Klasifikacija zlonamjernog softvera

Stanišić Matea
stmatea@student.math.hr

Škrabo Petra
skpetra@student.math.hr

Terzanović Mateja
matterz@student.math.hr

Tolja Margarita
tomarga@student.math.hr

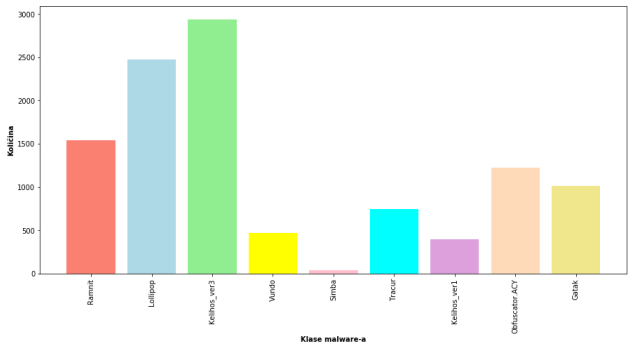
***Sažetak*—Ovaj rad nastao je kao odgovor na *Microsoft Malware Classification Challenge* [7] koji je 2015. godine predstavljen na *Kaggle*-u. Cilj zadatka bio je izrada modela strojnog učenja kojim bi se *malware*-i reprezentirani svojim .bytes i .asm datotekama klasificirali u jednu od 9 poznatih grupa. Ključna faza rješavanja problema sastojala se od dobrog odabira značajki. Model smo izradili koristeći XGBoost algoritam.**

I. Uvod

Jedna od najbolje financiranih i najvažnijih *IT* industrija današnjice zasigurno je ona koja se bavi zlonamjernim softverima (eng. *malware*). Većina *malware*-a kreira se konstantnim „malicioznim“ mijenjanjem već postojećih, što dovodi do velikog broja različitih datoteka koje predstavljaju istu kategoriju *malware*-a, tj. imaju isti obrazac malicioznog ponašanja. Ogromna količina podataka koje treba provjeriti predstavlja jedan od najvećih problema s kojima se susreću tvrtke za internetsku sigurnost te stoga na važnost dolazi klasifikacija *malware*-a.

II. Podaci

Microsoft je na *Kaggle*-u objavio skup podataka od oko pola `TB` koji sadrži više od 20 000 primjeraka *malware*-a podijeljenih na podatke za treniranje i testiranje. Podaci za treniranje imaju specificiranu jednu od devet klasa kojima mogu pripadati (*Ramnit, Lollipop, Kelihos_ver3, Vundo, Simba, Tracur, Kelihos_ver1, Obfuscator.ACY, Gatak*). Na slici 1 možemo vidjeti raspodjele količine podataka *train* skupa po klasama *malware*-a.



Slika 1. Rapodjela količine podataka iz *train* skupa po klasama.

Svaki *malware* je identificiran jedinstvenim *hash* kodom te je reprezentiran dvjema datotekama – heksadecimalnom reprezentacijom binarnog sadržaja (*.bytes* datoteka) i datotekom koja sadrži metapodatke (funkcijski pozivi, stringovi) izvučene iz binarnih podataka korištenjem *Interactive Disassembler*-a (*.asm* datoteka).

Primjer linije iz jedne *.bytes* datoteke:

```
00401000 56 8D 44 24 08 50 8B F1 E8 1C 1B 00 00 C7 06 08
```

Prvi broj, npr. `00401000` označava početak adrese desno napisane heksadecimalne reprezentacije binarnog koda u memoriji. Ostale, desne, heksadecimalne vrijednosti predstavljaju važne informacije o samom *malware*-u, npr. instrukcije u kodu.

To se može vidjeti i u primjeru linije iz jedne *.asm* datoteke:

```
.text:00401026 E9 26 1C 00 00 jmp  sub\_402C51
```

Kako originalni podaci nemaju točno određene značajke koje bi upotrijebili u algoritmu za klasificiranje, rudarili smo podatke po uzoru na članak [1] koristeći njihov objavljeni kod, uz male izmjene.

A. Značajke izvučene iz .bytes datoteka

1) ***Unigrami****:* Po definiciji iz [4], N-gram je sekvencija određene duljine koju sačinjavaju znakovi ili riječi koje se pojavljuju unutar teksta. Kao svojstvo, najčešće se koristi pri analizi teksta. Unigram (1G) je jedna riječ, bigram je sekvencija od dvije riječi, trigram je sekvencija od tri riječi itd.

Radi smanjenja složenosti, iz podatka smo izrudarili samo informacije o unigramima, to jest o frekvencijama svakog od 257 byte-ova (256 + dodatni bajt `??` koji reprezentira da taj byte u memoriji nije povezan sa izvršnom datotekom, odnosno da taj prostor u memoriji uopće nije inicijaliziran).

Ovim značajkama svrha je opisati strukturu programa. Detaljnije o tome možemo vidjeti u [13].

2) ***Metapodaci****:* Pod ovim pojmom mislimo na veličinu *.bytes* datoteke te adresu prve bit sekvence (dakle, npr. 00401000 iz opisanoga primjera) koju smo pretvorili u dekadski oblik radi homogenosti sa drugim značajkama.

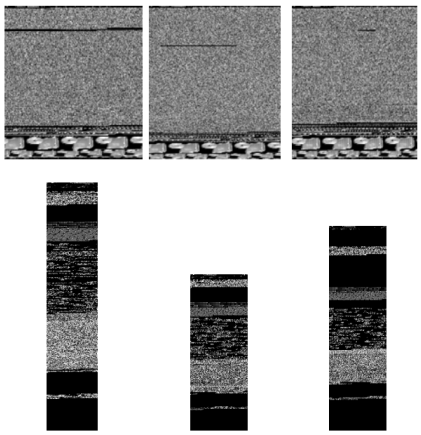
Jasno je da se ove značajke imaju važnost u klasificiranju *malware*-a jer, po istraživanju iz 2006. godine ([12]), 97 % *malware*-a su veličine ispod $1MB$ jer im to omogućava brže izvršavanje na ciljanoj lokaciji. Veličine njihovih binarnih reprezentacija, naravno onda variraju ovisno o familiji *malware*-a.

*3) **Entropija**:* Entropija je, po definiciji iz [3], termin koji označava mjeru neodređenosti informacija. Laički rečeno, u našem slučaju, računat ćemo mjeru količine nereda u distribuciji `byte`-ova u *.byte* datoteci. Stoga je, u *.byte* datotekama koje reprezentiraju *malware*, vrijednost entropije vrlo visoka u odnosnu na normalne datoteke.
Značajke koje ćemo izračunati na ovaj način koriste se za detekciju moguće prisutnosti skrivanja određenih poteza (eng. *obfuscation*). Kao takve, vrlo su bitne za klasifikaciju *malware*-a.
Kod računanja entropije prvi korak bio je podijeliti *.bytes* datoteku u N prozora. Dalje se, *sliding window* [11] metodom računaju entropije:

$$e_i = -\sum_{j=1}^{m} p(j)\ log_2 p(j)$$

gdje je $p(j)$ frekvencija `byte`-a j u prozoru i, a m broj različitih `byte`-a u prozoru.

*4) **Slikovna reprezentacija malware-a**:* U raznim istraživanjima [8] ispostavilo se da slikovne reprezentacije *malware*-a različitih familija, ukoliko svaki `byte` reprezentiramo kao količinu sive boje jednog piksela na slici, slijede određeni uzorak.
Pogledajmo na primjer fotografiju 2.



Slika 2. Primjer slikovne reprezentacije *malware*-a.

Gornje tri slike reprezentiraju *malware* iz familije *Fakerean*, dok donje tri *Dontovo.A*. Naravno, uočavamo uzorak.

*5) **Duljine stringova**:* Poznato nam je da se $ASCII$ znakovi u decimalnom rasponu $32 - 127$ mogu ispisati, odnosno da reprezentiraju neki znak. Na primjer, *2A* reprezentira znak `*`. Ovu transformaciju `byte`-ova iskoristili smo i za dobivanje novih značajki našeg problema.

B. Značajke izvučene iz .asm datoteka

*1) **Metapodaci**:* Pod ovim pojmom smatramo veličinu *.asm* datoteke i broj linija unutar nje. Ove značajke odabrane su iz istog razloga kao i značajke o metapodacima *.bytes* datoteka.

*2) **Simboli**:* Ova skupina značajki sastoji se od broja pojavljivanja sljedećih značajnih simbola `+ - * ] [ ? @` u asemblerskom kodu. Navedene simbole uzeli smo u obzir jer njihova velika frekvencija općenito znači da je simbol namjenjen za izbjegavanje otkrivanja od zlonamjernog napada.

*3) **Operation Code (OPC)**:* Ova grupa značajki odnosi se na broj pojavljivanja instrukcija koje specificiraju koje operacije procesor treba učiniti. Npr. instrukcija `add` govori procesoru da treba zbrojiti dvije vrijednosti navedene neposredno nakon te instrukcije. Izmjenama uobičajenih instrukcija može se izbjeći otkrivanje zlonamjernih *sofrware*-a, ali ne i njihova klasifikacija [10] zbog čega smo se odlučili za ove značajke.

*4) **Registri**:* Ova skupina značajki koja govori o frekvenciji korištenja registara koji nisu namjenjeni za zadatke. Iako je moguće izmjeniti imena registara, ovako dobivene značajke mogu biti korisne pri klasifikaciji *malware*-a u jednu od familija.

*5) **Sekcije**:* Mnoge verzije *malware*-a dobivene su pakiranjem, odnosno metodom izmjene izvršnih datoteka bez mijenjanja njihove izvorne funkcionalnosti. Svaka pakirana datoteka sastoji se od nekih predefiniranih dijelova, na primjer `.text`, `.data`, `.bss`, `.rdata`, `.edata`, `.idata`, `.rsrc`, `.tls`, `.reloc`. Na temelju tih dijelova, definirali smo još 24 značajke.

*6) **Data define (DD)**:* Neki *malware*-i uopće ne koriste ili koriste samo nekoliko API poziva i operacijskih instrukcija. Umjesto njih, koriste `db`, `dw` i `dd` instrukcije. Stoga ova skupina značajki služi za opisivanje razmjera korištenja istih u *malware*-ima.

*7) **Ključne riječi**:* Pod ovim pojmom smatramo skupinu od 95 značajki koje objašnjavaju frekvencije korištenja nekih ključnih riječi - npr. `hkey local machine` koja označava pristup putanji *Windows registry*-a (tekstna baza podataka koja sadrži manje ili više važne podatke/zapise vezane uz rad *Windows* operacijskog sustava).

*8) **API**:* Značajke koje predstavljuju frekvencije pozivanja svakog od 794 najkorištenijih API-ja u zlonamjernim datotekama.

Na ovaj način prikupili smo 1693 značajki za svaki *malware*.

III. Problem

Problem se sastoji od odabira značajki i izgradnje modela strojnog učenja koji će najbolje predviđati u koju klasu pripada promatrani *malware*.

Kao mjeru uspješnosti klasifikacije koristmo točnost i tzv. *logloss*. Točnost mjerimo kao udio točnih predviđanja, no to obično nije dovoljno za procjenu robusnosti predviđanja. Zato još koristimo i klasifikacijsku funkcija gubitka *logloss* često korištenu evaluacijsku mjeru na *Kaggle* natjecanjima. Računa se prema sljedećoj formuli

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} log(p_{ij}) \quad (1)$$

gdje je N broj datoteka u skupu za testiranje, M broj oznaka, `log` prirodni logaritam, a p_{ij} je predviđena vjerojatnost da zapažanje i pripada klasi j. Vrijednost y_{ij} je 1 ako je zapažanje i u klasi j, a 0 inače. Cilj je minimizirati vrijednost te funkcije. Radi jednostavnosti, rješenje ćemo učitati na *Kaggle* koji će nam kao povratnu informaciju dati mjeru uspješnosti.

IV. Pristup problemu

Naš pristup problemu bazira se na odabiru manjeg skupa značajki koji dovoljno dobro reprezentira sve podatke. Na taj način pojednostavljujemo model i uspostavljamo ravnotežu između kompleksnosti i performanse. Odabir značajki provodimo kroz nekoliko faza.

1) ***`NaN` vrijednosti*** *:* Prije samog reduciranja broja značajki, posebnu pozornost dajemo postojećim `NaN` vrijednostima. Ustanovili smo da se u slučaju *train* i *test* skupa podataka radi o značajkama iz skupine *section* i *data define*. Iz eksploratorne analize je jasno kako se sve ove značajke odnose na omjer broja određenih linija (*section*) ili omjer korištenja određenih instrukcija (*data define*) u *.asm* datoteci. `NaN` vrijednosti su stoga rezultat dijeljenja $\frac{0}{0}$. Informacija da ne postoji određenih linija ili instrukcija u nekoj sekciji *.asm* datoteke nam nije beznačajna, tako da na mjesta `NaN` vrijednosti postavljamo 0.

2) ***Izbacivanje značajki na temelju točnosti modela****:* Iz eksploratorne analize smo naslutili kako kategorije uvelike variraju po značajnosti. Zato nastavljamo analizu po kategorijama te kao prvi korak selekcije provodimo eliminaciju beznačajnih kategorija. Značajnost svake kategorije određujemo prema rezultatima jednostavnog *XGBoost* modela za klasifikaciju koji se trenira na značajkama isključivo te kategorije. Uspješnost modela određujemo pomoću različitih metrika, promatranjem konfuzijske matrice, usporedbom distribucija *malware*-a *train* i *test* skupa po klasama te konačno učitavanjem rezultata na *Kaggle*.

Ovako dobiveni modeli su provodili dobru klasifikaciju, osim oni stvoreni na temelju sljedećih značajki: *section*, *data define*, *api*, *keywords*, koje zbog toga eliminiramo.

Modeli trenirani na *section* i *data define* značajkama većinu *malware*-a krivo klasificiraju u *Kelihos ver3* klasu.

Modeli trenirani na *api* i *keywords* značajkama dobro klasificiraju *malware*-a iz *train* skupa, dok daju izrazito loše rezultate na *test* skupu.

Keywords značajke su usko povezane sa *section* i *data define* značajkama koje su se također pokazale beznačajnima.

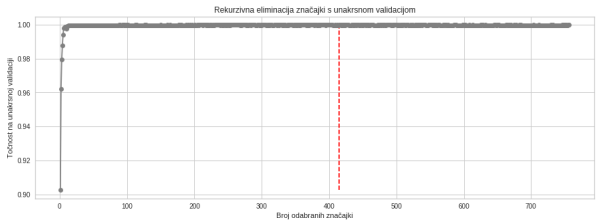
Promatrajući strukturu *api* značajki, primjećujemo da je velika većina vrijednosti zapravo jednaka 0, uz iznimke neovisne o klasama, pa kao takve ne nose neku informativnu vrijednost. Nadalje, svojstvo 'korištenje određenog api-ja' se lako može zaobići korištenjem srodnog api-ja ili internom implementacijom vlastitih metoda, te na taj način i nastaju varijante nekih *malwarea*. Iz toga zaključujemo da su ove značajke pogodnije za otkrivanje zlonamjernog softvera nego za njihovu klasifikaciju.

Ovako provedenom predselekcijom, broj značajki je prepolovljen.

3) ***Odabir značajki uz pomoć prediktivnog modela****:* Konačan skup značajki na kojima ćemo trenirati model odabiremo koristeći proceduru za rekurzivnu eliminaciju značajki uz klasifikator slučajnih šuma (implementiran u `sklearn.ensemble.RandomForestClassifier`). Uz navedeno, pomoću peterostruke unakrsne validacije saznajemo koliki je optimalan broj značajki u finalnoj selekciji (funkcija `sklearn.feature_selection.RFECV` implementirana u `sklearn.cross_validation.StratifiedKFold`). Budući da baratamo s nebalansiranim podacima u *train* skupu, funkcijom `StratifiedKFold()` osiguravamo očuvanje distribucija klasa u *trian* i *test* skupovima za svako vrednovanje određenog modela.
Za parametar koji opisuje sustav vrednovanja modela (`scoring`) testiranjem pronalazimo najbolji od dolje navedenih:

- `accuracy`
- `roc_auc_ovo`
- `f1_micro`.

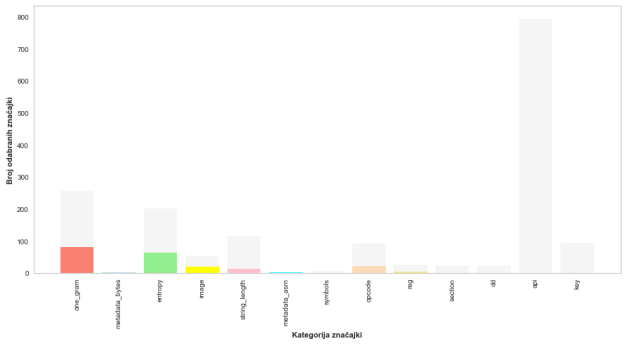
Najbolje rezultate smo dobili korištenjem značajki dobivenih uz `scoring='roc_auc_ovo'` (*area under curve*) koji vrednuje modele s obzirom na površinu ispod pripadne ROC krivulje.



Slika 3. Rekurzivna eliminacija značajki s unakrsnom validacijom

Kao optimalan broj značajki po RFEVC-u dobivamo 415 značajki. Međutim, iz grafa je vidljivo da je i za puno manji broj značajki točnost na unakrsnoj validaciji zanemarivo lošija, pa je moguće dodatno smanjiti broj značajki uz i dalje visoku točnost klasifikacije. Točan broj najboljih značajki stoga odabiremo jednostavnom *pohlepnom* metodom; trenirali smo model koristeći od 50 do 415 najboljih značajki s korakom 4,

te uspoređivali rezultate.
Napokon, najbolji rezultat smo dobili uz najvažnijih 208 **značajki.**



Slika 4. Konačno odabrane značajke

Gornji prikaz opisuje distribuciju konačno odabranih značajki po grupama. Grupe s najvećim internim omjerom izabranih i svih značajki su *entropy, image, opcode* te *one gram*. Upravo modeli trenirani na tim grupama značajki su u predselekciji davali najbolje rezultate.

V. Rješenje

Za izradu klasifikatora odlučili smo koristiti `XGBoost Classifier` iz biblioteke `xgboost` u `python`-u. Da bismo odredili *hiperparametre* modela koji daju najbolje rezultate našeg problema koristili smo `Randomized Search CV` iz biblioteke `sklearn`. Ta metoda na nasumičan način bira neke od zadanih *hiperparametara* s ciljem odabira najboljih. Željeli smo pretražiti prostor vrijednosti za sljedeće parametre:

- min_samples_split $([2, 5, 10])$
- min_samples_leaf $([1, 2, 4])$
- bootstrap $([True, False])$
- max_features $(['log2', 'auto'])$
- n_estimators (10 nasumičnih vrijednosti iz intervala $[70, 1000]$)

Naknadno, prostor najboljih *hiperparametara* dodatno smo istražili koristeći `Grid Search CV` iz biblioteke `sklearn`, koji, za razliku od `Randomized Search CV` , pretražuje svaku kombinaciju parametara zbog čega njegova vremenska složenost veća. Ovom metodom pretražili smo prostor najboljih *hiperparametara* dobivenih sa prethodno spomenutim `Randomized Search CV`.

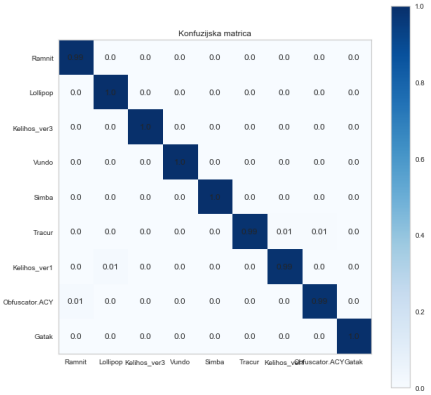
Najbolji model dobili smo za sljedeće parametre:

- min_samples_split 5
- min_samples_leaf 2
- bootstrap $True$
- max_features $'auto'$
- n_estimators 586

Kako se iz slike 1 jasno vidi da su podaci nebalansirani, pokušali smo poboljšati model dodatnim uzorkovanjem metodom *Smote* iz biblioteke `imblearn` no tim modelom nismo dobili značajnije rezultate.

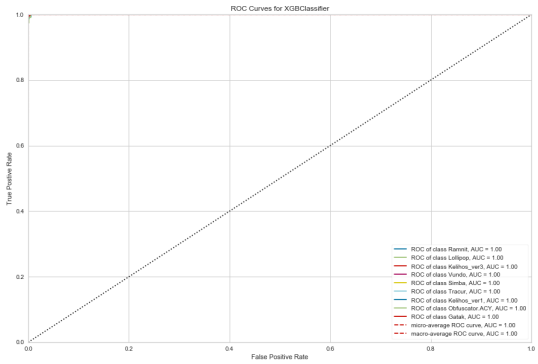
VI. Rezultati

Kako bi naš model testirali na podacima za koje nam je poznata klasa u koju pripadaju, u svrhu lokalnog računanje točnosti našega modela podijelili smo podatke za treniranje na *train* i *test* skup. Rezultati našega najboljeg modela mogu se vidjeti na slikama 5, 6, 7, 9, 8, 10.



Slika 5. Konfuzijska matrica

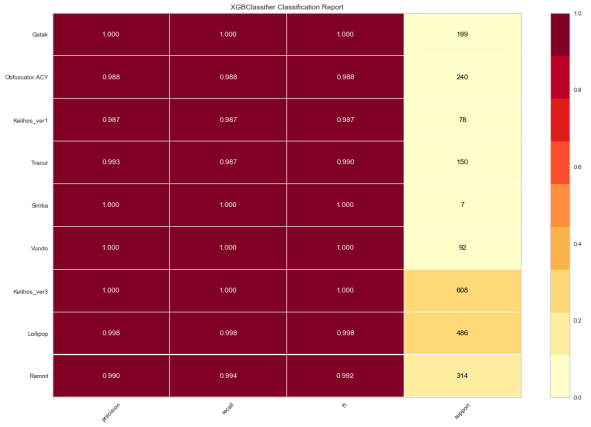
Iz konfuzijske matrice na slici 5 vidimo da naš model odlično grupira podatke u klase u koje pripadaju. Napomenimo da to vrijedi i za *Simba* klasu za koju imamo najmanje ulaznih podataka.



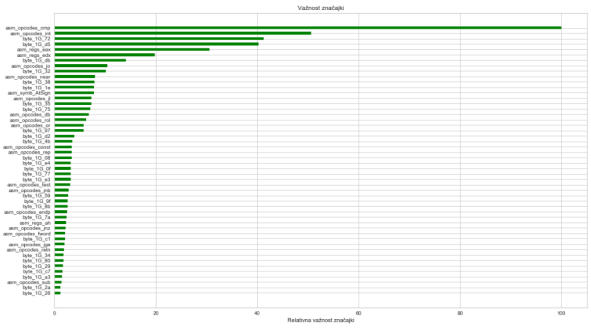
Slika 6. *ROC* krivulje

Na slici 6 možemo vidjeti uspješnost modela prikazanu preko ROC krivulje. Za sve klase, površina ispod te krivulje (AUC) je maksimalna, odnosno iznosi 1 , što je još jedan pokazatelj dobrog modela.

Slika 7 prikazuje rezultate mjera točnosti *precision*, *recall* i *f1* svake klase zasebno. Vidimo da smo, iako se radi o nebalansiranim podacima, za svaku od klasa uspjeli dobiti vrlo visoku točnost.

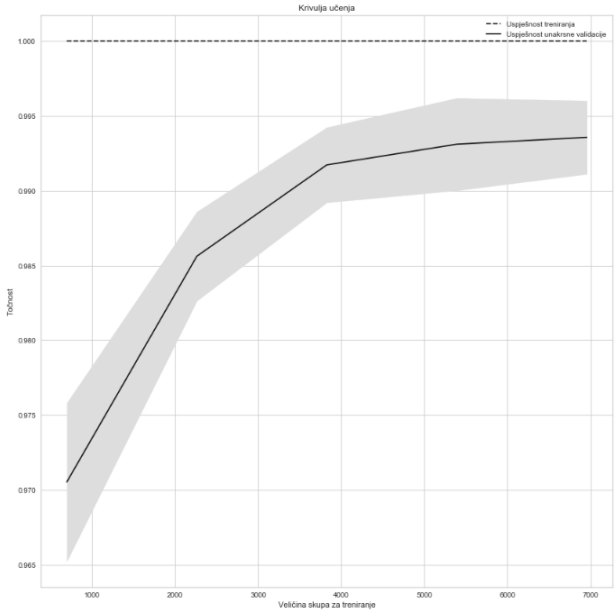


Slika 7. Mjere točnosti modela



Slika 8. Važnost značajki

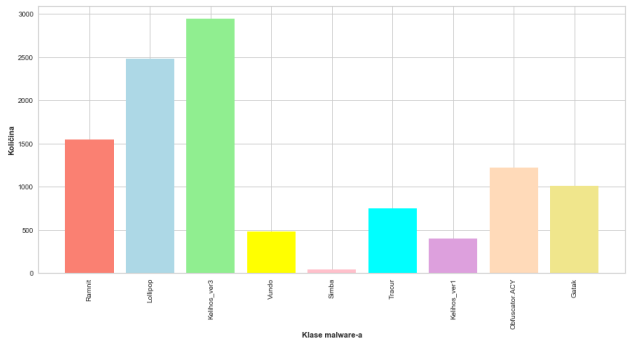
Najbitnijim značajkama pokazale su grupe značajki *OPC* i *unigram*, što možemo vidjeti na slici 8 koja prikazuje 50 najvažnijih značajki poredane po njihovoj značajnosti.



Slika 9. Krivulja učenja

Prije evaluacije našeg modela na *Kaggle*-u, mogli bismo nešto reći o našim rezultatima na 10873 *malware*-a za koje nam nije poznata klasa pomoću slike 10. Naime, ona pokazuje raspodjelu tih podataka po klasama. Ako se prisjetimo kako je izgledao isti graf na podacima za koje nam je bila poznata klasa *malware*-a (1) u koju pripadaju, mogli bismo, samo iz ta dva grafa, reći da smo dobro grupirali podatke jer je distribucija po klasama vrlo slična i u jednom i u drugom grafu.

Nakon što smo lokalno istrenirali model, *logloss* točnost klasifikacije na *Kaggle* test skupu iznosila je 0.01281**.**



Slika 10. Dobivena distribucija *test* skupa

VII. Osvrt na druge pristupe

Na službenoj *Kaggle*-ovoj stranici natječaja mogu se vidjeti pobjednički rezultati *logloss*-a [7]. Najbolji rezultat postigao je tim s *logloss*-om 0.00283 koristeći *XGBoost* algoritam pri izradi modela. Uz `XGBoost`, pri izradi modela bili su popularni i algoritmi `ExtraTreesClassifier` i `LightGBM`.

Naš rezultat sa *loglossom* = 0.01281 nalazi se u prvih 50 najboljih.

Literatura

[1] Mansour Ahmadi i dr. "Novel Feature Extraction, Selection and Fusion for Effective Malware Family Classification". Ožujak 2016. DOI: 10.1145/2857705.2857713.

[2] *Harlick texture features*. http://murphylab.web.cmu.edu/publications/boland/boland_node26.html.

[3] *Hrvatska enciklopedija, entropija*. https://www.enciklopedija.hr/natuknica.aspx?ID=18042.

[4] *Hrvatski mrežni riječnik*. http://ihjj.hr/mreznik/page/pojmovnik/6/.

[5] *Making Sense of Logarithmic Loss*. https://datawookie.netlify.app/blog/2015/12/making-sense-of-logarithmic-loss/.

[6] *Microsoft malware classification Challenge*. http://arxiv.org/abs/1802.10135.

[7] *Microsoft malware classification Kaggle Challenge*. https://www.kaggle.com/c/malware-classification.

[8] Lakshmanan Nataraj i dr. “Malware Images: Visualization and Automatic Classification”. (2011).

[9] Lakshmanan Nataraj i dr. “Malware Images: Visualization and Automatic Classification”. (Srpanj 2011). DOI: 10.1145/2016904.2016908.

[10] Igor Santos i dr. “Idea: Opcode-Sequence-Based Malware Detection”. 2010.

[11] *Sliding Window Method and Exponential Weighting Method*. https://www.mathworks.com/help/dsp/ug/sliding-window-method-and-exponential-weighting-method.html.

[12] *Understanding and Detecting Malware Threats Based on File Size*. https://d3gpjj9d20n0p3.cloudfront.net/fortiguard/research/DetectingMalwareThreats.pdf.

[13] *What Can N-Grams Learn for Malware Detection?* https://www.edwardraff.com/publications/what_can_ngrams_learn.pdf.