Korištenje algoritama strojnog učenja za potrebe kibernetičke sigurnosti

ProjektR

Tin Popović, Marko Kremer, Martina Galić, Ivan Hajpek

Ak. godina 2021./2022.

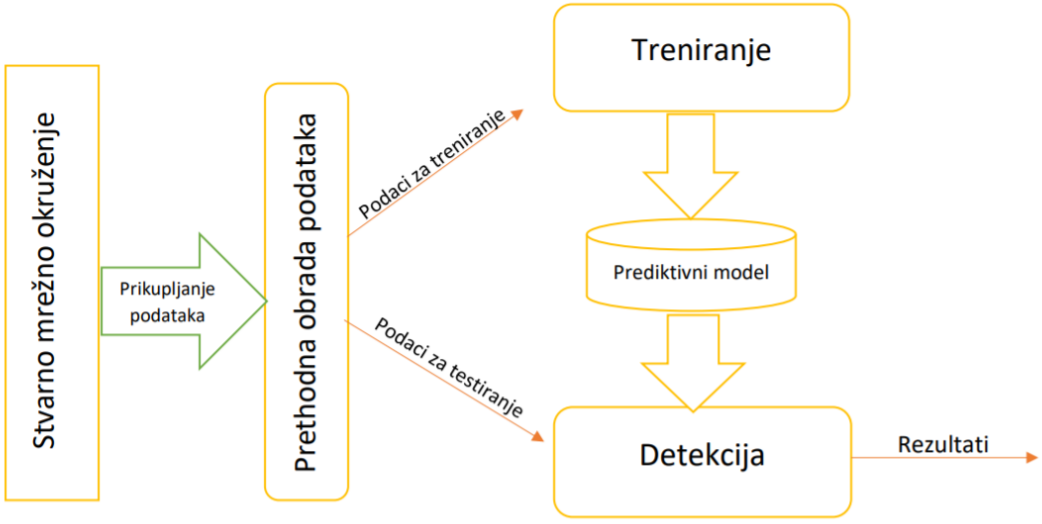
Mentor: Damir Pintar

Sadržaj

O projektu

S eksponencijalnim rastom veličine računalnih mreža i razvijenih aplikacija dolazi do povećanja broja napada i količine štete koju uzrokuju napadi. Sustavi za otkrivanje napada (IDS) i sustavi za sprječavanje napada (IPS) jedni su od najvažnijih obrambenih alata protiv sofisticiranih i stalno rastućih mrežnih napada.

Cilj ovog projekta je upoznati se sa strojnim učenjem i njegovom primjenom nad skupom podataka. Strojno učenje (engl. Machine Learning) je jedan od pristupa za rješavanje problema kod kojih je cilj donositi opće zaključke i predviđati na osnovu dobivenih (često mnoštva) podataka. Nad već unaprijed pripremljenim skupom podataka razvit ćemo prediktivni model. Projekt smo implementirali koristeći programski jezik R, te radno okruženje R-studio. Rad je usko povezan sa statistikom, znanošću o podacima (engl. Data Science) te drugim srodnim područjima. Cilj je dobiti (matematičke) modele na osnovu kojih se kasnije provode zaključivanja i predviđanja. Algoritmi za strojno učenje imaju matematičke modele u pozadini te je svrha većine algoritama na temelju podataka odrediti parametre modela kako bismo mogli predviđati i zaključivati s određenom razinom preciznosti.



Strojno učenje ćemo provesti u nastavku upravo na način prikazan na slici. Pripremljeni skup podataka ćemo analizirati te ga nakon toga podijeliti na dva dijela: * skup za treniranje * skup za testiranje. Radom nad skupom za treniranje izgradit ćemo model čiju ćemo preciznost provjeriti skupom za testiranje.

Uvod

Kanadski institut za kibernetičku sigurnost(CIC) objavio je CICIDS2017 dataset. Navedeni podatkovni skup uključuje rezultate analize mrežnog prometa koristeći CICIFlowMeter(algoritam razvijen od strane istog instituta). Podaci su prikupljeni iz stvarnog svijeta te sadrže niz BENIGN-a i napada. Skup prikazuje apstraktno ponašanje ljudskih interakcija i generirani naturalistički benigni pozadinski promet. Konkretno, prikazano je apstraktno ponašanje 25 korisnika na temelju HTTP, HTTPS, FTP, SSH i protokola e-pošte.

Razdoblje prikupljanja podataka započelo je u 9 sati, ponedjeljak, 3. srpnja 2017. i završilo je u petak 07.07.2017 u 17 sati. Ponedjeljak je jedini dan koji uključuje samo benigni promet, dok u ostalim danima su prisutni i kibernetički napadi. CICIDS2017 uključuje sljedeće napade: DoS, DDoS, Heartbleed, Web Attack, Infiltration, Botnet, i Brute Force napade.

Kroz par koraka upoznat ćemo se pobliže s podatkovnim skupom, te na kraju razviti prediktivni model na temelju algoritama strojnog učenja.

U svrhu učenja i istraživanja, CICI je objavio MachineLearningCSV.zip, što predstavlja preuređeni data-set koji je spreman za implementaciju algoritama strojnog učenja. ("https://www.unb.ca/cic/datasets/ids-2017.html")

Podatkovni skup koji koristimo sadrži sljedeće vrste napada:

Napadi temeljeni na volumenu prometa.

Ovi napadi generiraju golemu količinu mrežnog prometa prema ciljnom uređaju kako bi zasitili resurse uređaja koji je napadnut. Neki od uobičajenih DDoS napada su User Datagram Protocol (UDP) Flood, Internet Control Message Protokol (ICMP) Flood, Synchronize (SYN) Flood.

Napadi temeljeni na protokolu.

Ovi napadi koriste ranjivost prisutnu kod specifičnih protokola poput TCP-a kako bi onemogućili korisnicima usluge servera.

Napadi na aplikacijskom sloju.

Ovi napadi ciljaju na određeni protokol aplikacijskog sloja poput HTTP-a i generiraju pakete protokola na način na koji će poslužitelj imati previše dugo otvorenih sesija što uzrokuje iscrpljivanje resursa. Paketi izgledaju vrlo slično normalnom paketu i ne otkrivaju ih uobičajeni algoritmi za otkrivanje anomalija. Poznata nam je informacija da je najčešći tip napada SYN flood koji čini 83.2% DDoS napada, dok ga slijede UDP Flood i HTTP napadi.

Spore napade s niskom stopom opasnosti, mnogo je teže otkriti jer su vrlo slični normalnom prometu. Napadač otvara sesiju s mrežnim uređajem i drži ju otvorenom dulje vrijeme, šaljući vrlo malo prometa. Na ovaj način mrežni uređaj neće zaustaviti idle timeout

napadača te će se činiti da je promet bezopasan i normalan jer ga tradicionalne tehnike otkrivanja DDoS-a neće razotkriti.

Za uspješno implementiranje prediktivnih modela, koristit ćemo sljedeće znanstvene radove:

1."Network Traffic Behavioral Analytics for Detection of DDoS Attacks"[1]

2."Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization"[2]

[1] https://scholar.smu.edu/cgi/viewcontent.cgi?article=1074&context=datasciencereview

[2] https://www.scitepress.org/papers/2018/66398/66398.pdf

Podatkovni skup - CICIDS2017

Za početak moramo učitati sve potrebne datoteke. CIC je objavio ukupno 8 datoteka mrežnog prometa i to su Monday, Tuesday, Wednesday, Thursday_Morning, Thursday_Afternoon, Friday_Morning, Friday_Afternoon_PortScan i Friday_Afternoon_DDos. Pošto trebamo razviti model predikcije na bazi cijelog tjedna, prvo ćemo spojiti sve datoteke kako bi dobili objekt "FullWeek".

Stvorili smo objekt FullWeek. Upoznajmo se s dimenzijama našeg objekta.

```
cat("Broj zapisa u objektu FullWeek je : ",nrow(FullWeek), '\n')

## Broj zapisa u objektu FullWeek je :  2830743

cat('Svaki zapis opisuje ',ncol(FullWeek) ,' varijabli.', ' ')

## Svaki zapis opisuje  79  varijabli.
```

Vidimo da raspolažemo s ogromnom količinom podataka, i detaljnim opisom tih podataka. Prije nego što se upoznamo s varijablama ovog podatkovnog skupa, moramo ispitati postojanje NA vrijednosti. Radimo s izrazito velikom količinom podataka, te u takvim slučajevima mogu se pojaviti NA vrijednosti. Ako one postoje moramo s njima pažljivo rukovati, jer mogu dovesti do pogrešnih zaključaka. Postoje par načina rukovanja s NA vrijednostima: - Zamjena s aritmetičkom sredinom - Zamjena s medijanom - Brisanje tih podataka, i slično

Prvo, ispitajmo pojavu NA vrijednosti.

```
for (col_name in names(FullWeek)){
  if (sum(is.na(FullWeek[,col_name])) > 0){
    cat('Ukupno nedostajućih vrijednosti za varijablu ',
        col_name, ': ', sum(is.na(FullWeek[,col_name])),'  ')
  }
}

## Ukupno nedostajućih vrijednosti za varijablu  Flow.Bytes.s :  1358
```

NA vrijednosti su prisutne u našem datasetu. Imamo ih 1358, za varijablu Flow.Bytes.s. NA vrijednosti čine svega 0.0005% našeg skupa, stoga ih možemo ukloniti bez brige o gubitku važne količine informacija. Brisanje NA vrijednosti u programskom jeziku R, implementirano je funkcijom na.omit()

```
FullWeek <- na.omit(FullWeek)
```

NA vrijednosti su izbačene, te prihvaćamo činjenicu da je podatkovni skup ispravan i spreman za implementiranje algoritama strojnog učenja. Broj varijabli (79) čini ovaj podatkovni skup nepreglednim, ali barem možemo reći da su svi zapisi detaljno opisani. Neke varijable ćemo kratko predstaviti:

Varijabla	Opis
Flow Duration	Ukupno trajanje protoka
Total Length of Fwd Packets	Ukupan broj paketa koji je poslan
Total Length of Bwd Packets	Ukupan broj paketa koji je primljen natrag
Flow Bytes/s	Broj bajtova po sekundi u protoku
Flow Packets/s	Broj paketa po sekundi u protoku
Average Packet Size	Srednja vrijednost veličine paketa
Label	Indikator koji opisuje vrstu mrežnog prometa

Putem zadnje varijable na slici("LABEL") ćemo klasificirati vrste prometa koje imamo. Pogledajmo s kojim vrijednostima naša varijabla LABEL raspolaže:

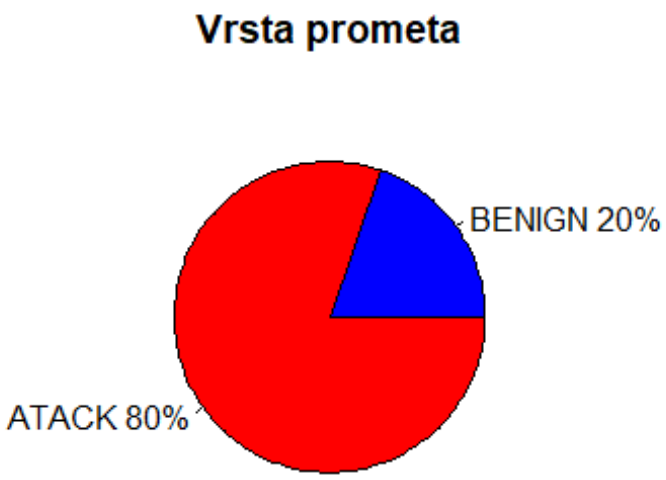
```
data.frame(FullWeek %>% count(Label))
```

```
##                            Label       n
## 1                         BENIGN 2272688
## 2                            Bot    1966
## 3                           DDoS  128027
## 4                  DoS GoldenEye   10293
## 5                       DoS Hulk  230124
## 6                DoS Slowhttptest    5499
## 7                   DoS slowloris    5796
## 8                     FTP-Patator    7938
## 9                      Heartbleed      11
## 10                   Infiltration      36
## 11                       PortScan  158930
## 12                    SSH-Patator    5897
## 13   Web Attack ďż\275 Brute Force    1507
## 14 Web Attack ďż\275 Sql Injection      21
## 15           Web Attack ďż\275 XSS     652
```

Varijabla LABEL ima svega 15 vrijednosti. Od toga jedna predstavlja benign promet, dok preostalih 14 predstavljaju vrstu napada. Za početak, sve napade ćemo svrstati pod jednu varijablu i zatim pogledati omjer napada I benign prometa.

```
FullWeek_original = FullWeek
FullWeek$Label[FullWeek$Label != "BENIGN"] ="Attack"
data.frame(FullWeek %>% count (Label))
```

```
##    Label       n
## 1 Attack  556697
## 2 BENIGN 2272688
```



Vidimo da 80% našeg podatkovnog skupa čini BENIGN, dok svega 20% su napadi. Očigledno je da imamo neuravnoteženi skup podataka. Neuravnotežene klasifikacije predstavljaju izazov za prediktivno modeliranje jer je većina algoritama strojnog učenja osmišljena uz pretpostavku da će svaka klasa imati jednak broj primjeraka. Rad s neuravnoteženim dataset-om rezultira modelima koji imaju loše prediktivne performanse, posebno za manjinsku klasu. To je problem jer je obično manjinska klasa važnija(kao i u našem slučaju) i stoga je problem osjetljiviji na pogreške u klasifikaciji za manjinsku klasu nego za većinsku klasu.

Trenutni dataset predstavlja veliku opasnost pri razvijanju prediktivnog modela. Zamislimo scenariji da smo razvili prediktivni model koji za sav mrežni promet kaže da je benign. Takav model bi u našem slučaju imao 80% točnosti, što nema smisla. Zbog toga moramo razviti uravnoteženi model dataset-a. Uzimamo standardni i sigurni omjer 50:50. Postoji više načina kako stvoriti uravnoteženi model, no razmotrit ćemo dvije najpogodnije za naš slučaj : Oversampling = duplicira zapise manje klase , i Undersampling = brisanje zapisa više klase.

Obe metode imaju svoje prednosti i nedostatke, ali prvenstveno moramo uzeti u obzir veličinu našeg skupa. Na početku smo pokazali kako imamo skoro 3 milijuna zapisa, stoga je bolje primijeniti metodu undersampling-a.

```
Attack <- which(FullWeek$Label != "BENIGN")
Ben <- which(FullWeek$Label == "BENIGN")

Ben.downSample <- sample(Ben, length(Attack))
FullWeekDS <- FullWeek[c(Ben.downSample,Attack),]
```

```
FullWeek = FullWeekDS
data.frame(FullWeek %>% count (Label))

##    Label      n
## 1 Attack 556697
## 2 BENIGN 556697
```

Koristeći metodu undersampling-a, stvorili smo uravnoteženi dataset, i smanjili broj zapisa. Imamo jednak omjer benign i attack prometa. Sada bi rad nad ovakvim podatkovnim skupom trebao davati ispravne modele.

Vrlo čest problem pri obučavanju modela je overfitting. Ovaj fenomen se događa kada model radi jako dobro na podacima koje smo koristili za obuku, ali ne uspijeva se dobro generalizirati na nove podatke. Neki od razloga zašto se to može dogoditi su: zbog buke u podacima ili to da je model naučio predvidjeti specifične ulazne podatke, a ne prediktivne parametre koji bi mu mogli pomoći da napravi ispravna predviđanja. Obično su složeniji modeli overfitted. Problem suprotan prethodnom bi bio underfitting.

Da nam se ne bi dogodili navedeni problemi, podijelit ćemo dataset na skup za treniranje i testiranje. Trenirat ćemo model koristeći skup za treniranje, a zatim ćemo primijeniti model na testni skup. Tako možemo ocijeniti performanse našeg modela. Na primjer, ako je točnost dobivena korištenjem trening skupa iznimno visoka, dok je točnost testiranja loša, onda je to dobar pokazatelj da je model vjerojatno overfitted.

Iako je jednostavan za korištenje i tumačenje, postoje slučajevi kada se postupak ne bi trebao koristiti, kao što je kada imamo mali skup podataka ili kada skup podataka nije uravnotežen. No pošto imamo velik skup podataka i prethodno smo ga uravnotežili, možemo koristiti navedeni postupak podjele dataset-a. 70% gore implementiranog dataseta iskoristit ćemo za treniranje modela, dok će ostatak za testiranje.

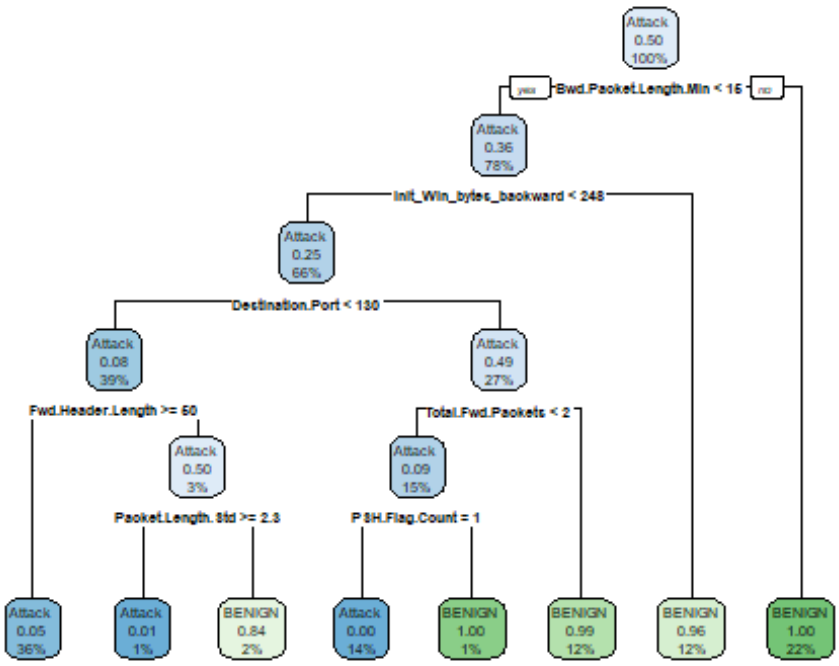
```
DT = sample(nrow(FullWeek), nrow(FullWeek)*.7)

Train = FullWeek[DT,]
Test = FullWeek[-DT,]
```

Sve je spremno za implementiranje algoritama strojnog učenja. Odlučili smo se za implementiranje Decision Tree-a i Naive Bayes algoritma.

Decision Tree

Započnimo s našim prvim algoritmom - "Decision Tree". Decision Tree je algoritam koji može provoditi klasifikacije i regresijske zadatke. Odlučili smo se za ovaj algoritam jer je interpretabilan. Lako ga je protumačiti. Za implementaciju algoritama strojnog učenja potrebno je samo specificirati varijablu za koju želimo razviti željeni prediktivni model. U programskom jeziku R, Decision Tree je implementiran u sklopu paketa : "rpart" ,i "rpart:plot"

```
decisionTreeModel = rpart::rpart(Label~., data = Train)
rpart.plot::rpart.plot(decisionTreeModel)
```



Imamo prediktivni model razvijen putem Decision Tree-a. Uočimo sada kako ovo stablo na prethodnoj slici radi predviđanja. U korijenu stabla provjeravamo prvo je li minimalna veličina paketa manja od 15. Ako nije, pomičemo se na desno dijete korijena gdje možemo zaključiti i klasificirati taj tok podataka kao benigan. U suprotnome slučaju, idemo na lijevo dijete gdje uspoređujemo broj bajtova inicijalnog prozora pri povratku s brojem 248. Ako je on veći, radi se o benignom toku podataka, dok u suprotnome slučaju moram ići na lijevo dijete kako bi provjerili idući uvjet. Slijedno, možemo zaključiti kako će izgledati daljnji tijek provjera, odnosno predikcije. Vidimo da su najvažnije varijable za razvoj prediktivnog modela: Bwd.Packet:Length.Min (minimalna veličina paketa pri povratku), init_Win_bytes_backward(svi bajtovi poslani u inicijalnom prozoru pri povratku) te Destination.Port

Znanstvenim radovima koje smo naveli na početku, sadrže tablicu "važnosti" pojedinih varijabla ,u kojoj se također nalaze varijable naših čvorova. Pa možemo zaključiti da je model ispravno izdvojio ključne varijable pri predikciji.

Testiranje dobivenog modela za rezultat daje tablicu sljedećeg oblika:

	Predicted class		
Actual Class		Class = Yes	Class = No
	Class = Yes	True Positive	False Negative
	Class = No	False Positive	True Negative

Ispravne predikcije su na tablici označene zelenom bojom, a neispravne crvenom bojom.

Rezultati testiranja našeg prediktivnog modela su:

```
Predictions = predict(decisionTreeModel, Test, type = 'class')

table = table(Test$Label, Predictions)
table

##         Predictions
##          Attack BENIGN
##   Attack 164150   2714
##   BENIGN   5913 161242
```

Za evaluaciju točnosti modela koristimo sljedeće metode:

1. Accuracy . Pokazuje omjer točno predviđenih opservacija u odnosu na ukupne opservacije. Najintuitivnija je metoda, ali može se koristiti samo ako imamo simetričan dataset(jednak omjer klasa, kao što mi imamo)

```
accuracy = sum(diag(table)) / sum(table)
print(paste("Accuracy našeg modela: ",round(accuracy,3)))

## [1] "Accuracy našeg modela:  0.974"
```

2. Precision. Ova metoda pokazuje omjer ispravno pozitivnih predikcija(True Positive) i ukupnog broja opservacija te klase(TP+FP). U našem kontekstu odgovara na pitanje : "Koliki je omjer ispravno detektiranih napada i ukupno detektiranih napada?"

```
precision = table[1] / (table[1]+table[2])
print(paste("Precision našeg modela: ",round(precision,3)))

## [1] "Precision našeg modela:  0.965"
```

3. Recall Predstavlja osjetljivost modela. Odgovara na pitanje:tp/tp+fn "Koji je omjer ispravno detektiranih napada i zbroja ispravnih i neispravnih detekcija napada(TP+FN)

```
recall = table[1] / (table[1]+ table[3])
print(paste("Recall of the model: ",round(recall,3)))
```

```
## [1] "Recall of the model:  0.984"
```

4. F1 score Predstavlja težinski prosjek precision i recall varijabli

```
f1_score = 2* (recall*precision) / (recall+precision)
print(paste("F1 score of the model: ",round(f1_score,3)))
```

```
## [1] "F1 score of the model:  0.974"
```

Točnost našeg Decision Tree modela je zadovoljavajuća, te možemo smatrati naš model ispravnim.

Naive Bayes

Nakon Decision Tree-a, razvit ćemo novi prediktivni model. Sada ćemo iskoristiti drugi algoritam strojnog učenja - Naive Bayes. Naive Bayes je algoritam temeljen na Bayesovom teoremu, zasnovan na pretpostavci nezavisnosti između prediktora(varijabli). Također druga značajka ovog algoritma je da svim varijablama daje jednaku važnost pri stvaranju predikcije, što nije primjenjivo u problemima stvarnog svijeta. Ta osnova imat će velike utjecaje na rezultate točnosti Naive Bayes modela. Ovaj model je jednostavno implementirati i koristan je za velike podatkovne skupove.Poznato je da je Naive Bayes jedna od najbržih metoda klasifikacije.
Naive Bayes računa vjerojatnost događaja u 4 koraka:

1. Računa vjerojatnosti 'a priori' za dani stupac, u ovom slučaju Label.
2. Računa uvjetnu vjerojatnost za svaki atribut za svaki tip Label-a.
3. Te vjerojatnosti stavlja u Bayesovu formulu i računa vjerojatnost 'a posteriori'.
4. Gleda koja klasa veću vjerojatnost te tako razvrstava promet.

Naziva se naivnim jer za sve varijable gleda kao da su nezavisne jedna od druge, iako u stvarnosti ne moraju biti.

$$P(Label|x) = \frac{P(x|Label) * P(Label)}{P(x)}$$

Gdje su "x" svi stupci u datasetu osim stupca Label.

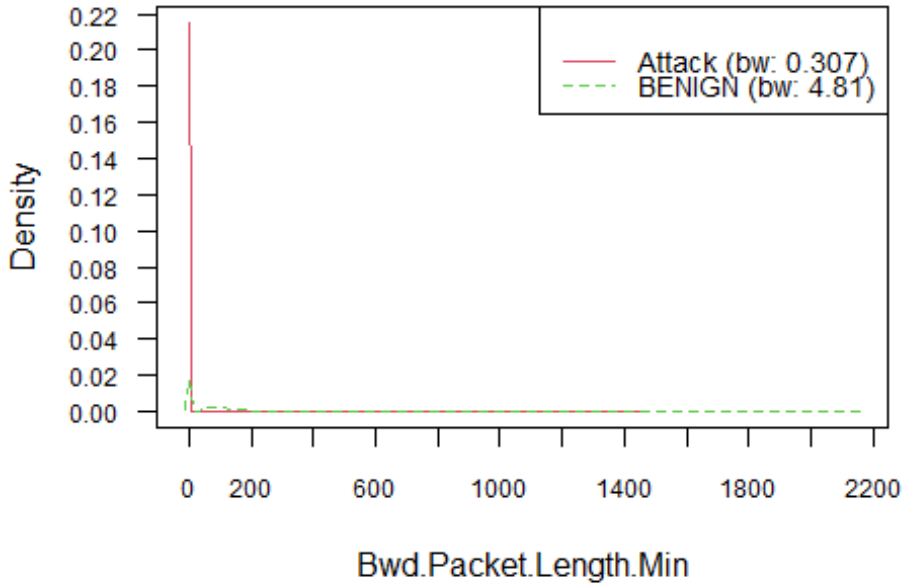
- $P(Label|x)$ = vjerojatnost pojave određenog $Label$-a(benign ili attack) za vrijednost varijable x
- $P(x|Label)$ = vjerojatnost pojave određene varijable x za zadani $Label$
- $P(Label)$ = vjerojatnost pojave određenog $Label$-a(benign ili attack)
- $P(x)$ = vjerojatnost pojave vrijednosti varijable x

Naive Bayes u programskom jezik R, se implementira pomoću paketa "naivebayes". Iskoristit ćemo iste skupove za testiranje i treniranje kao kod Decision Tree-a. Započnimo s treniranjem našeg skupa, te u svrhu boljeg razumijevanja modela prikazat ćemo vizualno rezultate treniranja modela.

```
NaiveBayesModel = naive_bayes(Label~., data = Train, usekernel = T)
```

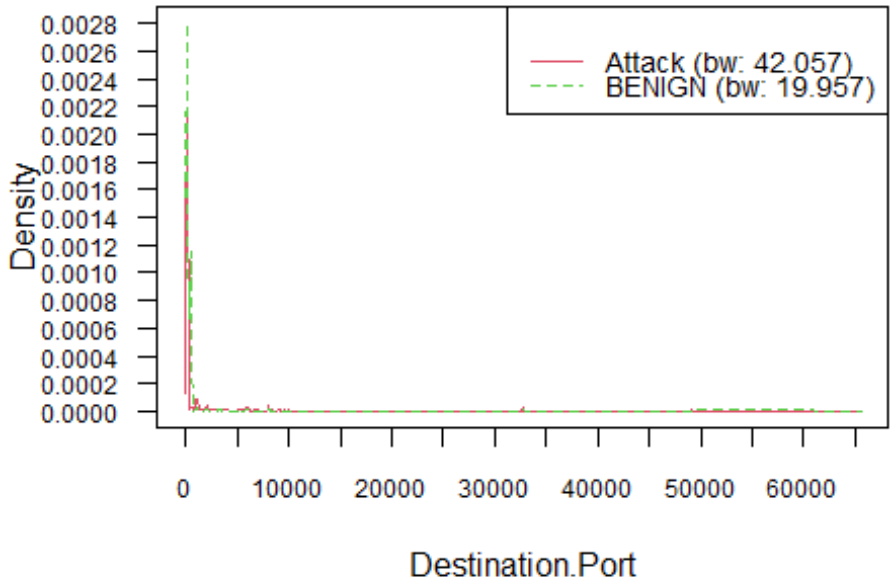
Nakon treniranja modela, pomoću plot() funkcije pogledajmo kako su vjerojatnosti raspoređene za prediktore(varijable). Izdvojit ćemo samo par varijabli.

```
plot(NaiveBayesModel, which = c("Bwd.Packet.Length.Min") ,legend = TRUE, lege
nd.box = TRUE)
```



Grafovi na prvu izgledaju nepregledno, ali važno je istaknuti uočene zaključke treniranja ovog modela. Za male vrijednosti varijable Bwd.Packet.Length.Min dominiraju napadi. Ako se prisjetimo, isti zaključak je bio u prvom čvoru našeg Decision Tree modela. Prikažimo sličan graf za prediktor Destination.Port.

```
plot(NaiveBayesModel, which = c("Destination.Port") ,legend = TRUE, legend.bo
x = TRUE)
```



Graf je opet nepregledan, ali izdvajamo sljedeće :

Za niže vrijednosti varijable "Destination.Port" prevladava benign promet.

Nakon treniranja, moramo testirati naš model, te ispitati točnost modela na isti način kao što smo to učinili s Decision Tree-om.

```
predictionsBayes = predict(NaiveBayesModel, Test, type = 'class')
tableBayes = table(Test$Label, predictionsBayes)
tableBayes
```

```
##         predictionsBayes
##          Attack BENIGN
##   Attack  54954 111910
##   BENIGN     10 167145
```

Dobili smo tablicu koja predstavlja rezultate treniranja našeg modela.Format tablice smo opisali prije. Ispitajmo točnost našeg modela:

```
accuracyBayes = sum(diag(tableBayes)) / sum(tableBayes)
print(paste("Accuracy našeg modela : ", round(accuracyBayes,3)))
```

```
## [1] "Accuracy našeg modela :  0.665"
```

```
precisionB = tableBayes[1] / (tableBayes[1]+tableBayes[2])
print(paste("Precision našeg modela: ",round(precisionB,3)))
```

```
## [1] "Precision našeg modela:  1"
```

```
recallB = tableBayes[1] / (tableBayes[1]+ tableBayes[3])
print(paste("Recall našeg modela: ",round(recallB,3)))

## [1] "Recall našeg modela:  0.329"

f1_scoreB = 2 * (recallB * precisionB) / (recallB+precisionB)
print(paste("F1 score našeg modela: ",round(f1_scoreB,3)))

## [1] "F1 score našeg modela:  0.495"
```

Uočavamo znatno manju točnost u ovom modelu, u odnosu na Decision Tree.

Decision Tree vs Naive Bayes

Zaključujemo da je Decision Tree bolji model jer daje znatno bolju pouzdanost. Međutim, nužno je izdvojiti veliku prednost Naive Bayes algoritma, a to je brzina. Decision Tree zahtijeva od 3-5 minuta za implementiranje, dok Naive Bayes se implementira za manje od 1 minute. No ispravnost Decision Tree-a je nezanemarivo veća u odnosu na Naive Bayes, te njega prihvaćamo kao bolju opciju.

Detekcija “Denial-of-Service” napada.

Uspješno smo razvili modele koji razvijaju predikciju pojave napada ili BENIGN prometa, i zaključili smo da je Decision Tree pouzdaniji. Sada ćemo otići korak dalje i razviti predikciju pojave pojedenih napada. Prisjetimo se koje smo napade imali na početku.

```
data.frame(FullWeek_original %>% count (Label))
```

```
##                            Label       n
## 1                         BENIGN 2272688
## 2                            Bot    1966
## 3                           DDoS  128027
## 4                  DoS GoldenEye   10293
## 5                       DoS Hulk  230124
## 6                DoS Slowhttptest    5499
## 7                   DoS slowloris    5796
## 8                     FTP-Patator    7938
## 9                      Heartbleed      11
## 10                   Infiltration      36
## 11                       PortScan  158930
## 12                    SSH-Patator    5897
## 13   Web Attack ďż\275 Brute Force    1507
## 14 Web Attack ďż\275 Sql Injection      21
## 15           Web Attack ďż\275 XSS     652
```

Vidimo prisutnost 14 napada. Najviše je DoS napada(DDoS i DoS). Oni predstavljaju “Denial-of-Service” napade. Ovi napadi rade tako što preopterete server s brojnim velikim paketima, i tako blokiraju pristup stvarnim korisnicima tog servera. Razlika između DoS i DDoS napad je broj napadača. DDoS je jači napad jer server je pod većom količinom napada.

Kratko smo se upoznali s prirodom “Denial-of-Service” napada, sada ćemo razviti prediktivan model koji detektira njihovu pojavu.

Započetak moramo pripremiti skup podataka. Mrežni promet ćemo podijeliti u 2 kategorije : DoS, i NOT_DoS(ostali napadi).

```
GroupAttacks = FullWeek_original
GroupAttacks$Label[GroupAttacks$Label == "DoS GoldenEye"] = "DoS"
GroupAttacks$Label[GroupAttacks$Label == "DoS Hulk"] = "DoS"
GroupAttacks$Label[GroupAttacks$Label == "DoS Slowhttptest"] = "DoS"
GroupAttacks$Label[GroupAttacks$Label == "DDoS"] = "DoS"
GroupAttacks$Label[GroupAttacks$Label == "DoS slowloris"] = "DoS"

GroupAttacks$Label[GroupAttacks$Label != "DoS"] = "NOT_DoS"


data.frame(GroupAttacks %>% count(Label))
```

```
##     Label       n
## 1     DoS  379739
## 2 NOT_DoS 2449646
```

Opet imamo neuravnotežen skup, kojeg ćemo opet riješiti metodom undersampling-a.

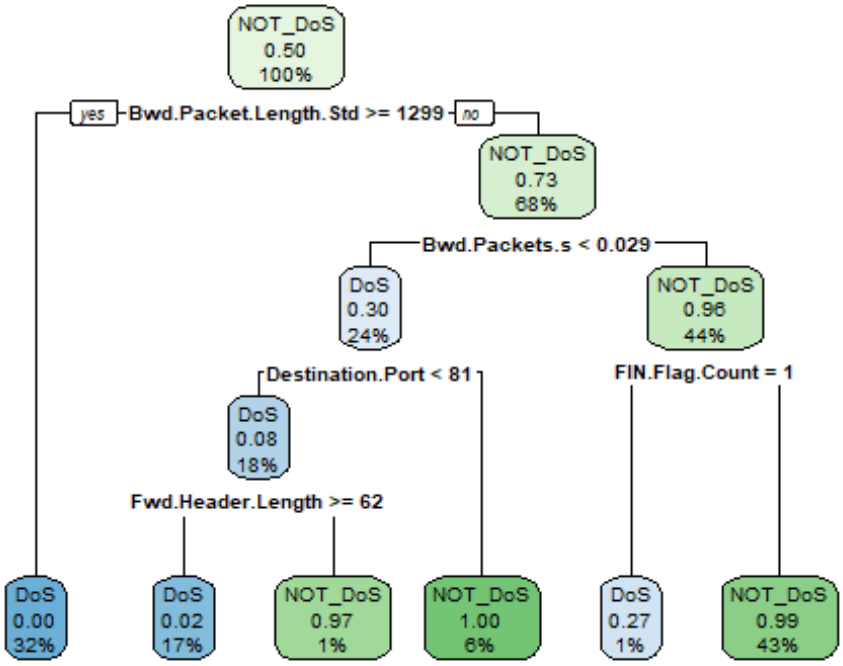
```
DoS <- which(GroupAttacks$Label == "DoS")
Not_DoS <- which(GroupAttacks$Label == "NOT_DoS")

NotDos.downSample <- sample(Not_DoS, length(DoS))
FullWeekDS <- GroupAttacks[c(NotDos.downSample,DoS),]

data.frame(FullWeekDS %>% count (Label))

##     Label      n
## 1     DoS 379739
## 2 NOT_DoS 379739
```

Imamo jednak broj elemenata u svakoj kategoriji. Pripremimo skup za testiranje i treniranje, prema istim omjerima kao i za prijašnje modele

```
DT_G = sample(nrow(FullWeekDS), nrow(FullWeekDS)*.7)

Train_group = FullWeekDS[DT_G,]
Test_group = FullWeekDS[-DT_G,]
```

Sve je spremno za Decision Tree, implementiramo ga na isti način kao i u prvom primjeru.

```
decisionTreeModel_group = rpart::rpart(Label~., data = Train_group)
rpart.plot::rpart.plot(decisionTreeModel_group)
```



Dobili smo novi prediktivni model, koji daje predikciju DoS napada. Ključna varijabla za DoS napade je "Bwd.Packet.Length.Std", koja predstavlja standardnu veličinu paketa u povratnom smjeru. Ako je ona iznad 1299 radi se o DoS napadu.Protivnom ,gledamo niz drugih varijabli i njihove vrijednosti i na temelju toga zaključujemo o kojoj vrsti prometa se radi. Još jedna kombinacija koja okarakterizira pojavu DoS napada je : Bwd.Packet.Length.Std < 1299 + Bwd.Packets.s < 0.029 + Destination.Port< 81 + Fwd.Header:length >= 62.

Testirajmo ispravnost našeg model:

```
Predictions2 = predict(decisionTreeModel_group, Test_group, type = 'class')

table = table(Test_group$Label, Predictions2)
table

##          Predictions2
##              DoS NOT_DoS
##   DoS     112691    1247
##   NOT_DoS   1968  111938

accuracy = sum(diag(table)) / sum(table)
print(paste("Accuracy našeg modela: ",round(accuracy,3)))

## [1] "Accuracy našeg modela:  0.986"

precision = table[1] / (table[1]+table[2])
print(paste("Precision našeg modela: ",round(precision,3)))
```

```
## [1] "Precision našeg modela:  0.983"
```

```
recall = table[1] / (table[1]+ table[3])
print(paste("Recall modela: ",round(recall,3)))
```

```
## [1] "Recall modela:  0.989"
```

```
f1_score = 2* (recall*precision) / (recall+precision)
print(paste("F1 score našeg modela: ",round(f1_score,3)))
```

```
## [1] "F1 score našeg modela:  0.986"
```

Razvili smo još jedan dobro pouzdan model koji uspješno detektira pojavu DoS napada.

Zaključak

U ovom projektu upoznali smo se s osnovama strojnog učenja, te koliko su napadi prisutni u mrežnom prometu. Objasnili smo koji su najvažniji problemi koji mogu biti prisutni prilikom implementacije strojnog učenja. Demonstrirali smo : rukovanje s velikom količinom podataka, izbacivanje NA vrijednosti, uravnoteživanje podataka, podjelu dataseta,i na kraju samu implementaciju algoritama za strojno učenje. Pored strojnog učenja, upoznali smo se s kibernetičkom sigurnosti. Pokazali smo koji su uvjeti dovoljni za detekciju napada. U budućnosti nadamo se pobliže upoznati s kibernetičkom sigurnosti, kako bi mogli donositi bolje zaključke i stvarati ispravnije prediktivne modele.