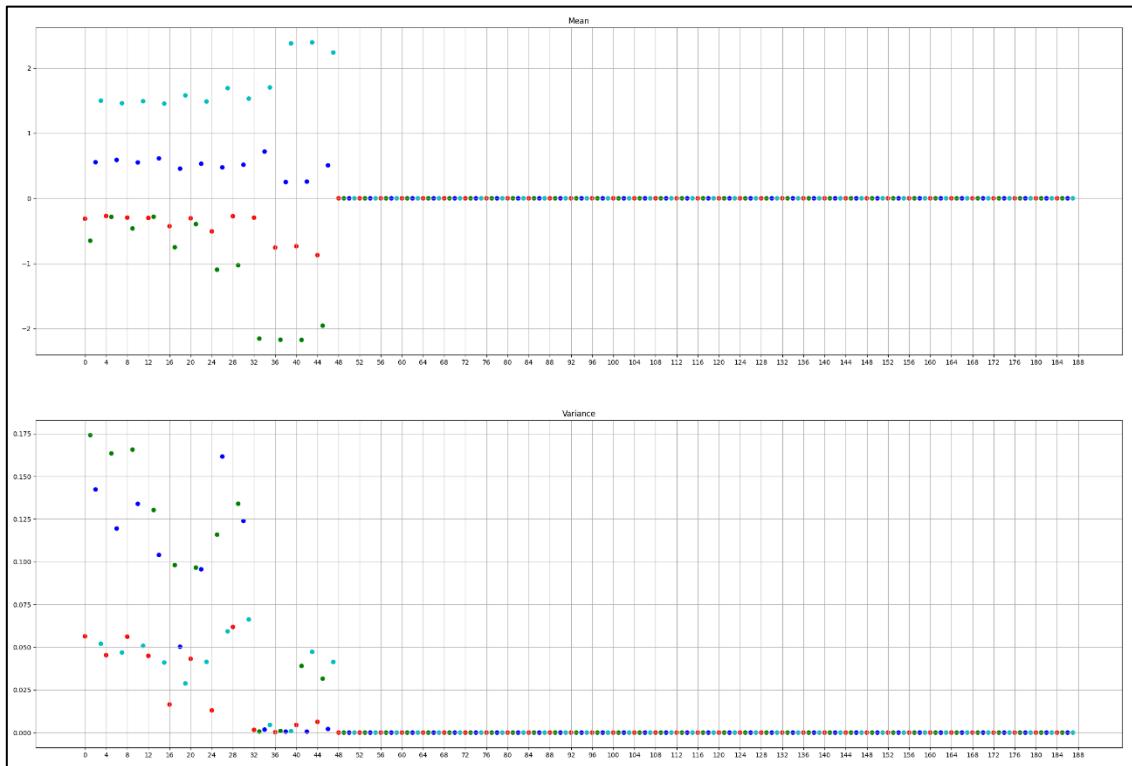


Figura 28: Media e Varianza di "bazarbackdoor" con dim=4

In entrambe le dimensioni si osserva una grande variabilità della media delle feature dei nomi di dominio *bazarbackdoor*, però allo stesso tempo i valori delle varianze sono piccoli rispetto ai valori medi. Questo,

combinato al fatto che la maggior parte dei nomi di dominio della famiglia ha lunghezza pari a 12 bigrammi, permette di segmentare gli elementi in un unico cluster. Solo pochi elementi non rientrano all'interno di quell'insieme, avendo una lunghezza inferiore pari a 10 bigrammi e a causa della grande variabilità della media.

Un'altra famiglia che viene analizzata in dettaglio è **zeus-newgoz**, che rappresenta circa un terzo del dataset. Anche se i campioni di questa famiglia non ricadono propriamente in un cluster, si può notare che essi sono catalogati quasi sempre come rumore. Quindi, questa analisi vuole verificare la presenza di un andamento differente da quelli osservati che giustifichi tale fenomeno. Effettivamente, le medie e le varianze non si assestano su valori precisi, ma hanno una certa periodicità. Nel caso della dimensione 2 ([Figura 29](#)) e della dimensione 3 ([Figura 30](#)), media e varianza delle feature hanno un andamento ondulatorio fino a quando iniziano a tendere verso lo 0. Per quanto riguarda la dimensione 4 ([Figura 31](#)), si nota che le quarte feature (non presente nei casi precedenti) ha un andamento più regolare: i valori della media e della varianza sono stabili (rispettivamente 1.4 e 0.04 circa), anche se gli ultimi valori differiscono. Dato che le lunghezze dei nomi di dominio appartenenti a questa famiglia sono fortemente variabili, i campioni sono sparsi e, di conseguenza, un algoritmo *density-based* fa fatica a isolargli correttamente.

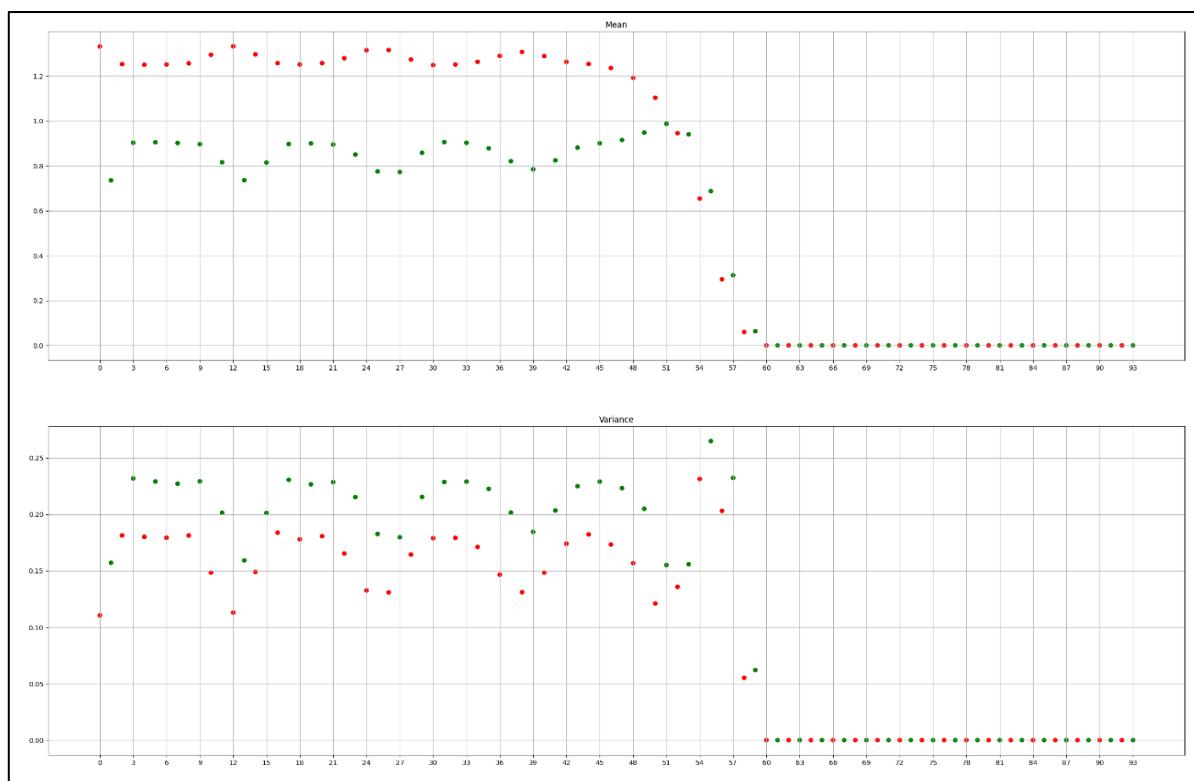


Figura 29: Media e Varianza di "zeus-newgoz" con dim=2

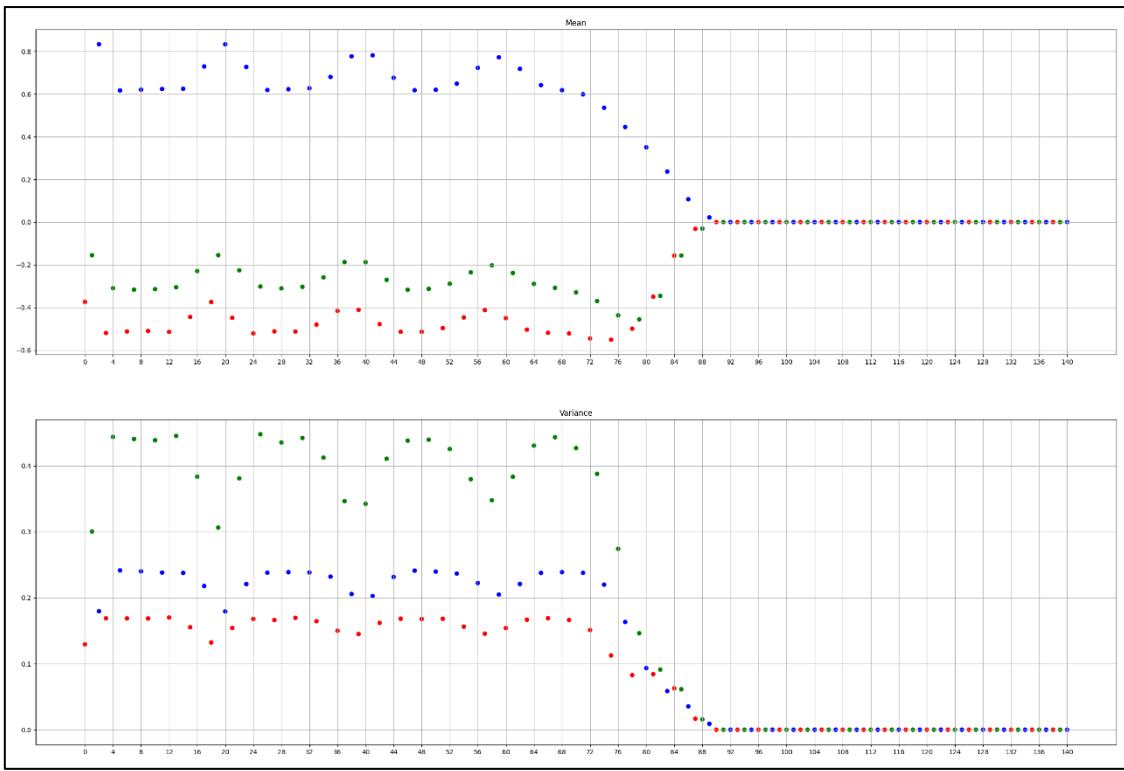


Figura 30: Media e Varianza di "zeus-newgoz" con $\text{dim}=3$

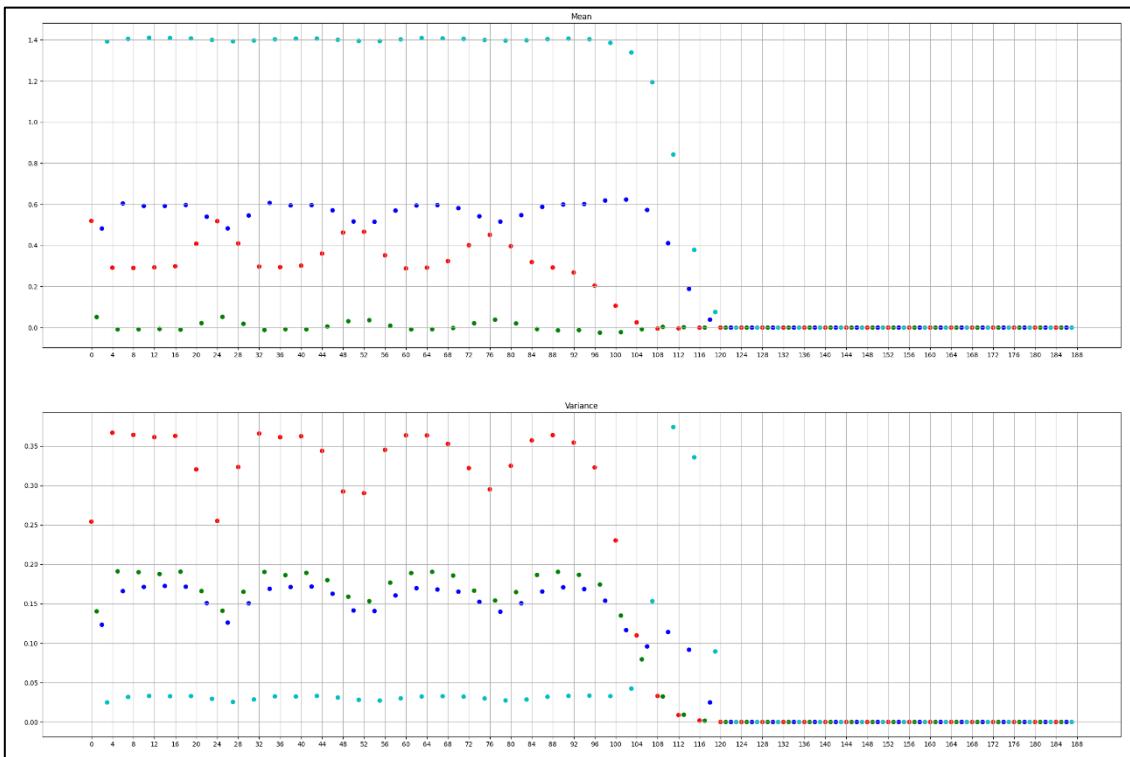


Figura 31: Media e Varianza di "zeus-newgoz" con $\text{dim}=4$

Analisi senza famiglie correttamente segmentate

Avendo individuato alcuni cluster con valori alti di *precision* e *recall*, si può considerare la parte di dataset che non viene segmentata in maniera corretta. Dunque, si possono escludere i campioni delle famiglie precedentemente individuate e si può eseguire il medesimo studio sul dataset rimanente. Infatti, potrebbe accadere che l'addestramento di FastText sia influenzato fortemente da quei campioni, per cui il processo di costruzione dei cluster venga sbilanciato verso poche famiglie.

Dopo aver rimosso *bazarbackdoor*, *dyre* e *monerodownloader* per le dimensioni in cui esse venivano segmentate correttamente, si eseguono nuovamente l'addestramento di FastText e l'algoritmo di clustering. Nel caso in cui $dim = 2$, $epochs = 20$, $minPoints = 95$, $eps = 2.25$, viene individuato un solo cluster ([Figura 32](#)). Nel *cluster_0* rientrano campioni di diverse famiglie, alcune delle quali piuttosto numerose, come *murofet* o *necurs*, e altre di dimensioni esigue, come *vidro* o *mirai*. All'interno del *cluster_noise* sono presenti elementi di diverse famiglie, compresa *zeus-newgoz*. Si può notare che *bazarbackdoor* viene inclusa all'interno del cluster individuato. In generale, i valori di *precision* sono bassi, mentre quelli di *recall* variano in base alle diverse famiglie.

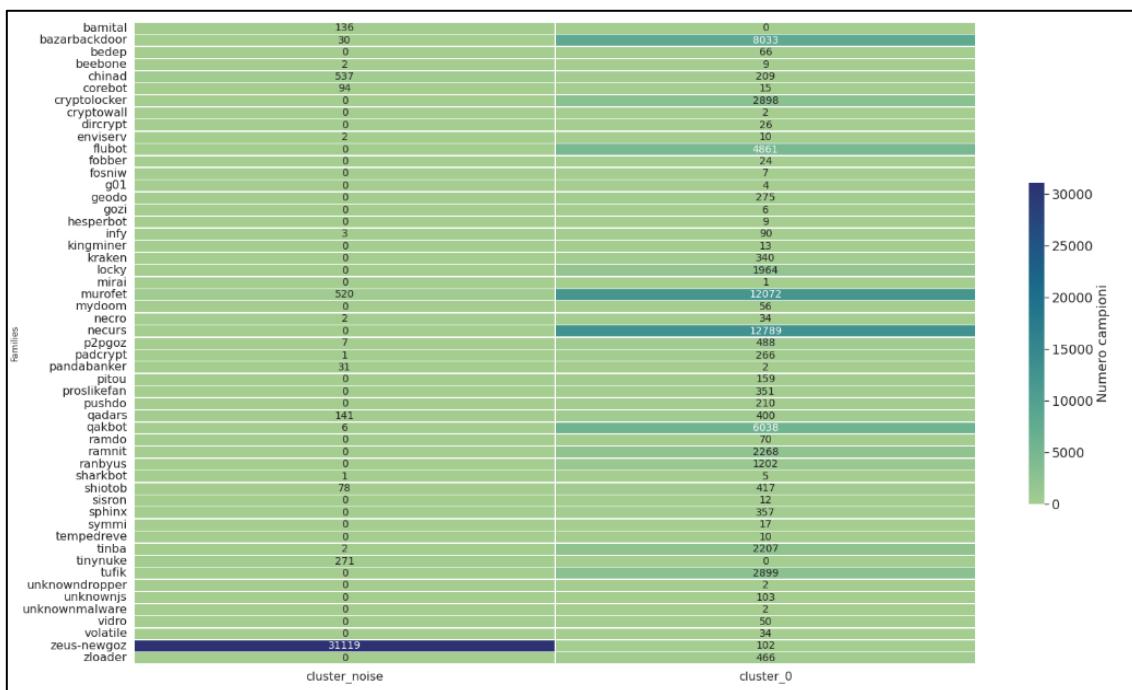


Figura 32: Matrice di confusione dim=2 minPoints=95 eps=2.25

Invece, con le configurazioni $dim = 3$, $epochs = 20$, $minPoints = 142$, $eps = 2.75$ ([Figura 33](#)), il DBSCAN produce due cluster (*cluster_0* e *cluster_1*). All'interno del primo sono presenti campioni provenienti da famiglie differenti, similmente al caso precedente, e quindi la *precision* delle singole famiglie in questo cluster si abbassa sensibilmente. Nel secondo cluster sono presenti solo campioni appartenenti alla classe *dyre*, però tale famiglia è presente anche nel *cluster_noise*. In aggiunta, molti punti di varie famiglie, tra cui *zeus-newgoz*, vengono inclusi nel rumore.

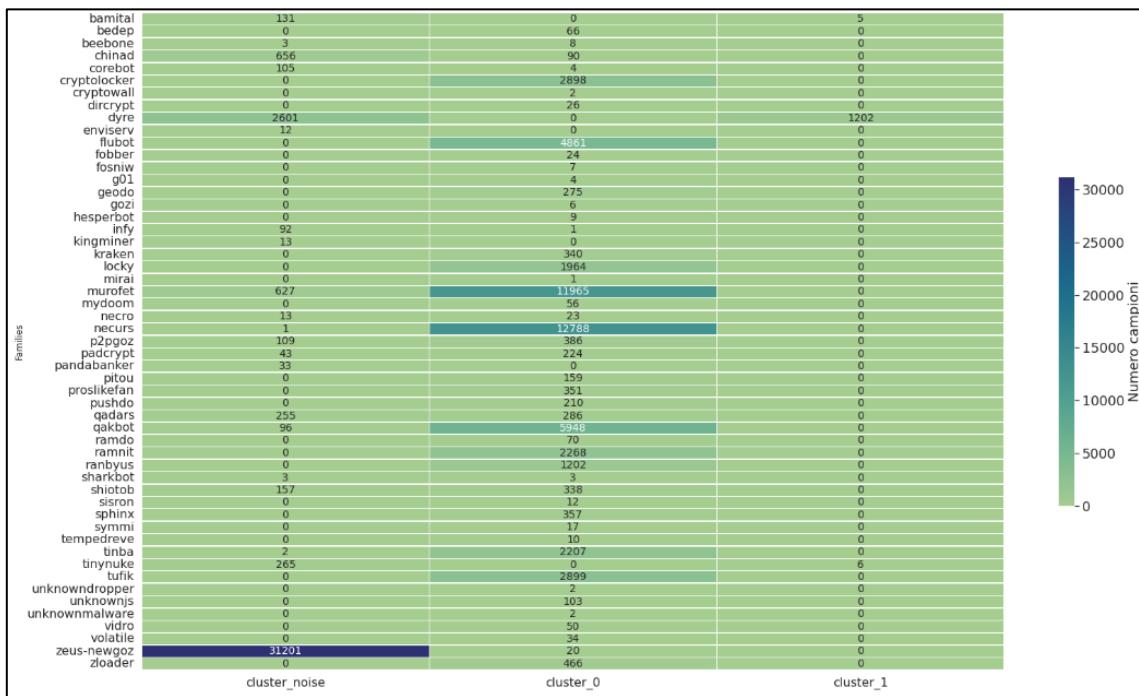


Figura 33: Matrice di confusione $dim=3$ $minPoints=142$ $eps=2.75$

Nel caso di $dim = 4$, $epochs = 20$, $minPoints = 189$, $eps = 3.0$ ([Figura 34](#)), si presenta un solo cluster, come accadeva per la dimensione 2, in cui sono presenti le stesse famiglie viste precedentemente. Si può sottolineare il fatto che gli elementi della famiglia *dyre* vengono catalogati come rumore, in cui vengono incluse anche altre famiglie, similmente ai casi precedenti. Dunque, questa analisi approfondita mostra che le famiglie eliminate non influenzano negativamente la segmentazione, che probabilmente dipende solamente dalle variazioni dei parametri liberi, in particolare *dim*, *minPoints* ed *eps*.

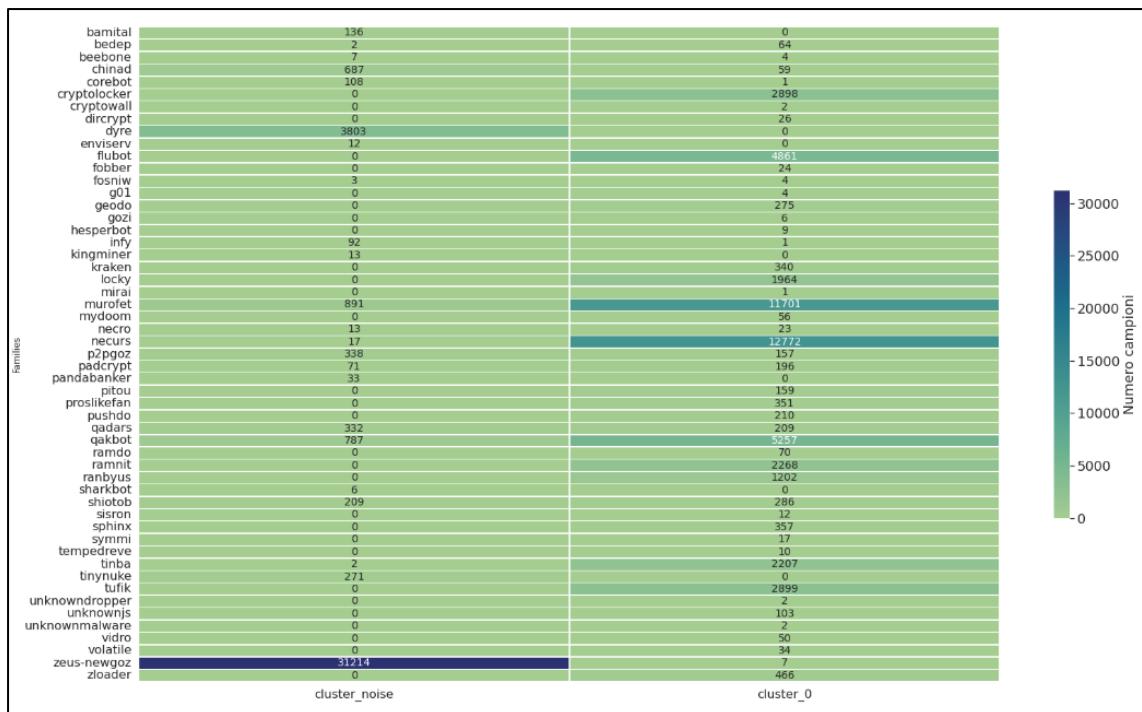


Figura 34: Matrice di confusione dim=4 minPoints=189 eps=3.0

CONCLUSIONI

Tramite l'analisi effettuata sul dataset a disposizione è stato possibile verificare la distribuzione dei campioni e la presenza di pattern all'interno delle famiglie, utilizzando la tecnica di segmentazione del DBSCAN. Le diverse configurazioni adottate hanno permesso di rilevare le caratteristiche delle famiglie e il loro comportamento al variare dei parametri. In aggiunta, l'utilizzo della segmentazione rappresenta una tecnica alternativa rispetto a quella utilizzata tipicamente in questo campo, ovvero la classificazione, spesso con reti neurali. Uno degli obiettivi di questo lavoro era studiare come un approccio non supervisionato si comportasse in questo dominio rispetto ai più classici approcci supervisionati.

I risultati ottenuti dagli esperimenti eseguiti sono stati negativamente influenzati dalle limitazioni computazionali riscontrate. Per poter analizzare più approfonditamente la distribuzione delle famiglie sarebbe necessaria una quantità di memoria molto maggiore di quella a disposizione durante i test. La prima limitazione è data dal valore della dimensione dei vettori in uscita dall'elaborazione di *FastText*. Infatti, la dimensione di default proposta da *FastText* è 128, ma in molti casi è richiesto che questo numero cresca al fine di migliorare le performance. Tuttavia, considerando che ogni singola parola del dataset è composta da varie decine di bigrammi, la memoria, occupata dal dataset trasformato in vettori, cresce così velocemente all'aumentare della dimensione che diventa in poco tempo ingestibile. Analoghe limitazioni sono state riscontrate anche nella scelta del parametro *eps*, scelto attraverso la tecnica euristica illustrata in precedenza. Purtroppo, venivano forniti valori di *eps* troppo grandi, i quali causavano un eccessivo riempimento della memoria, per costruire le rappresentazioni dei cluster.

Oltre ad aumentare la potenza di calcolo a disposizione, un'altra possibile soluzione è rappresentata dall'addestramento e dall'utilizzo di un Encoder: trovando uno spazio equivalente a quello di partenza, ma di dimensione minore, si può diminuire il numero di feature in ingresso al DBSCAN, in modo da poter semplificare i calcoli. In questo modo, nello spazio di arrivo, il clustering risulta essere meno pesante dal punto di vista computazionale, e conseguentemente si possono utilizzare parametri più grandi. Naturalmente, per tornare al dominio di partenza e verificare il risultato è necessario il Decoder.

In conclusione, sebbene non si possano escludere performance migliori con parametri più adatti, i risultati ottenuti da tale approccio non sono incoraggianti. Difatti, per diverse combinazioni dei parametri non solo è variata la bontà della segmentazione, ma sono proprio cambiate le famiglie che l'algoritmo riusciva a identificare. Se tale fenomeno fosse caratteristico dell'approccio e non fosse causato dalle limitazioni riscontrate, significherebbe che il clustering non rappresenta una strada praticabile. Infatti, sarebbe necessaria una combinazione di parametri per ciascuna famiglia da isolare e, di conseguenza, il processo ne risulterebbe fin troppo complesso e poco adattabile alla comparsa di nuove famiglie di malware in futuro.