Diario degli esperimenti

Lorenzo Tiseni

Settembre 2021

1 Introduzione

In questo documento terremo conto di tutti gli esperimenti che sono stati fatti sinora sia per il training con FastText sia per il test della rete LSTM. La piattaforma utilizzata è `google colab` di cui specificheremo i vari parametri nei vari esperimenti

2 Esperimenti FastText

2.1 Primo Training FastText

Di seguito si riportano tutte le informazioni riguardo i vari strumenti e risultati ottenuti nel training eseguito il 9 settembre 2021 a partire dalle ore 17:12:

2.1.1 Google Colab

L'esperimento è stato eseguito utilizzando la TPU di google colab. Nel runtime ospitato utilizzato sono stati messi a disposizione 12 GB di RAM

2.1.2 Dataset

Il dataset utilizzato per l'esperimento è stato costruito con lo script python PreProcessingFastText.py che per prima cosa ha selezionato 18 milioni di nomi di dominio circa da un dataset con 130 milioni di nomi compresi i duplicati che erano separati da andata a capo. Dopodichè lo script ha rimosso utti i domain name non ben formati selezionando solo quelli che contenevano underscore, trattini, lettere e numeri. Infine sono stati rimossi i punti contenuti all'interno dei nomi e inserito uno spazio tra ogni carattere creando così il dataset

2.1.3 Risultati

Lo script con cui sono stati ottenuti i risultati sperimentali è FastText.py e il modello di fasttext scelto per questo primo esperimento è stato il modello skipgram. Il training ha impiegato circa 3 ore e 43 minuti e le epoche di training fatte sono state 20. Il risultato prodotto dall'esperimento è stato un file .vec

ben formato e conforme con quanto ci si aspettava. I caratteri trovati sono stati 65 e ad ognuno di essi sono stati associati vettori numerici con 128 componenti.

2.2 Secondo Training FastText

Di seguito si riportano tutte le informazioni riguardo i vari strumenti e risultati ottenuti nel training eseguito l'11 settembre 2021 a partire dalle ore 11:30:

2.2.1 Google Colab

L'esperimento è stato eseguito utilizzando la CPU di google colab. Nel runtime ospitato utilizzato sono stati messi a disposizione 12 GB di RAM

2.2.2 Dataset

Il dataset utilizzato stavolta, nonostante è stato ottenuto dallo stessa fase di preprocessing che ha eliminato i domini malformati, eliminato i punti, e scelto solo nomi di dominio unici, non contiene su ogni riga un nome di dominio i cui caratteri sono separati da spazio ma contiene per ogni nome di dominio i bigrammi del nome in questione

2.2.3 Risultati

Lo script con cui sono stati ottenuti i risultati sperimentali è FastText.py e il modello di fasttext scelto per questo primo esperimento è stato il modello skipgram. Il training ha impiegato circa 10 ore e 10 minuti e le epoche di training fatte sono state 5. Il risultato prodotto dall'esperimento è stato un file .vec ben formato e conforme con quanto ci si aspettava. I Bigrammi trovati sono stati 4093 e ad ognuno di essi sono stati associati vettori numerici con 128 componenti.

2.3 Terzo Training FastText

Di seguito si riportano tutte le informazioni riguardo i vari strumenti e risultati ottenuti nel training eseguito il 12 settembre 2021 a partire dalle ore 9:01:

2.3.1 Google Colab

L'esperimento è stato eseguito utilizzando la CPU di google colab. Nel runtime ospitato utilizzato sono stati messi a disposizione 12 GB di RAM

2.3.2 Dataset

Il dataset utilizzato per l'esperimento è stato costruito con lo script python PreProcessingFastText.py che per prima cosa ha selezionato 8 milioni di nomi di dominio circa da un dataset con 32 milioni di nomi compresi i duplicati che erano separati da andata a capo. Dopodichè lo script ha rimosso utti i domain name

non ben formati selezionando solo quelli che contenevano underscore, trattini, lettere e numeri. Infine sono stati rimossi i punti contenuti all'interno dei nomi e inserito uno spazio tra ogni carattere creando così il dataset da 8 milioni di nomi. Il fatto che questo dataset è più piccolo viene fatto solo per risparmiare tempo

2.3.3 Risultati

Lo script con cui sono stati ottenuti i risultati sperimentali è FastText.py e il modello di fasttext scelto per questo primo esperimento è stato il modello cbow. Il training ha impiegato circa 38 minuti e le epoche di training fatte sono state 20. Il risultato prodotto dall'esperimento è stato un file .vec ben formato e conforme con quanto ci si aspettava. I Caratteri trovati sono stati 65 e ad ognuno di essi sono stati associati vettori numerici con 128 componenti.

2.4 Quarto Training FastText

Di seguito si riportano tutte le informazioni riguardo i vari strumenti e risultati ottenuti nel training eseguito il 12 settembre 2021 a partire dalle ore 20:55:

2.4.1 Google Colab

L'esperimento è stato eseguito utilizzando la CPU di google colab. Nel runtime ospitato utilizzato sono stati messi a disposizione 12 GB di RAM

2.4.2 Dataset

Il dataset utilizzato per l'esperimento è stato costruito con lo script python PreProcessingFastText.py che per prima cosa ha selezionato 8 milioni di nomi di dominio circa da un dataset con 32 milioni di nomi compresi i duplicati che erano separati da andata a capo. Dopodichè lo script ha rimosso utti i domain name non ben formati selezionando solo quelli che contenevano underscore, trattini, lettere e numeri. Infine sono stati rimossi i punti contenuti all'interno dei nomi e inserito uno spazio tra ogni carattere creando così il dataset da 8 milioni di nomi. Il fatto che questo dataset è più piccolo viene fatto solo per risparmiare tempo

2.4.3 Risultati

Lo script con cui sono stati ottenuti i risultati sperimentali è FastText.py e il modello di fasttext scelto per questo esperimento è stato il modello cbow. Il training ha impiegato circa 7 ore e le epoche di training fatte sono state 20. Il risultato prodotto dall'esperimento è stato un file .vec ben formato e conforme con quanto ci si aspettava. I Bigrammi trovati sono stati 4091 e ad ognuno di essi sono stati associati vettori numerici con 128 componenti.

3 Esperimenti LSTM

3.1 Esperimenti con dataset 1-80.csv

3.1.1 Esperimento con random embedding

Il primo esperimento condotto è stato quello con l'embedding randomico, come fatto già da Duc Tran, però solo per il caso multiclasse. L'esperimento è stato fatto usando la k-fold cross-validation e iterando per 5 fold, in ognuno dei quali il classificatore veniva addestrato su 20 epoche. Il dataset di training conteneva circa 4200 nomi di dominio e si è deciso di dare al test dataset una grandezza del 20% del totale e al training dataset l'80%.Nell'addestramento del classificatore multiclasse sono stati aggiunti i pesi per il multiclass imbalance. Il tempo impiegato è stato circa 9 minuti usando la TPU su google colab. Ad ogni fold veniva stampato il report della classificazione multiclasse. Dopodichè con i dati contenuti in tutti i fold è stata fatta la media, per ottenere un report totale che sintetizzasse il training svolto. I risultati di questo report totale sono presentati nell'immagine sotto:

Class report:

	precision	recall	f1-score	support
conficker	0.544444	0.247059	0.306333	17.000000
corebot	0.957310	0.976471	0.966003	17.000000
cryptolocker	0.311126	0.288235	0.292053	16.000000
dircrypt	0.301236	0.176471	0.216364	17.000000
emotet	0.746709	0.800000	0.758195	17.000000
fobber	0.441168	0.709559	0.534029	16.000000
gozi	0.400000	0.107353	0.161111	17.000000
kraken	0.500000	0.155882	0.226560	17.000000
matsnu	0.679444	0.541176	0.575947	17.000000
murofet	0.822367	0.658824	0.724963	17.000000
necurs	0.565714	0.119853	0.186950	17.000000
nymaim	0.000000	0.000000	0.000000	16.000000
padcrypt	0.796049	0.905882	0.841415	17.000000
pushdo	0.764923	0.450735	0.541547	17.000000
pykspa	0.596032	0.204412	0.289992	17.000000
qadars	0.686923	0.870588	0.759994	17.000000
ramdo	0.803893	0.903676	0.848690	16.000000
ramnit	0.470000	0.261029	0.333854	17.000000
ranbyus	0.414438	0.330882	0.353892	17.000000
rovnix	0.517782	0.786765	0.594062	16.000000
simda	0.629924	0.497059	0.536034	17.000000
suppobox	0.599286	0.420588	0.464029	17.000000
symmi	0.923947	0.962500	0.941221	17.000000
tinba	0.370810	0.358824	0.358367	17.000000
vawtrak	0.200000	0.011765	0.022222	17.000000
alexa	0.771083	0.936190	0.845030	420.000000
	Precision	Recall	F1_score	support
accuracy				0.703333
macro avg	0.569793	0.487761	0.487648	840.000000
weighted avg	0.666714	0.703333	0.659573	840.000000

	macro	micro
f1_score	0.487648	0.703333
precision	0.569793	0.703333
recall	0.487761	0.703333

Overall accuracy = 0.7033333333333334

Figure 1: Total report random embedding 1-80.csv

Osserviamo innanzitutto che il classificatore, riesce a distinguere abbastanza bene i nomi di dominio in malevoli e benevoli. Infatti notiamo che la famiglia alexa, cioè quella dei domini benevoli, ottiene una precision di circa 0.77, il che vuol dire che, su tutti i nomi di dominio predetti come alexa dalla rete, il'77% effettivamente lo erano. Il punteggio di recall risulta essere ottimo, circa 0.94, il che vuol dire che su tutte le istanze di nomi alexa esistenti, il 94% viene predetto correttamente. Dati questi due valori anche l'F1-score per la famiglia alexa risulta alto.

Venendo invece alle famiglie di DGA notiamo che il classificatore ottiene buone performance specialmente con le famiglie corebot, emotet, qadars, ramdo, padcrypt e symmi, per cui abbiamo un punteggio di F1-score pari o superiore a 0.75. In particolare le famiglie corebot e symmi ottengono punteggi di precision e recall molto alti, mentre le altre famiglie a una precision inferiore compensano una minor recall e viceversa. Ad esempio qadars ha un'alta recall, cioè l'87% di tutte le istanze di nomi appartenenti a qadars sono predette come qadars, però ha una più bassa precision, cioè solo il 68% di tutte le predizioni di nomi come qadars sono effettivamente qadars. Esistono però famiglie che il classificatore predice in maniera quasi completamente errata, arrivando addirittura ad avere un punteggio di F1-score inferiore a 0.2, come ad esempio le famiglie gozi, necurs, nymaim, e vawtrak. In particolare la famiglia nymaim(questo dipende anche magari dal fatto che il dataset è piccolo), ha un punteggio di F1-score pari a 0, e allo stesso tempo precision pari a 0 e recall pari a 0. Ciò significa che la rete non classificare correttamente tale famiglia di malware. Dato questo problema si è deciso di guardare la matrice di confusione relativa al fold in cui tale famiglia otteneva il punteggio di F1-score migliore.

Notiamo dalla figura sopra che su 17 esemplari della famiglia nymaim nessuno è riconosciuto come tale: 9 sono identificati come appartenenti ad altre famiglie di malware, cioè tinba e simda e infine ben 15 sono classificati come alexa. Quest'ultima cosa è preoccupante poiché circa l'88% dei nomi di dominio nymaim è ritenuto benevolo. Tale cosa dipende probabilmente dal fatto che il dataset è piccolo e i campioni della famiglia nymain sono particolarmente difficili da riconoscere, per la rete, se questa durante la fase di training non ne ha visti abbastanza. Comunque nymaim non è l'unica famiglia i cui nomi sono scambiati come benevoli. Anche molti di nomi delle famiglie gozi, pushdo, vawtrak e matsnu sono scambiati come alexa.

In conclusione osserviamo che i valori di macro-average e micro-average per tutte e tre le metriche, precision, recall e F1-score, non sono molto alti, proprio a causa di quelle famiglie riportate prima che vengono classificate male. L'accuracy totale del modello risulta essere del 70.3%, il che testimonia che le prestazioni sono discrete, ma sicuramente si potrebbero ottenere risultati migliori.

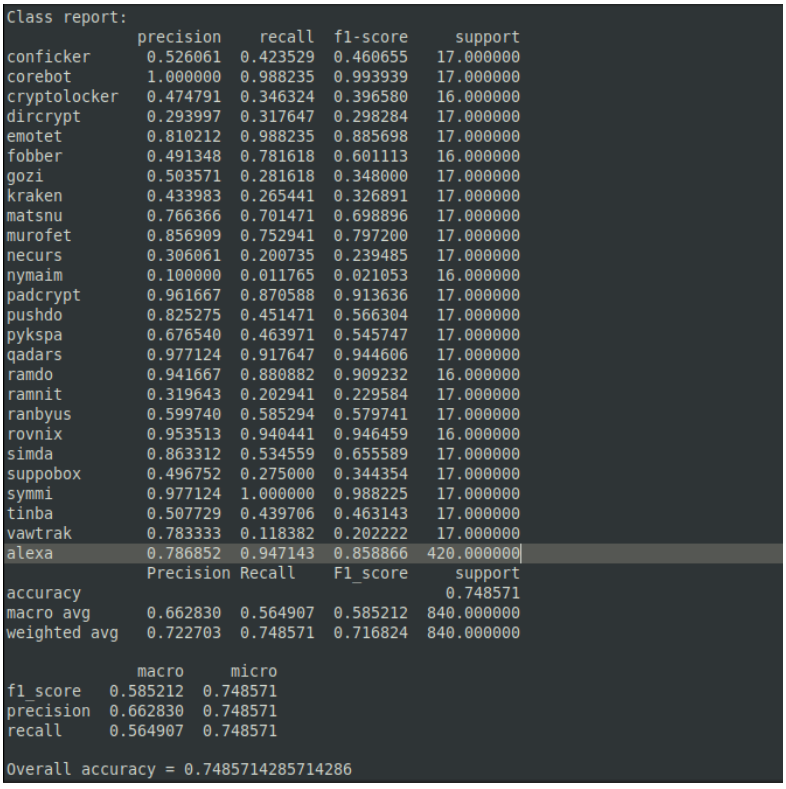
3.1.2 Esperimento con FastText embedding basato su caratteri

Il secondo esperimento condotto è stato quello con l'embedding basato su FastText, usando i caratteri, per il caso multiclasse. L'esperimento è stato fatto

	symmi	tinba	vawtrak	alexa
conficker	0	0	0	10
corebot	0	0	0	0
cryptolocker	0	5	0	0
dircrypt	0	1	0	2
emotet	0	0	0	0
fobber	0	0	0	3
gozi	0	0	0	14
kraken	0	1	0	5
matsnu	0	0	0	11
murofet	0	0	0	0
necurs	0	1	0	2
nymaim	0	1	0	15
padcrypt	0	0	0	1
pushdo	0	0	0	11
pykspa	0	2	0	2
qadars	0	0	0	1
ramdo	0	0	0	1
ramnit	0	0	0	2
ranbyus	0	2	0	0
rovnix	0	0	0	0
simda	0	0	0	4
suppobox	0	0	0	7
symmi	15	0	0	0
tinba	0	4	0	1
vawtrak	0	0	0	16
alexa	0	0	0	398

Figure 2: matrice di confusione fold 3 1-80.csv

usando la k-fold cross-validation e iterando per 5 fold, in ognuno dei quali il classificatore veniva addestrato su 20 epoche. Il dataset di training conteneva circa 4200 nomi di dominio e si è deciso di dare al test dataset una grandezza del 20% del totale, e al training dataset l'80%.Nell'addestramento del classificatore multiclasse sono stati aggiunti i pesi per il multiclass imbalance. Il tempo impiegato è stato circa 9 minuti e 44 secondi usando la TPU su google colab. Ad ogni fold veniva stampato il report della classificazione multiclasse. Dopodichè con i dati contenuti in tutti i fold è stata fatta la media, per ottenere un report totale che sintetizzasse il training svolto. I risultati di questo report totale sono presentati nell'immagine sotto:



Class report:

	precision	recall	f1-score	support
conficker	0.526061	0.423529	0.460655	17.000000
corebot	1.000000	0.988235	0.993939	17.000000
cryptolocker	0.474791	0.346324	0.396580	16.000000
dircrypt	0.293997	0.317647	0.298284	17.000000
emotet	0.810212	0.988235	0.885698	17.000000
fobber	0.491348	0.781618	0.601113	16.000000
gozi	0.503571	0.281618	0.348000	17.000000
kraken	0.433983	0.265441	0.326891	17.000000
matsnu	0.766366	0.701471	0.698896	17.000000
murofet	0.856909	0.752941	0.797200	17.000000
necurs	0.306061	0.200735	0.239485	17.000000
nymaim	0.100000	0.011765	0.021053	16.000000
padcrypt	0.961667	0.870588	0.913636	17.000000
pushdo	0.825275	0.451471	0.566304	17.000000
pykspa	0.676540	0.463971	0.545747	17.000000
qadars	0.977124	0.917647	0.944606	17.000000
ramdo	0.941667	0.880882	0.909232	16.000000
ramnit	0.319643	0.202941	0.229584	17.000000
ranbyus	0.599740	0.585294	0.579741	17.000000
rovnix	0.953513	0.940441	0.946459	16.000000
simda	0.863312	0.534559	0.655589	17.000000
suppobox	0.496752	0.275000	0.344354	17.000000
symmi	0.977124	1.000000	0.988225	17.000000
tinba	0.507729	0.439706	0.463143	17.000000
vawtrak	0.783333	0.118382	0.202222	17.000000
alexa	0.786852	0.947143	0.858866	420.000000
	Precision	Recall	F1_score	support
accuracy				0.748571
macro avg	0.662830	0.564907	0.585212	840.000000
weighted avg	0.722703	0.748571	0.716824	840.000000

	macro	micro
f1_score	0.585212	0.748571
precision	0.662830	0.748571
recall	0.564907	0.748571

Overall accuracy = 0.7485714285714286

Figure 3: Total report FastText embedding basato su caratteri 1-80.csv

Rispetto al caso precedente, notiamo subito che c'è un piccolo miglioramento nella classificazione dei nomi in benevoli e malevoli. Infatti il classificatore riesce a riconoscere leggermente meglio i domini benevoli. Aumentano di poco(circa 0.01)precision, recall e F1-score, per la famiglia alexa.

La cosa interessante da notare però risiede nella classificazione multiclasse legata ai nomi DGA. Notiamo innanzitutto che il classificatore continua a ottenere ottime prestazioni con le famiglie corebot, emotet, qadars, ramdo, padcrypt e symmi, arrivando a raggiungere punteggi di F1-score superiori a 0.9,

fatta eccezione per emotet, che comunque mantiene un F1-score sotto soglia ma vicino(0.88). Rispetto al caso precedente assistiamo inoltre a una crescita di circa 0.35 in F1-score per la famiglia rovnix, che si aggiunge a quelle per cui la rete performa meglio. Notiamo inoltre che l'embedding con FastText genera aumenti di prestazioni in generale per tutte le famiglie, comprese quelle per cui nel caso randomico ottenevamo performance inferiori. Degna di nota è la famiglia pykspa che fa registrare un raddoppio in F1-score andando a superare 0.5, un raddoppio della recall, e un aumento di 0.08 per la precision. Anche per la famiglia gozi c'è un raddoppio in F1-score, e la famiglia vawtrak cresce di 0.2 in F1-score. Purtroppo otteniamo ancora punteggi troppo bassi per le famiglie dircrypt, necurs, nymaim, ramnit(che addirittura rimane stabile e peggiora lievemente) e vawtrak stessa che ottengono un punteggio di F1-score inferiore a 0.3. In particolare notiamo che notiamo che per la famiglia nymaim abbiamo un F1-score pessimo e simile al caso precedente. Per questo è importante analizzare la matrice di confusione:

	symmi	tinba	vawtrak	alexa
conficker	0	0	0	10
corebot	0	0	0	0
cryptolocker	0	0	0	3
dircrypt	0	0	0	3
emotet	0	0	0	0
fobber	0	1	0	3
gozi	0	0	0	9
kraken	0	0	0	6
matsnu	0	0	0	4
murofet	0	0	0	0
necurs	0	2	0	3
nymaim	0	0	0	16
padcrypt	0	0	0	2
pushdo	0	0	0	10
pykspa	0	1	0	6
qadars	0	0	0	1
ramdo	0	0	0	2
ramnit	0	0	0	4
ranbyus	1	0	0	1
rovnix	0	0	0	1
simda	0	0	0	7
suppobox	0	0	0	10
symmi	16	0	0	0
tinba	0	5	0	1
vawtrak	0	0	1	16
alexa	0	0	0	406

Figure 4: matrice di confusione FastTextChars fold 5 1-80.csv

In conclusione osserviamo che i punteggi di macro-average e micro-average per tutte e tre le metriche, precision, recall e F1-score, si alzano grazie ai miglioramenti ottenuti. Le macro average crescono per tutte e tre le metriche di 0.1.

Le micro-average invece superano 0.7 per tutte e tre le metriche . L'accuracy totale del modello risulta essere del 75%, il che testimonia che le prestazioni migliorano abbastanza usando un embedding FastText basato su caratteri addestrato con dati reali.

3.1.3 Esperimento con FastText embedding basato su bigrammi

Il terzo esperimento condotto è stato quello con l'embedding basato su FastText, usando i bigrammi, per il caso multiclassificatore senza andare ad addestrare il classificatore binario. L'esperimento è stato fatto usando la k-fold cross-validation e iterando per 5 fold, in ognuno dei quali il classificatore veniva addestrato su 20 epoche. Il dataset di training conteneva circa 4200 nomi di dominio e si è deciso di dare al test dataset una grandezza del 20% del totale, e al training dataset l'80%.Nell'addestramento del classificatore multiclasse sono stati aggiunti i pesi per il multiclass imbalance. Il tempo impiegato è stato circa 10 minuti e 30 secondi usando la TPU su google colab. Ad ogni fold veniva stampato il report della classificazione multiclasse. Dopodichè con i dati contenuti in tutti i fold è stata fatta la media, per ottenere un report totale che sintetizzasse il training svolto. I risultati di questo report totale sono presentati nell'immagine sotto:

Rispetto ai casi precedenti, c'è un miglioramento nella classificazione dei nomi in benevoli e malevoli. Infatti il classificatore riesce a riconoscere leggermente meglio i domini benevoli aumentando la precision per la famiglia alexa di circa il 0.07 rispetto all'embedding basato su caratteri e di circa 0.08 rispetto all'embedding randomico. La recall resta sostanzialmente uguale. Di conseguenza l'F1-score è cresciuto di 0.03

Per quanto riguarda la classificazione multiclasse dei domini in famiglie di DGA, vediamo un miglioramento nel complesso, sia rispetto al'embedding FastText basato su caratteri, sia rispetto all'embedding randomico. La cosa importante da sottolineare nel passaggio da caratteri a bigrammi è che, le metriche, per alcune famiglie per cui ottenevamo risultati bassi nel caso precedente, migliorano sensibilmente. Esempio è la famiglia vawtrak per la quale aumenta la precision di 0.05, quintuplica la recall, e triplica l'F1-score arrivando a superare 0.6. Oltre a questa, tra le famiglie per cui prima ottenevamo punteggi pessimi o comunque non soddisfacenti, assistiamo a un miglioramento anche per gozi, kraken(che raddopia F1-score e aumenta precision e recall), necurs(raddopia F1-score e recall), nymaim, pushdo(per cui arriviamo ad ottenere un F1-score ottimo e pari a 0.82), pykspa e suppobox. Da registrare nel passaggio da caratteri a bigrammi però anche dei peggioramenti nelle metriche per alcune famiglie: otteniamo dei peggioramenti, specialmente in recall, per famiglie come qadars, rovnix(che comunque mantiene ottime prestazioni al livello globale) e fobber. Mentre per qadars e rovnix i peggioramenti sono contenuti, per fobber il peggioramento è vistoso in termini di recall. Ciò significa che la rete su tutte le istanze della famiglia fobber ne riconosce meno come fobber. Otteniamo dei peggioramenti in precision specialmente per murofet, ranbyus che però sono contenuti. Infine conficker e tinba peggiorano di poco sia in recall che in precision. Per

```
Class report:
              precision    recall  f1-score     support
conficker      0.518182  0.400000  0.448127   17.000000
corebot        1.000000  0.988235  0.993939   17.000000
cryptolocker   0.462103  0.394853  0.401992   16.000000
dircrypt       0.362491  0.329412  0.299953   17.000000
emotet         0.977124  0.964706  0.970400   17.000000
fobber         0.434804  0.373529  0.395364   16.000000
gozi           0.736401  0.491912  0.530669   17.000000
kraken         0.621885  0.619118  0.617411   17.000000
matsnu         0.807546  0.683824  0.712597   17.000000
murofet        0.769195  0.741176  0.743075   17.000000
necurs         0.448146  0.464706  0.444894   17.000000
nymaim         0.472857  0.178676  0.253771   16.000000
padcrypt       0.962402  0.870588  0.912326   17.000000
pushdo         0.839945  0.819853  0.824298   17.000000
pykspa         0.784786  0.737500  0.744074   17.000000
qadars         0.974167  0.870588  0.918903   17.000000
ramdo          0.884676  0.964706  0.919318   16.000000
ramnit         0.389892  0.345588  0.355263   17.000000
ranbyus        0.538066  0.595588  0.555586   17.000000
rovnix         0.956656  0.854412  0.897967   16.000000
simda          0.779524  0.688235  0.720073   17.000000
suppobox       0.552329  0.508088  0.516485   17.000000
symmi          1.000000  1.000000  1.000000   17.000000
tinba          0.389744  0.332353  0.332736   17.000000
vawtrak        0.733846  0.512500  0.602829   17.000000
alexa          0.855608  0.927143  0.889292  420.000000
               Precision Recall    F1_score     support
accuracy                                       0.778571
macro avg      0.702014  0.640665  0.653898  840.000000
weighted avg   0.775720  0.778571  0.767182  840.000000

              macro     micro
f1_score   0.653898  0.778571
precision  0.702014  0.778571
recall     0.640665  0.778571

Overall accuracy = 0.7785714285714286
```

Figure 5: Total report FastText embedding basato su bigrammi

quanto rigurda nymaim essendo una delle famiglie che ancora mantiene un valore molto basso e inferiore a 0.3 per F1-score siamo andati a vedere la matrice di confusione:

	symmi	tinba	vawtrak	alexa
conficker	0	0	0	6
corebot	0	0	0	0
cryptolocker	0	3	0	3
dircrypt	0	2	0	1
emotet	0	0	0	0
fobber	0	1	2	2
gozi	0	0	0	5
kraken	0	1	1	4
matsnu	0	0	0	5
murofet	0	1	0	0
necurs	0	3	0	2
nymaim	0	0	0	10
padcrypt	0	0	0	2
pushdo	0	0	0	5
pykspa	0	0	1	0
qadars	0	1	0	1
ramdo	0	0	0	0
ramnit	0	3	1	1
ranbyus	0	3	0	1
rovnix	0	0	0	0
simda	0	0	0	4
suppobox	0	0	0	8
symmi	16	0	0	0
tinba	0	6	0	3
vawtrak	0	0	5	12
alexa	0	0	2	401

Figure 6: matrice di confusione FastTextBigrams fold 5 1-80.csv

Ancora purtroppo notiamo che su 17 campioni nymaim, nel caso migliore, sono classificati come benevoli 10 nomi e cioè circa il 59% del totale. Ciò conferma ancora una volta le pessime prestazioni della rete nel riconoscimento di nomi appartenenti a questa famiglia.

Al livello globale notiamo che comunque l'uso di embedding FastText basato su bigrammi migliora le prestazioni rispetto ai due casi precedenti. Crescono sia le macro-average, che le micro-average per tutte e 3 le metriche. L'uso dei bigrammi in particolare causa un grande aumento della recall rispetto alla precision nelle macro-average e un aumento sostanziale della precision per quanto riguarda la micro-average. l'accuracy del modello in totale cresce ancora di 0.03 arrivando a sfiorare l'80%.

3.2 Esperimenti con dataset 1-40.csv

3.2.1 Esperimento con random embedding

Il primo esperimento condotto è stato quello con l'embedding randomico, come fatto già da Duc Tran, però solo per il caso multiclasse. L'esperimento è stato fatto usando la k-fold cross-validation e iterando per 5 fold, in ognuno dei quali il classificatore veniva addestrato su 20 epoche. Il dataset di training conteneva circa 8400 nomi di dominio e si è deciso di dare al test dataset una grandezza del 20% del totale e al training dataset l'80%.Nell'addestramento del classificatore multiclasse sono stati aggiunti i pesi per il multiclass imbalance. Il tempo impiegato è stato circa 11 minuti e 5 secondi usando la TPU su google colab. Ad ogni fold veniva stampato il report della classificazione multiclasse. Dopodichè con i dati contenuti in tutti i fold è stata fatta la media, per ottenere un report totale che sintetizzasse il training svolto. I risultati di questo report totale sono presentati nell'immagine sotto:

Class report:

	precision	recall	f1-score	support
conficker	0.616407	0.410695	0.491643	34.000
corebot	0.954902	0.988057	0.970926	34.000
cryptolocker	0.525671	0.443850	0.464572	33.000
dircrypt	0.489953	0.264706	0.326443	34.000
emotet	0.890346	0.982353	0.933345	34.000
fobber	0.561371	0.804635	0.653930	33.000
gozi	0.825000	0.226203	0.325123	34.000
kraken	0.738723	0.390731	0.507753	34.000
matsnu	0.759782	0.659180	0.679562	34.000
murofet	0.900948	0.691087	0.781004	34.000
necurs	0.622887	0.218895	0.317118	34.000
nymaim	0.486111	0.071658	0.117978	33.000
padcrypt	0.948836	0.863280	0.902307	34.000
pushdo	0.838593	0.615686	0.696061	34.000
pykspa	0.561459	0.398396	0.453049	33.000
qadars	0.931238	0.885561	0.907580	34.000
ramdo	0.916152	0.945989	0.929618	33.000
ramnit	0.340976	0.294118	0.308932	33.000
ranbyus	0.568776	0.665062	0.611443	34.000
rovnix	0.843522	0.898039	0.861751	33.000
simda	0.717809	0.645276	0.672765	34.000
suppobox	0.546751	0.526916	0.516191	34.000
symmi	0.964118	0.957754	0.959402	33.000
tinba	0.527637	0.611765	0.544128	33.000
vawtrak	0.562232	0.165241	0.244532	33.000
alexa	0.805842	0.945476	0.869495	840.000
	Precision	Recall	F1_score	support
accuracy				0.765
macro avg	0.709463	0.598870	0.617179	1680.000
weighted avg	0.755971	0.765000	0.738203	1680.000

	macro	micro
f1_score	0.617179	0.765
precision	0.709463	0.765
recall	0.598870	0.765

Overall accuracy = 0.765

Figure 7: Total report random embedding

Avendo utilizzato un nuovo dataset per questa serie di esperimenti, innanzitutto focalizziamoci su come cambiano le prestazioni, al crescere delle dimen-

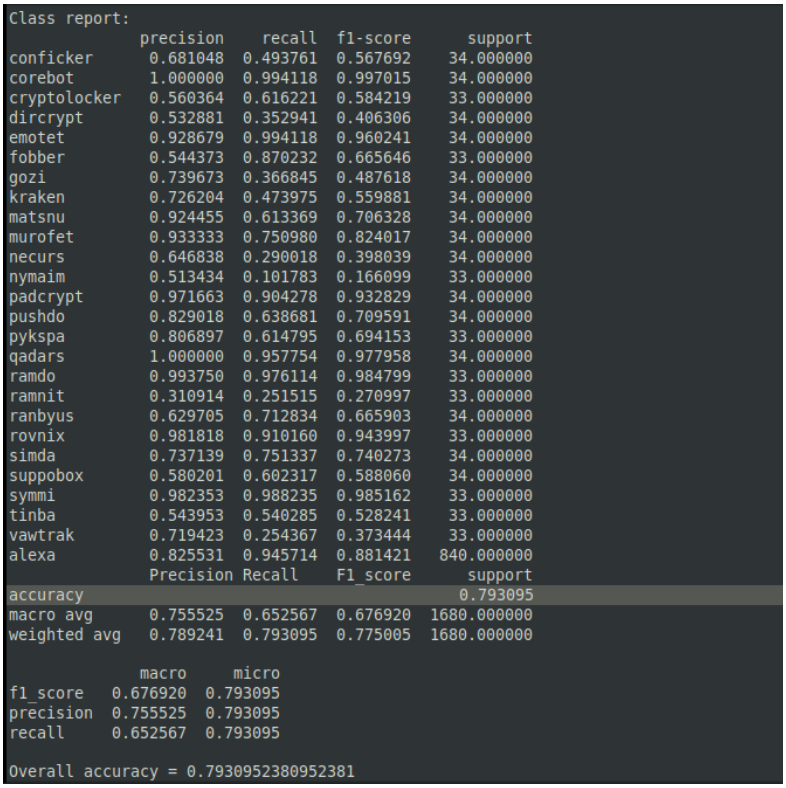
sioni del dataset. Confrontando i risultati della figura sopra e della figura 1, è chiaro che al crescere delle dimensioni del dataset c'è un sensibile aumento delle prestazioni, al livello globale per tutte le classi:

- La classificazione in nomi di dominio benevoli e malevoli ottiene ancora buoni punteggi, in quanto l'F1-score per la famiglia alexa risulta essere 0.87 circa, contro lo 0.84 dell'esperimento condotto con il dataset più piccolo.
- La classificazione multiclasse per i nomi DGA migliora di tanto: per le famiglie, per cui anche con $\frac{1}{80}$ del dataset si ottenevano ottime prestazioni, si continuano a ottenere prestazioni ottime, e superiori al caso precedente; contemporaneamente a questo si assiste anche a un miglioramento significativo delle prestazioni per famiglie di malware che avevano invece punteggi pessimi nel caso precedente. Esempi di questo sono conficker, cryptolocker, gozi, kraken, pykspa. Oltre a questo tra le famiglie per cui otteniamo ottimi risultati si aggiungono anche emotet e rovnix: quest'ultima in particolare registra un grosso aumento delle prestazioni con l'aumento del volume del dataset. Infine notiamo che solo per la famiglia ramnit le prestazioni rimangono simili al caso col dataset più piccolo.
- Nonostante un miglioramento generale nella classificazione multiclasse per i DGA rimangono delle famiglie che ottengono prestazioni pessime, come ad esempio nymaim(0.11 F1-score), vawtrak(0.24 F1-score). Tale cosa è dovuta al fatto che per entrambe le famiglie ci sono molti nomi di dominio(29 per nymaim e 25 per vawtrak) classificati come alexa(vedi fold 5 per vawtrak e fold 2 per nymaim che sono i casi migliori
- Al livello generale notiamo che per tutte e tre le metriche c'è un aumento significativo sia in termini di micro-average che di macro-average. Le macro-average aumentano tutte di almeno 0.11 e le micro-average invece crescono tutte di almeno 0.06. Notiamo anche inoltre che l'aumento per le macro-average è simile al caso in cui lasciamo costante il dataset e cambiamo embedding usando quello con FastText basato su bigrammi. Aumento significativo anche per l'accuracy che sale a 0.77 andando a equiparare anche l'accuracy del modello con FastText Embedding basato su caratteri, addestrato con $\frac{1}{80}$ del dataset.

3.2.2 Esperimento con FastText embedding basato su caratteri

Il secondo esperimento condotto è stato quello con l'embedding basato su FastText, usando i caratteri, per il caso multiclasse. L'esperimento è stato fatto usando la k-fold cross-validation e iterando per 5 fold, in ognuno dei quali il classificatore veniva addestrato su 20 epoche. Il dataset di training conteneva circa 8400 nomi di dominio e si è deciso di dare al test dataset una grandezza del 20% del totale, e al training dataset l'80%.Nell'addestramento del classificatore multiclasse sono stati aggiunti i pesi per il multiclass imbalance. Il tempo impiegato è stato circa 11 minuti e 20 secondi usando la TPU su google colab. Ad

ogni fold veniva stampato il report della classificazione multiclasse. Dopodichè con i dati contenuti in tutti i fold è stata fatta la media, per ottenere un report totale che sintetizzasse il training svolto. I risultati di questo report totale sono presentati nell'immagine sotto:



Class report:

	precision	recall	f1-score	support
conficker	0.681048	0.493761	0.567692	34.000000
corebot	1.000000	0.994118	0.997015	34.000000
cryptolocker	0.560364	0.616221	0.584219	33.000000
dircrypt	0.532881	0.352941	0.406306	34.000000
emotet	0.928679	0.994118	0.960241	34.000000
fobber	0.544373	0.870232	0.665646	33.000000
gozi	0.739673	0.366845	0.487618	34.000000
kraken	0.726204	0.473975	0.559881	34.000000
matsnu	0.924455	0.613369	0.706328	34.000000
murofet	0.933333	0.750980	0.824017	34.000000
necurs	0.646838	0.290018	0.398039	34.000000
nymaim	0.513434	0.101783	0.166099	33.000000
padcrypt	0.971663	0.904278	0.932829	34.000000
pushdo	0.829018	0.638681	0.709591	34.000000
pykspa	0.806897	0.614795	0.694153	33.000000
qadars	1.000000	0.957754	0.977958	34.000000
ramdo	0.993750	0.976114	0.984799	33.000000
ramnit	0.310914	0.251515	0.270997	33.000000
ranbyus	0.629705	0.712834	0.665903	34.000000
rovnix	0.981818	0.910160	0.943997	33.000000
simda	0.737139	0.751337	0.740273	34.000000
suppobox	0.580201	0.602317	0.588060	34.000000
symmi	0.982353	0.988235	0.985162	33.000000
tinba	0.543953	0.540285	0.528241	33.000000
vawtrak	0.719423	0.254367	0.373444	33.000000
alexa	0.825531	0.945714	0.881421	840.000000
	Precision	Recall	F1_score	support
accuracy				0.793095
macro avg	0.755525	0.652567	0.676920	1680.000000
weighted avg	0.789241	0.793095	0.775005	1680.000000

	macro	micro
f1_score	0.676920	0.793095
precision	0.755525	0.793095
recall	0.652567	0.793095

Overall accuracy = 0.7930952380952381

Figure 8: Total report FastText embedding 1-40

Confrontando questi risultati con quelli ottenuti utilizzando $\frac{1}{80}$ del dataset notiamo che c'è un miglioramento, conseguente all'aumento del volume del dataset. Questo vale per tutte le famiglie fatta eccezione per alcune, che semplicemente però sono famiglie per cui il punteggio già era alto con 1-80.csv e rimane alto e quasi invariato anche con 1.40.csv. Pericoloso è invece il caso di nymaim che viene per lo più interpretato come benevolo. Migliora anche la capacità della rete nel riconoscere nomi benevoli come tali, in quanto per la famiglia alexa aumentano sia precision, sia recall. Chiaramente dato quanto detto aumentano anche macro e micro per tutte e tre le metriche, e aumenta anche l'accuracy.

Confrontiamo ora i risultati con quelli ottenuti nel caso dell'embedding randomico con la stessa taglia di dataset. Come nel caso del dataset 1-80.csv anche con questo dataset assistiamo a una crescita in tutte e 3 le metriche per tutte le famiglie che abbiamo. La rete con l'embedding FastText riesce a riconoscere

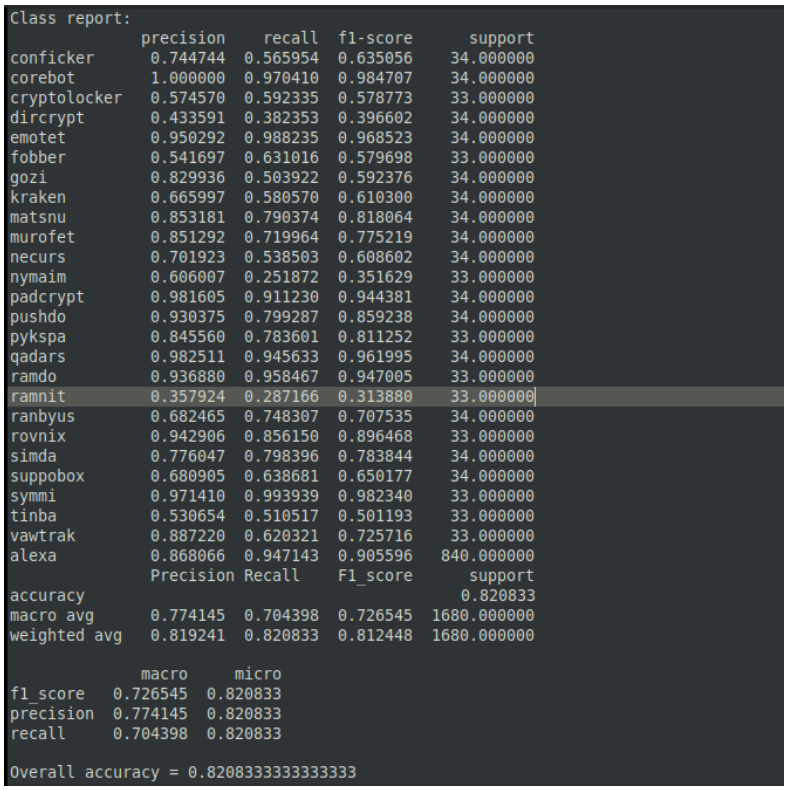
meglio i domini benevoli come tali anche in questo caso. PEr la famiglia alexa abbiamo infatti un aumento in precision, recall stabile e un aumento in F1-score. Diminuiscono perciò il numero di falsi positivi rispetto al caso con embedding randomico.

Dal punto di vista della classificazione multiclasse in famiglie di DGA, vediamo generalmente precision, recall e F1-score aumentano per tutte le famiglie. Come già visto nel caso del dataset 1-80.csv, si assiste in particolare a un grosso miglioramento per le famiglie rovnix e pykspa. Rovnix infatti diventa una delle famiglie per cui la rete ha prestazioni migliori, e pykspa migliora sensibilmente andando a diventare una delle famiglie, per cui la classificazione funziona discretamente al pari di murofet, pushdo e ranbyus. Inoltre come nel caso del dataset precedente anche con questo vediamo che non ci sono grossi miglioramenti per la famiglia ramnit a seguito del cambio di embedding. Le prestazioni della rete per tale famiglia rimangono quasi inalterate rispetto al caso con embedding random. Inoltre ramnit è una delle famiglie, insieme a nymaim e vawtrak, per cui le prestazioni della rete sono mediocri(sotto 0.4 F1-score). Guardando le matrici di confusione, si osserva anche in questo caso che i nomi di dominio delle famiglie vawtrak e nymaim sono scambiati come nomi alexa(rispettivamente 30 per nymaim e 24 per vawtrak nel fold 2 che è quello in cui entrambe hanno F1-score migliore).

Si conferma ancora una volta, nel passaggio da embedding randomico a FastText embedding basato su caratteri, la crescita delle macro e micro average per tutte e 3 le metriche, con aumenti inferiori rispetto al caso in cui il dataset era più piccolo. Aumenta anche l'accuracy di 0.03 arrivando a toccare 0.793 in questo caso.

3.2.3 Esperimento con FastText embedding basato su bigrammi

Il terzo esperimento condotto è stato quello con l'embedding basato su FastText, usando i bigrammi, per il caso multiclassificatore senza andare ad addestrare il classificatore binario. L'esperimento è stato fatto usando la k-fold cross-validation e iterando per 5 fold, in ognuno dei quali il classificatore veniva addestrato su 20 epoche. Il dataset di training conteneva circa 8400 nomi di dominio e si è deciso di dare al test dataset una grandezza del 20% del totale, e al training dataset l'80%.Nell'addestramento del classificatore multiclasse sono stati aggiunti i pesi per il multiclass imbalance. Il tempo impiegato è stato circa 12 minuti e 19 secondi usando la TPU su google colab. Ad ogni fold veniva stampato il report della classificazione multiclasse. Dopodichè con i dati contenuti in tutti i fold è stata fatta la media, per ottenere un report totale che sintetizzasse il training svolto. I risultati di questo report totale sono presentati nell'immagine sotto:



Class report:

	precision	recall	f1-score	support
conficker	0.744744	0.565954	0.635056	34.000000
corebot	1.000000	0.970410	0.984707	34.000000
cryptolocker	0.574570	0.592335	0.578773	33.000000
dircrypt	0.433591	0.382353	0.396602	34.000000
emotet	0.950292	0.988235	0.968523	34.000000
fobber	0.541697	0.631016	0.579698	33.000000
gozi	0.829936	0.503922	0.592376	34.000000
kraken	0.665997	0.580570	0.610300	34.000000
matsnu	0.853181	0.790374	0.818064	34.000000
murofet	0.851292	0.719964	0.775219	34.000000
necurs	0.701923	0.538503	0.608602	34.000000
nymaim	0.606007	0.251872	0.351629	33.000000
padcrypt	0.981605	0.911230	0.944381	34.000000
pushdo	0.930375	0.799287	0.859238	34.000000
pykspa	0.845560	0.783601	0.811252	33.000000
qadars	0.982511	0.945633	0.961995	34.000000
ramdo	0.936880	0.958467	0.947005	33.000000
ramnit	0.357924	0.287166	0.313880	33.000000
ranbyus	0.682465	0.748307	0.707535	34.000000
rovnix	0.942906	0.856150	0.896468	33.000000
simda	0.776047	0.798396	0.783844	34.000000
suppobox	0.680905	0.638681	0.650177	34.000000
symmi	0.971410	0.993939	0.982340	33.000000
tinba	0.530654	0.510517	0.501193	33.000000
vawtrak	0.887220	0.620321	0.725716	33.000000
alexa	0.868066	0.947143	0.905596	840.000000
	Precision	Recall	F1_score	support
accuracy				0.820833
macro avg	0.774145	0.704398	0.726545	1680.000000
weighted avg	0.819241	0.820833	0.812448	1680.000000

	macro	micro
f1_score	0.726545	0.820833
precision	0.774145	0.820833
recall	0.704398	0.820833

Overall accuracy = 0.8208333333333333

Figure 9: Total report FastText embedding Bigrams 1-40

Confrontando i risultati con quelli ottenuti con la taglia più piccola del dataset, si vede un miglioramento globale per tutte le famiglie, fatta eccezione per alcune che mantengono prestazioni simili(dipende dal fatto che le famiglie

che ottengono alte prestazioni difficilmente tra un esperimento e un altro migliorano sensibilmente). L'unica per cui le prestazioni non migliorano e rimangono basse sembra essere ramnit nel caso dei bigrammi.

Passando al confronto con il caso precedente e cioè embedding FastText basato su caratteri sottolineamo subito una cosa. Come accadeva con la precedente taglia del dataset assistiamo a un aumento della precision di 0.04 rispetto all'embedding FastText basato su caratteri, e di 0.06 rispetto all'embedding randomico, per la famiglia alexa. Ciò vuol dire che la rete riesce sempre di più a generare meno falsi positivi per questa famiglia il che è un bene. La recall rimane invariata più o meno nei tre casi e di conseguenza quindi l'F1-score aumenta arrivando nel caso dei bigrammi a 0.91.

Andando alla classificazione multiclasse dei DGA il passaggio da caratteri a bigrammi genera certamente grossi miglioramenti. In particolare si assiste a un miglioramento delle prestazioni della rete specialmente per alcune famiglie(conficker,gozi,kraken, matsnu, necurs, nymaim, pushdo, pykspa, simda, suppobox e vawtrak) tra cui in particolare ci sono vawtrak, necurs, pushdo e pykspa. Infatti vawtrak per cui ottenevamo pessime prestazioni nel caso dell'embedding randomico arriva ad ottenere un F1-score superiore a 0.7(precision tendente all'88% e recall sopra il 60%). Lo stesso accade per pykspa e pushdo le quali superano entrambe 0.8 per F1-score(cita le precision e le recall). In ultimo osserviamo che per necurs, per cui ottenevamo pessime prestazioni nei casi precedenti, otteniamo ora un punteggio di F1-score superiore a 0.6. A questi miglioramenti, però segue nel passaggio ai bigrammi un degrado delle prestazioni per le famiglie murfet, ramdo, rovnix e fobber. Mentre le prime tre comunque le prestazioni della rete peggiorano di poco oppure, rimangono abbastanza elevate, per la famiglia fobber assistiamo a un peggioramento più marrcato di circa il 10%in F1-score rispetto al sia al caso con embedding FastText basato su caratteri, sia al caso randomico. A parte queste famiglie per il resto delle famiglie di malware nel passaggio da caratteri a bigrammi le prestazioni della rete rimangono stabili. La famiglia con l'F1-score più basso rimane sempre nymaim, che insieme a dircrypt non supera 0.4 di F1-score. Guardando le matrici di confusione si vede infatti che, nel caso migliore, 18 nomi di dominio nymaim sono scambiati come alexa, e quindi nymaim continua a essere confusa con la famiglia benevola. Invece per quanto riguarda dircrypt notiamo che pochi nomi, solo 4, sono scambiati per nomi alexa. Il resto viene attratto da altre famiglie di malware specialmente ramnit.

Globalmente assistiamo comunque a un miglioramento sia delle micro che delle macro average. La macro average cresce per tutte le metriche almeno di 0.02 e l'aumento più alto si ha in recall. La micro average cresce almeno di 0.03 per tutte le metriche. L'accuracy del modello sale di 0.03 rispetto al caso con i caratteri arrivando a 0.82.

3.3 Esperimenti con dataset 1-20.csv

3.3.1 Esperimento con random embedding

Il primo esperimento condotto è stato quello con l'embedding randomico, come fatto già da Duc Tran, però solo per il caso multiclasse. L'esperimento è stato fatto usando la k-fold cross-validation e iterando per 5 fold, in ognuno dei quali il classificatore veniva addestrato su 20 epoche. Il dataset di training conteneva circa 16800 nomi di dominio e si è deciso di dare al test dataset una grandezza del 20% del totale e al training dataset l'80%.Nell'addestramento del classificatore multiclasse sono stati aggiunti i pesi per il multiclass imbalance. Il tempo impiegato è stato circa 15 minuti usando la TPU su google colab. Ad ogni fold veniva stampato il report della classificazione multiclasse. Dopodichè con i dati contenuti in tutti i fold è stata fatta la media, per ottenere un report totale che sintetizzasse il training svolto. I risultati di questo report totale sono presentati nell'immagine sotto:

```
Class report:
              precision    recall  f1-score      support
conficker      0.730543  0.507243  0.592298    68.000000
corebot        0.973991  0.997015  0.985314    67.000000
cryptolocker   0.672442  0.581782  0.622470    67.000000
dircrypt       0.650342  0.348683  0.415884    68.000000
emotet         0.967647  0.970588  0.968748    68.000000
fobber         0.686593  0.762818  0.717460    67.000000
gozi           0.743459  0.493942  0.542020    67.000000
kraken         0.819813  0.584284  0.677495    68.000000
matsnu         0.896279  0.730290  0.793616    67.000000
murofet        0.917927  0.721071  0.799332    68.000000
necurs         0.698863  0.438806  0.535568    67.000000
nymaim         0.676354  0.125241  0.203178    67.000000
padcrypt       0.984826  0.914486  0.947853    68.000000
pushdo         0.893400  0.710448  0.786997    67.000000
pykspa         0.819796  0.753731  0.774786    67.000000
qadars         0.977871  0.905092  0.939809    68.000000
ramdo          0.973200  0.943415  0.957644    67.000000
ramnit         0.388573  0.469842  0.405820    67.000000
ranbyus        0.799966  0.781299  0.788848    68.000000
rovnix         0.987292  0.901888  0.942309    67.000000
simda          0.846841  0.719315  0.773545    68.000000
suppobox       0.681371  0.485601  0.545870    68.000000
symmi          0.993884  0.982090  0.987784    67.000000
tinba          0.578535  0.720983  0.631064    67.000000
vawtrak        0.840748  0.330158  0.460163    67.000000
alexa          0.833130  0.954777  0.889667  1685.000000
               Precision Recall    F1_score      support
accuracy                                       0.815015
macro avg      0.808988  0.685957  0.718675  3370.000000
weighted avg   0.820596  0.815015  0.800783  3370.000000

              macro     micro
f1_score   0.718675  0.815015
precision  0.808988  0.815015
recall     0.685957  0.815015

Overall accuracy = 0.8150148367952522
```

Figure 10: Total report random embedding

Avendo utilizzato un nuovo dataset per questa serie di esperimenti, innanzitutto focalizziamoci su come cambiano le prestazioni, al crescere delle dimen-

sioni del dataset. Confrontando i risultati della figura sopra e della figura 1, è chiaro che al crescere delle dimensioni del dataset c'è un sensibile aumento delle prestazioni, al livello globale per tutte le classi:

- La classificazione in nomi di dominio benevoli e malevoli ottiene ancora buoni punteggi, in quanto l'F1-score per la famiglia alexa risulta essere 0.89 circa, contro lo 0.87 dell'esperimento condotto con il dataset 1-40.csv.

- La classificazione multiclasse per i nomi DGA migliora di tanto: per le famiglie, per cui anche con $\frac{1}{80}$ del dataset si ottenevano ottime prestazioni, si continuano a ottenere prestazioni ottime, e superiori al caso precedente; contemporaneamente a questo si assiste anche a un miglioramento significativo delle prestazioni per famiglie di malware che avevano invece punteggi mediocri nel caso precedente. Esempi di questo sono cryptolocker, gozi, kraken, matsnu, necurs, vawtrak e pykspa. Oltre a questo tra le famiglie per cui otteniamo ottimi risultati si aggiungono anche emotet e rovnix: quest'ultima in particolare registra un buon aumento delle prestazioni con l'aumento del volume del dataset anche in questo caso.

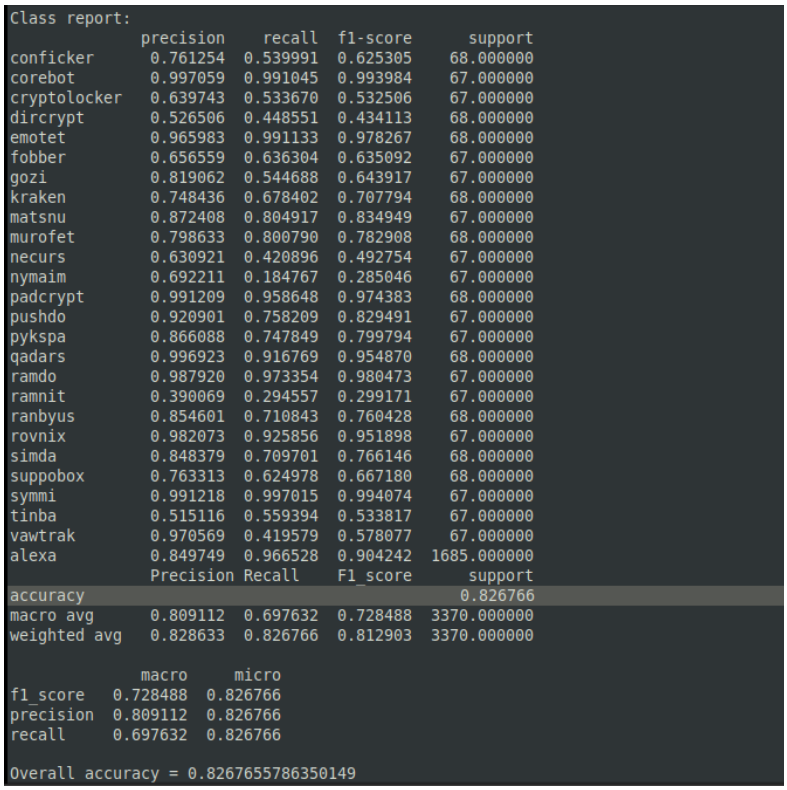
- Nonostante un miglioramento generale nella classificazione multiclasse per i DGA rimangono delle famiglie che ottengono prestazioni pessime, come ad esempio nymaim(0.2 F1-score). Tale cosa è dovuta al fatto che per nymaim ci sono molti nomi di dominio, 52, classificati come alexa(vedi fold 2 1-20)

- Al livello generale notiamo che per tutte e tre le metriche c'è un aumento significativo sia in termini di micro-average che di macro-average. Le macro-average aumentano tutte di almeno 0.09 e le micro-average invece crescono tutte di almeno 0.05. Aumento significativo anche per l'accuracy che sale a 0.815 andando a superare anche l'accuracy del modello con FastText Embedding basato su caratteri, addestrato con $\frac{1}{40}$ del dataset e quasi ad equiparare il modello con FastText FastText Embedding basato su bigrammi, addestrato con $\frac{1}{40}$ del dataset che ha accuracy di 0.82.

3.3.2 Esperimento con FastText embedding basato su caratteri

Il secondo esperimento condotto è stato quello con l'embedding basato su FastText, usando i caratteri, per il caso multiclasse. L'esperimento è stato fatto usando la k-fold cross-validation e iterando per 5 fold, in ognuno dei quali il classificatore veniva addestrato su 20 epoche. Il dataset di training conteneva circa 16800 nomi di dominio e si è deciso di dare al test dataset una grandezza del 20% del totale, e al training dataset l'80%.Nell'addestramento del classificatore multiclasse sono stati aggiunti i pesi per il multiclass imbalance. Il tempo impiegato è stato circa 15 minuti usando la TPU su google colab. Ad ogni fold veniva stampato il report della classificazione multiclasse. Dopodichè con i dati contenuti in tutti i fold è stata fatta la media, per ottenere un report totale che

sintetizzasse il training svolto. I risultati di questo report totale sono presentati nell'immagine sotto:



```
Class report:
              precision    recall  f1-score      support
conficker      0.761254  0.539991  0.625305    68.000000
corebot        0.997059  0.991045  0.993984    67.000000
cryptolocker   0.639743  0.533670  0.532506    67.000000
dircrypt       0.526506  0.448551  0.434113    68.000000
emotet         0.965983  0.991133  0.978267    68.000000
fobber         0.656559  0.636304  0.635092    67.000000
gozi           0.819062  0.544688  0.643917    67.000000
kraken         0.748436  0.678402  0.707794    68.000000
matsnu         0.872408  0.804917  0.834949    67.000000
murofet        0.798633  0.800790  0.782908    68.000000
necurs         0.630921  0.420896  0.492754    67.000000
nymaim         0.692211  0.184767  0.285046    67.000000
padcrypt       0.991209  0.958648  0.974383    68.000000
pushdo         0.920901  0.758209  0.829491    67.000000
pykspa         0.866088  0.747849  0.799794    67.000000
qadars         0.996923  0.916769  0.954870    68.000000
ramdo          0.987920  0.973354  0.980473    67.000000
ramnit         0.390069  0.294557  0.299171    67.000000
ranbyus        0.854601  0.710843  0.760428    68.000000
rovnix         0.982073  0.925856  0.951898    67.000000
simda          0.848379  0.709701  0.766146    68.000000
suppobox       0.763313  0.624978  0.667180    68.000000
symmi          0.991218  0.997015  0.994074    67.000000
tinba          0.515116  0.559394  0.533817    67.000000
vawtrak        0.970569  0.419579  0.578077    67.000000
alexa          0.849749  0.966528  0.904242  1685.000000
               Precision Recall    F1_score      support
accuracy                                       0.826766
macro avg      0.809112  0.697632  0.728488  3370.000000
weighted avg   0.828633  0.826766  0.812903  3370.000000

              macro     micro
f1_score   0.728488  0.826766
precision  0.809112  0.826766
recall     0.697632  0.826766

Overall accuracy = 0.8267655786350149
```

Figure 11: Total report FastText Embedding Chars 1-20.csv

Confrontando questi risultati con quelli ottenuti utilizzando $\frac{1}{80}$ del dataset notiamo che c'è un miglioramento, conseguente all'aumento del volume del dataset. Questo vale per tutte le famiglie fatta eccezione per alcune, che semplicemente però sono famiglie per cui il punteggio già era alto con 1-40.csv e rimane alto e quasi invariato anche con 1.20.csv e per alcune famiglie per cui avvengono piccole fluttuazioni come cryptlocker e fobber. Migliora anche la capacità della rete nel riconoscere nomi benevoli come tali, in quanto per la famiglia alexa aumentano sia precision, sia recall. Chiaramente dato quanto detto aumentano anche macro e micro per tutte e tre le metriche, e aumenta anche l'accuracy.

Confrontando ora i risultati ottenuti con embedding FastText basato sui caratteri, con quelli provenienti dall'embedding randomico, con la stessa taglia di dataset notiamo una cosa importante. Globalmente si assiste a un miglioramento delle prestazioni per molte famiglie di malware come conficker, corebot, gozi, nymaim, padcrypt, pushdo, pykspa, qadars, ramdo, suppobox e vawtrak. In particolare, e questa cosa sembra essere una costante, migliorano ancora una

volta di molto le prestazioni relative alla famiglia vawtrak cambiando tipo di embedding, oltre che di un'altra famiglia che è suppobox. Sostanzialmente invece rimangono stabili le prestazioni della rete per altre famiglie come dircrypt, emotet, kraken, matsnu, murofet, ranbyus, rovnix, simda e symmi. Assistiamo questa volta a un degrado delle prestazioni della rete per le famiglie di malware come cryptolocker, fobber, necurs, ramnit e tinba. Mentre per cryptlocker, necurs e tinba è la prima volta che si registra questo fenomeno al cambiare dell'embedding, notiamo che invece per fobber avevamo già assistito a un fenomeno del genere ma cambiando da FastText con i caratteri a FastText con i bigrammi. Invece per quanto riguarda ramnit già nei casi precedenti avevamo osservato che il cambio di embedding non aumentava di molto l prestazioni e quindi è possibile che per questa famiglia qualche volta nel cambio di embedding si possa riscontrare un peggioramento. Infine notiamo che la famiglia di malware per cui si ottengono prestazioni più basse da parte della rete risulta essere sempre nymaim.

Non peggiorano invce le prestazioni della rete per i nomi benevoli appartenenti alla famiglia alexa. Anche in questo caso il cambio di embedding fa salire le prestazioni sia per precision che recall.

Concludiamo dicendo che a differenza del caso con le taglie di dataset più piccole, in questo caso il passaggio dall'embedding randomico a quello con FastText non rappresenta un beneficio enorme sia per le micro average di tutte e 3 le metriche, che per le macro average di tutte e 3 le metriche. Anche l'accuracy non sale di molto infatti passa da 0.815 a 0.827.

3.3.3 Esperimento con FastText embedding basato su bigrammi

Il terzo esperimento condotto è stato quello con l'embedding basato su FastText, usando i bigrammi, per il caso multiclasse. L'esperimento è stato fatto usando la k-fold cross-validation e iterando per 5 fold, in ognuno dei quali il classificatore veniva addestrato su 20 epoche. Il dataset di training conteneva circa 16800 nomi di dominio e si è deciso di dare al test dataset una grandezza del 20% del totale, e al training dataset l'80%.Nell'addestramento del classificatore multiclasse sono stati aggiunti i pesi per il multiclass imbalance. Il tempo impiegato è stato circa 9 minuti usando la TPU su google colab. Ad ogni fold veniva stampato il report della classificazione multiclasse. Dopodichè con i dati contenuti in tutti i fold è stata fatta la media, per ottenere un report totale che sintetizzasse il training svolto. I risultati di questo report totale sono presentati nell'immagine sotto:

Class report:

	precision	recall	f1-score	support
conficker	0.753315	0.581782	0.649670	68.000000
corebot	0.991089	0.973134	0.981670	67.000000
cryptolocker	0.677373	0.650044	0.656069	67.000000
dircrypt	0.434184	0.298727	0.349601	68.000000
emotet	0.974364	0.979324	0.976526	68.000000
fobber	0.599390	0.580114	0.575058	67.000000
gozi	0.763379	0.554434	0.616917	67.000000
kraken	0.851288	0.634021	0.725762	68.000000
matsnu	0.803008	0.751668	0.775977	67.000000
murofet	0.789738	0.753687	0.760478	68.000000
necurs	0.652361	0.573134	0.607313	67.000000
nymaim	0.760930	0.452371	0.563128	67.000000
padcrypt	0.964892	0.961589	0.963085	68.000000
pushdo	0.938921	0.865672	0.900085	67.000000
pykspa	0.874158	0.875461	0.874154	67.000000
qadars	0.993939	0.955487	0.974224	68.000000
ramdo	0.972947	0.961326	0.967070	67.000000
ramnit	0.315087	0.470237	0.376464	67.000000
ranbyus	0.805665	0.751668	0.777569	68.000000
rovnix	0.971094	0.877963	0.921847	67.000000
simda	0.902536	0.858033	0.877091	68.000000
suppobox	0.829054	0.703863	0.759751	68.000000
symmi	0.979832	0.988104	0.983855	67.000000
tinba	0.614176	0.535953	0.568075	67.000000
vawtrak	0.886676	0.753117	0.811058	67.000000
alexa	0.890682	0.957982	0.922977	1685.000000
	Precision	Recall	F1_score	support
accuracy				0.845757
macro avg	0.807311	0.742265	0.765980	3370.000000
weighted avg	0.847347	0.845757	0.841314	3370.000000

	macro	micro
f1_score	0.765980	0.845757
precision	0.807311	0.845757
recall	0.742265	0.845757

Overall accuracy = 0.8457566765578635

Figure 12: Total report FastText Embedding Bigrams 1-20.csv

Confrontando i risultati con quelli ottenuti con la taglia 1-40 del dataset, si vede un miglioramento globale per tutte le famiglie, fatta eccezione per alcune che mantengono prestazioni simili(dipende dal fatto che le famiglie che ottengono alte prestazioni difficilmente tra un esperimento e un altro miglio-

rano sensibilmente) e alcune che subiscono fluttuazioni oppure per cui anche coi cambi di embedding non si riesce ad aumentare le prestazioni(es. fobber, dircrypt). L'unica per cui le prestazioni non migliorano di tanto e rimangono basse sembra essere ramnit nel caso dei bigrammi.

Passando al confronto con il caso precedente e cioè embedding FastText basato su caratteri sottolineamo subito una cosa. Come accadeva con la precedente taglia del dataset assistiamo a un aumento della precision di 0.05 rispetto all'embedding FastText basato su caratteri, e di 0.06 rispetto all'embedding randomico, per la famiglia alexa. Ciò vuol dire che la rete riesce sempre di più anche in questo caso a generare meno falsi positivi per questa famiglia il che è un bene. La recall rimane invariata più o meno nei tre casi e di conseguenza quindi l'F1-score aumenta arrivando nel caso dei bigrammi a 0.923.

Andando alla classificazione multiclasse dei DGA il passaggio da caratteri a bigrammi genera certamente grossi miglioramenti. In particolare si assiste a un miglioramento delle prestazioni della rete specialmente per alcune famiglie(conficker, cryptolocker, necurs, nymaim, pushdo, pykspa, ramnit, simda, suppobox e vawtrak) tra cui in particolare ci sono vawtrak, nymaim, e necurs. Infatti vawtrak per cui ottenevamo pessime prestazioni nel caso dell'embedding randomico arriva ad ottenere un F1-score superiore a 0.81(precision tendente all'89% e recall sopra il 75%). Lo stesso accade per nymaim e necurs le quali superano entrambe 0.55 per F1-score(cita le precision e le recall). A questi miglioramenti, però segue nel passaggio ai bigrammi un degrado delle prestazioni per le famiglie matsnu, dircrypt e fobber. Mentre per la prima comunque le prestazioni della rete peggiorano di poco, e rimangono abbastanza elevate, per la famiglia fobber assistiamo a un peggioramento più marcato di circa il 6%in F1-score rispetto al caso con embedding FastText basato su caratteri, e del 14% rispetto al caso con embedding randomico. Per quanto riguarda le prestazioni relative alla famiglia dircrypt esse peggiorano del 7% in F1-score portandolo ad essere inferiore a 0.4. A parte queste famiglie per il resto delle famiglie di malware nel passaggio da caratteri a bigrammi le prestazioni della rete rimangono stabili in quanto per molte comunque anche cambiando embedding non si riescono ad ottenere aumenti esorbitanti. La famiglia con l'F1-score più basso in questo caso è dircrypt che non supera 0.4 di F1-score. Nonostante ciò questo avviene principalmente perchè a famiglia dircrypt viene attratta da altre famiglie di malware come ramnit. Anche le prestazioni basse di ramnit dipendono dal fatto che essa viene attratta da altre famiglie di malware come Fobber. Pochi nomi di entrambe le famiglie sono scambiati come nomi alexa.

In conclusione osserviamo anche qui che con le taglie di dataset più grandi il cambio di embedding in questo caso da caratteri a bigrammi non produce aumenti esorbitanti sia per le macro che per le micro average di tutte e 3 le metriche. Tale cosa vale anche per l'accuracy che nel passaggio da caratteri a bigrammi cresca da 0.827 a 0.846

3.4 Conclusioni esperimenti

Riportiamo analizzati tutti gli esperimenti delle brevi conclusioni

- L'aumento del volume del dataset produce un miglioramento delle prestazioni della rete relativamente a tutte le famiglie e globalmente. probabilmente però superata una certa taglia comunque l'aumento di prestazioni si arresterà e ci saranno delle fluttuazioni per alcune famiglie(esempi sono murofet e cryptolocker a volte)

- Uno strato di embedding basato su FastText sia che si usino i caratteri, sia che si usino i bigrammi è sempre migliore di uno strato di embedding randomico sia per il binario che per il multiclasse. L'aumento di prestazioni decresce con l'aumento del volume del dataset. Più il dataset è piccolo e più FastText aumenta le prestazioni. Più il dataset è grande e meno FastText aumenta le prestazioni.

- Globalmente le prestazioni della rete aumentano se si passa da uno strato di embedding con FastText basato su caratteri, a uno strato di embedding con FastText basato su bigrammi

- Non tutte le famiglie di malware beneficiano del cambio di embedding(da random a FastText). Ad esempio per la famiglia vawtrak migliorano molto le prestazioni se passiamo da random a FastText basato su caratteri, e se passiamo da FastText basato su caratteri a FastText basato su bigrammi. Per la famiglia fobber ad esempio no è detto che il cambio di embedding produca un miglioramento(con tutte e 3 le taglie per fobber peggiorano le prestazioni passando da Fasttext basato su caratteri a Fasttext basato su bigrammi).

- Per alcune famiglie di malware le prestazioni della rete non cambiano linearmente quando ci sono cambi di embedding o cambi di taglia del dataset. Esempio lampante è la famiglia ramnit per cui a volte abbiamo visto aumenti e altre volte degradi delle prestazioni in risposta o a cambi di volume del dataset o a cambi di embedding.

- nymaim, qualunuque sia lo strato di embedding e qualunque sia la taglia del dataset sperimentata rimane sempre la famiglia con più nomi di dominio che vengono classificati come alexa.