Primena neuronskih mreža na Network Intrusion Data

Seminarski rad u okviru kursa
Računarska inteligencija
Matematički fakultet

Aleksandra Jovičić 83/2015, Vladana Đorđević 89/2015
jovicicaleksandra96ajs@gmail.com, vladana1711@gmail.com

3. septembar 2019

Sadržaj

1 Uvod

Tema ovog istraživanja je primena veštačkih neuronskih mreža na skup podataka koji se odnosi na napadanje računarskih mreža u cilju utvrđivanja da li je data konekcija normalan saobraćaj ili napad. Ovaj skup je korišćen na KDD (Knowledge Discovery and Data Mining) kupu održanom 1999. godine i može se naći na ovde. Celokupan skup sadrži 5 000 000 instanci, a mi smo koristili 10% skupa (kddcup.data_10_percent) što iznosi 494020 instanci. Obrada ovih podataka rađena je u Python programskom jeziku.

Svaka instanca predstavlja jedan zapis o konekciji i opisana je pomoću 41 atributa i označena kao napad na mrežu određenog tipa ili kao normalan sadržaj. Svi atributi se mogu podeliti u tri grupe, mi smo za svaku grupu izdvojili najvažnije atribute i prikazali ih tabelarno[1]

1. osnovni atributi: u ovoj kategoriji se nalaze svi atributi koji mogu biti izdvojeni iz TCP/IP konekcije.

ime atributa	opis	tip
protocol_type	tip protokola	diskretan
service	internet servis na destinaciji	diskretan
src_bytes	broj bajtova podataka od izvora do destinacije	neprekidan
flag	status konekcije(normalan ili greška)	diskretan

2. atributi saobraćaja: ova kategorija uključuje atribute koji su izračunati uzimajući u obzir vremenski interval i podeljena je u dve grupe (”same host” i ”same service”).

ime atributa	opis	tip
count	br. kon. ka istom hostu kao i trenutna kon. u poslednje 2 sekunde	neprekidan
serror_rate	procenat konekcija koje imaju SYN greške	neprekidan
rerror_rate	procenat konekcija koje imaju REJ greške	neprekidan
same_srv_rate	procenat konekcije ka istom servisu	neprekidan
diff_srv_rate	procenat konekcije ka različitim servisima	neprekidan

3. atributi sadržaja: uključuje atribute koji pretražuju sumnjiva ponašanja u podacima.

ime atributa	opis	tip
num_failed_logins	broj neuspelih pokušaja prijavljivanja	neprekidan
logged_in	1 - za uspešno prijavljivanje, 0 - inače	diskretan
su_attempted	1 - ako je pokušana komanda ”su root”, 0 - inače	diskretan
num_file_creations	broj operacija kreiranja fajlova	neprekidan

1.1 Problem

Zajedno sa prednostima, Internet je istovremeno stvorio brojne načine da kompromituje stabilnost i bezbednost sistema koji su povezani na njega. Iako statički odbrambeni mehanizmi pružaju razuman nivo bezbednosti, takođe treba koristiti i dinamičke mehanizme, kao što su sistemi za otkrivanje upada. Sistemi za otkrivanje upada (SZOU) su obično ili zasnovani na hostu ili zasnovani na mreži. SZOU zasnovani na hostu nadgledaju resurse kao što su sistemski logovi, fajl sistemi i resursi sa diska, dok SZOU zasnovani na mreži nadgledaju podatke koji prolaze kroz mrežu. Potrebno je napraviti algoritam za učenje čiji je glavni

zadatak da otkrije direktno iz (trening) podataka odgovarajuće modele koji su u stanju da razlikuju normalno ponašanje od napada. Dobijeni model se potom koristi za klasifikaciju nad nepoznatim podacima.

KDD99 baza podataka je zasnovana na inicijativi 1998 DARPA da obezbedi dizajnerima sistema za otkrivanje upada benchmark pomoću koga će oceniti različite metodologije. Da bi se to postiglo napravljena je simulacija veštačke vojne mreže koja se sastoji od tri "ciljane" mašine na kojima se izvršavaju različiti operativni sistemi i servisi. Dodatne tri mašine su korišćene da"zamaskiraju" različite IP adrese i na taj način generišući saobraćaj između različitih IP adresa. Na kraju, korišćeno je "njuškalo" koje beleži sav mrežni saobraćaj. Ukupan simulirani period je sedam nedelja. Normalne konekcije su napravljene da bi se prikazalo šta je očekivano u vojnoj mreži, a napadi potpadaju u jednu od četiri kategorije: User to Root; Remote to Local; Denial of Service; Probe.[2]

- **Denial of Service (dos):** Napadač pokušava da spreči legitimne korisnike da koriste servis.
- **Remote to Local (r2l):** Napadač nema nalog na ciljanoj mašini, stoga pokušava da dobije pristup.
- **User to Root (u2r):** Napadač ima lokalni pristup ciljanoj mašini i pokušava da dobije root privilegije.
- **Probe:** Napadač pokušava da dobije informacije o ciljanom hostu.

1.2 Veštačke neuronske mreže

Veštačke neuronske mreže su inspirisane studijama o modeliranju ljudskog mozga. Implementacije u brojnim oblastima su pokazale odlične rezultate u pogledu efikasnosti i sposobnosti da se reše kompleksni problemi. Neke od klasa aplikacija na kojima su primenjivane veštačke neuronske mreže uključuju: klasifikaciju, pattern matching, pattern completion, optimizaciju, aproksimaciju funkcija, modeliranje vremenskih serija, data mining.

Neuronska mreža je praktično realizacija nelinearnog preslikavanja R^I u R^K.

$$f_{NN} : R^I \to R^K \tag{1}$$

gde su I i K dimenzije ulaza i izlaza, respektivno. Funkcija f_{NN} je obično kompleksna funkcija ili skup nelinearnih funkcija, jedna za svaki neuron u mreži. Neuroni su osnovni gradivni blok u neuronskoj mreži.

Jedan neuron može da implementira samo linearno razdvojivu funkciju. Čim je potrebno da se nauči funkcija koja nije linearno razdvojiva, neophodna je slojevita mreža neurona. Učenje tih mreža je kompleksnije nego učenje jednog neurona i ono može biti nadgledano ili nenadgledano.

S obzirom na to da u ovom radu vršimo klasifikaciju nad označenim podacima, u daljem tekstu ćemo se fokusirati na nadgledano učenje.

Nadgledano učenje zahteva trening skup koji se sastoji od ulaznih vektora i ciljnog vektora koji je spojen sa svakim ulaznim vektorom. Model neuronske mreže koristi ciljni vektor da bi odredio koliko dobro je naučio, kao i da prilagodi vrednosti težina da bi se smanjila celokupna greška.

Kada se mreža istrenira korišćenjem podataka iz trening skupa, uvode se novi podaci, takozvani test skup, kod kojih nedostaje atribut koji predstavlja klasu.

Ovim se postiže da mreža na osnovu prethodno stečenog znanja treniranjem, može u fazi testiranja da zaključi koje instance pripadaju kojoj klasi.[5]

2 Rešenje

2.1 Pretprocesiranje

Pre same implementacije veštačke neuronske mreže bilo je neophodno pripremiti podatke nad kojima se radi. Naime, potrebno je napraviti određene izmene u podacima tako da model bude što verodostojniji i da se izbegne preprilagođavanje i potprilagođavanje u klasifikaciji. Prvo su izbačeni duplikati, odnosno instance sa istim vrednostima za svaki atribut. Na taj način je smanjena verovatnoća da će doći do preprilagođavanja modela onim instancama koje se pojavljuju više puta.[4]

Nakon toga su izdvojene kolone sa atributima koji će biti korišćeni za učenje modela, kao i ciljna klasa.

```
x = information_base.iloc[:, :-1].values
y = information_base.iloc[:, 41].values
```

Listing 1: Izdvajanje klasa za učenje i ciljne klase

S obzirom na to da koristimo veštačku neuronsku mrežu, koja radi samo sa numeričkim podacima, bilo je potrebno kodirati kategoričke podatke u numeričke. To je postignuto korišćenjem funkcija LabelEncoder() i OneHotEncoder() iz biblioteke sklearn.preprocessing, a način korišćenja ovih funkcija je prikazan u narednom kôdu.

```
LEncoderX1 = LabelEncoder()
LEncoderX2 = LabelEncoder()
LEncoderX3 = LabelEncoder()

# Kolone na pozicijama 1,2 i 3 su kategoricke
x[:, 1] = LEncoderX1.fit_transform(x[:, 1])
x[:, 2] = LEncoderX2.fit_transform(x[:, 2])
x[:, 3] = LEncoderX3.fit_transform(x[:, 3])

# Kolona na poziciji 1 je imala 3 razlicite vrednosti, pa se deli na 3 kolone
OHEncoder1 = OneHotEncoder(categorical_features = [1])
x = OHEncoder1.fit_transform(x).toarray()

# Kolona na poziciji 2 se sada nalazi na poziciji 4 jer se prethodna kolona
    podelila na 3 kolone
OHEncoder2 = OneHotEncoder(categorical_features = [4])
x = OHEncoder2.fit_transform(x).toarray()

# Iz istog razloga se kolona na poziciji 3 sada nalazi na poziciji 70
OHEncoder3 = OneHotEncoder(categorical_features = [70])
x = OHEncoder3.fit_transform(x).toarray()

# Primena f-je LabelEncoder() na ciljnu klasu koja sadrzi kategoricke
    vrednosti
LEncoderY = LabelEncoder()
y = LEncoderY.fit_transform(y)
```

Listing 2: Transformacija kategoričkih atributa u numeričke

Funkcija LabelEncoder() transformiše kategoričke podatke tako što svaku vrednost atributa kodira u broj. Recimo da naš atribut ima 3 različite kategoričke vrednosti, LabelEncoder() će te vrednosti zameniti brojevima 0, 1 i 2 na odgovarajućim mestima u instanci. Međutim, ovakvo kodiranje uvodi novi

problem. Brojevi 0, 1 i 2 se mogu porediti ($0 < 1 < 2$) čime sugerišemo modelu da je neka vrednost veća (važnija) od druge, a to nije slučaj. Ovaj problem se rešava korišćenjem funkcije OneHotEncoder(). Funkcija uzima kolonu koja ima kategoričke podatke koji su transformisani korišćenjem LabelEncoder() i onda podeli tu kolonu u više kolona. Brojevi su zamenjeni nulama i jedinicama u zavisnosti od toga koja kolona ima tu vrednost. Na primer, ako je kategorička kolona imala 3 različite vrednosti sada je podeljena u 3 kolone gde svaka odgovara toj vrednosti. Instanca će imati jedinicu u koloni koja odgovara vrednosti koja se u njoj nalazila, a u ostale dve kolone će imati nule.

Kao što je već pomenuto, za nadgledano učenje je potrebno da postoje dva skupa - trening i test skup. Na prvom se model uči, a na drugom testira. Stoga je neophodno da naše podatke podelimo u ta dva skupa. Veličina test skupa će biti 30% celokupnih podataka, a parametrom random_state zadajemo seed za nasumičan odabir podataka. To se postiže korišćenjem funkcije train_test_split iz biblioteke sklearn.model_selection.

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3,
    random_state = 0)
```

Listing 3: Podela podataka na trening i test skup

Nakon toga su numerički podaci skalirani tako da vrednosti svakog atributa budu u istom rasponu korišćenjem StandardScaler() funkcije iz biblioteke sklearn.preprocessing. Na taj način se obezbeđuje da svaki numerički atribut podjednako učestvuje u pravljenju modela, a ne da se prednost daje onim atributima koji sadrže velike numeričke vrednosti.

```
scalerX = StandardScaler()
x_train = scalerX.fit_transform(x_train)
x_test = scalerX.transform(x_test)
```

Listing 4: Skaliranje podataka

2.2 Formiranje veštačke neuronske mreže

Želeli smo prvo da napravimo model koji će biti u stanju da razgraniči da li je konekcija normalna ili napad, bez obzira na to koji je napad u pitanju. Za to je bilo potrebno izmeniti ciljnu klasu tako da se ime svakog napada zameni vrednošću "attack".[3]

```
information_base['normal.'] = information_base['normal.'].replace(['back.', '
    buffer_overflow.', 'ftp_write.', 'guess_passwd.', 'imap.', 'ipsweep.', '
    land.', 'loadmodule.', 'multihop.', 'neptune.', 'nmap.', 'perl.', 'phf.'
    , 'pod.', 'portsweep.', 'rootkit.', 'satan.', 'smurf.', 'spy.', '
    teardrop.', 'warezclient.', 'warezmaster.'], 'attack')
```

Listing 5: Izmena imena napada u "attack"

Nakon što smo pripremili podatke tako da se nad njima može primeniti veštačka neuronska mreža bilo je potrebno da se ona napravi. Model se pravi pozivom funkcije Sequential() iz biblioteke keras.models, a nakon toga se mreža oblikuje dodavanjem slojeva, definisanjem broja čvorova u sloju, definisanjem načina inicijalizovanja težina kao i funkcijom aktivacije.

Naš model ima četiri sloja - ulazni, dva skrivena i izlazni. Ulazni sloj ima 118 neurona što odgovara broju kolona u našoj tabeli (izuzev ciljne klase). Na

početku smo imali 41 atribut, međutim, primenom LabelEncoder() i OneHotEncoder() funkcija jedna kolona se razbila na 3, druga na 66, treća na 11, što u zbiru sa ostalim netransformisanim kolonama (38) daje zbir 118. Upravo taj broj prosleđujemo kao argument za dimenziju ulaza funkcije Dense. Argument init govori mreži na koji način će da izabere inicijalne težine između čvorova i mi smo postavili da biranje bude uniformno. Za funkciju aktivacije je postavljena sigmoidna funkcija.

$$f_{AN}(net - \theta) = \frac{1}{1 + e^{-\lambda(net-\theta)}} \tag{2}$$

Pri čemu važi $f_{AN}(net) \in [0, 1]$. U istoj naredbi smo postavili i output_dim = 30, čime smo napravili jedan skriveni sloj od 30 neurona.

Naredne dve naredbe prave još jedan skriveni sloj sa 30 neurona i jedan izlazni sloj sa jednim neuronom, koji nam je dovoljan jer imamo samo dva moguća izlaza. Pri tome se u oba sloja koristi uniformno biranje inicijalnih težina i sigmoidna funkcija aktivacije.

```
# Inicijalizacija neuronske mreze
ANN = Sequential()

# Dodaje se ulazni sloj koji ima 118 neurona i jedan skriveni koji ima 30
    neurona
ANN.add(Dense(output_dim = 30, init = 'uniform', activation = 'sigmoid',
    input_dim = 118))

# Dodaje se skriveni sloj sa 30 neurona
ANN.add(Dense(output_dim = 30, init = 'uniform', activation = 'sigmoid'))

# Dodaje se izlazni sloj koji se sastoji od jednog neurona
ANN.add(Dense(output_dim = 1, init = 'uniform', activation = 'sigmoid'))
```

Listing 6: Pravljenje veštačke neuronske mreže

Pre samog treniranja modela potrebno je definisati proces učenja, što se postiže metodom compile koji ima tri argumenta. Prvi argument - optimizer - se odnosi na optimizator koji se koristi. Mi smo koristili Adam, algoritam zasnovan na gradijentnoj optimizaciji stohastičkih ciljnih funkcija. Računarski je efikasan, zahteva malo memorijskih resursa i pogodan je za probleme koji imaju mnogo podataka i/ili parametara. Drugi parametar - loss - definiše funkciju kojom se pokušava minimizovati gubitak. U našem slučaju, pošto ciljna klasa može imati samo dve vrednosti, koristimo binary_crossentropy funkciju, datu sledećom formulom:

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i * \log(p(y_i)) + (1 - y_i) * \log(1 - p(y_i)) \tag{3}$$

Treći argument - metrics - je funkcija koja se koristi za procenu performanse modela. Ona je veoma slična loss funkciji, osim što se rezultati evaluacije metrics f-je ne koriste tokom treniranja modela. Bilo koja loss funkcija se može koristiti kao metrics funkcija. Mi koristimo preciznost modela za procenu njegovog kvaliteta.

```
ANN.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['
    accuracy'])
ANN.fit(x_train, y_train, batch_size = 10, nb_epoch = 10)
```

Listing 7: Definisanje procesa učenja i učenje

Koristeći metod fit treniramo mrežu. Kao argumente mu prosleđujemo x i y trening skup, batch_size koji predstavlja broj instanci koje model trenira u jednoj iteraciji kao i nb_epoch koji predstavlja broj epoha. Mi smo postavili 10 kao vrednost oba argumenta.

Nakon što smo napravili model vreme je da ga primenimo na test skup. To se postiže funkcijom predict kojoj kao argument prosleđujemo test skup. Povratna vrednost funkcije su predviđene vrednosti ciljne klase u intervalu [0,1], a pošto su vrednosti ciljne klase 0 ili 1, potrebno je dobijene vrednosti koje su veće od 0.5 preslikati u 1, a one koje nisu u 0.

```
1000 y_pred = ANN.predict(x_test)
     y_pred = (y_pred > 0.5)
```

Listing 8: Primena modela na test skup

Na kraju, da bismo videli koliko naš model dobro radi, pozivamo funkciju confusion_matrix kojoj prosleđujemo prave i predviđene vrednosti test skupa, a koja vraća matricu konfuzije.

```
1000 cmatrix = confusion_matrix(y_test, y_pred)
```

Listing 9: Matrica konfuzije

Izvršavanje svih 10 epoha, matricu konfuzije i mere kvaliteta modela se mogu videti na sledećoj slici:

```
101909/101909 [==============================] - 11s 106us/step - loss: 0.0674 - acc: 0.9823
Epoch 2/10
101909/101909 [==============================] - 11s 106us/step - loss: 0.0172 - acc: 0.9957
Epoch 3/10
101909/101909 [==============================] - 11s 108us/step - loss: 0.0093 - acc: 0.9978
Epoch 4/10
101909/101909 [==============================] - 11s 103us/step - loss: 0.0078 - acc: 0.9981
Epoch 5/10
101909/101909 [==============================] - 11s 104us/step - loss: 0.0073 - acc: 0.9982
Epoch 6/10
101909/101909 [==============================] - 11s 104us/step - loss: 0.0067 - acc: 0.9983
Epoch 7/10
101909/101909 [==============================] - 10s 103us/step - loss: 0.0062 - acc: 0.9983
Epoch 8/10
101909/101909 [==============================] - 11s 104us/step - loss: 0.0058 - acc: 0.9985
Epoch 9/10
101909/101909 [==============================] - 11s 109us/step - loss: 0.0054 - acc: 0.9985
Epoch 10/10
101909/101909 [==============================] - 12s 114us/step - loss: 0.0051 - acc: 0.9986
[[17270    45]
 [   30 26331]]
Preciznost: 0.9982828097811155
Udeo lazno pozitivnih: 0.0017341040462427746
Udeo stvarno pozitivnih: 0.9988619551610334
Entropija je 0.0011373970201432806
```

Slika 1: Izvršavanje epoha, matrica konfuzije i mere kvaliteta modela

S obzirom na to da je model veoma precizan i dobro je razgraničio napade od normalne konekciji, želeli smo da napravimo još jedan model koji će raspoznavati vrste napada, a ne samo da li je napad ili nije. Veštačka neuronska mreža za drugi model je malo drugačija. Naime, sastoji se od tri sloja - ulaznog, skrivenog i izlaznog. Ulazni sloj koristi ReLU (Rectified Linear Unit) funkciju aktivacije. Skriveni ima 30 neurona dok izlazni ima 23 neurona, pri čemu svaki neuron predstavlja jednu klasu i koristi softmax funkciju aktivacije.

```
1000 ANN = Sequential()
     ANN.add(Dense(output_dim = 30, init = 'uniform', activation = 'relu',
          input_dim = 118))
1002 ANN.add(Dense(output_dim = 23, init = 'uniform', activation = 'softmax'))
```

Listing 10: Veštačka neuronska mreža za drugi model

Još jedna razlika je što je korišćena loss funkcija categorical_crossentropy zbog toga što naša klasa ima više od dve vrednosti. Iz tog razloga nije korišćena sigmoidna funkcija aktivacije, već softmax.

```
1000 ANN.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics =
         ['accuracy'])
     ANN.fit(x_train, y_train, batch_size = 10, epochs = 10)
```

Listing 11: Definisanje procesa učenja i učenje za drugi model

Softmax funkcija je aktivaciona funkcija koja transformiše brojeve u verovatnoće koji sumiranjem daju jedan. Izlaz funkcije je vektor koji predstavlja verovatnoće raspodele liste mogućih događaja (izlaza).

Naredna slika pokazuje izvršavanje epoha drugog modela i krajnju preciznost.

```
101909/101909 [==============================] - 10s 99us/step - loss: 0.0791 - acc: 0.9854
Epoch 2/10
101909/101909 [==============================] - 10s 96us/step - loss: 0.0174 - acc: 0.9960
Epoch 3/10
101909/101909 [==============================] - 10s 96us/step - loss: 0.0146 - acc: 0.9967
Epoch 4/10
101909/101909 [==============================] - 10s 96us/step - loss: 0.0119 - acc: 0.9972
Epoch 5/10
101909/101909 [==============================] - 10s 96us/step - loss: 0.0108 - acc: 0.9975
Epoch 6/10
101909/101909 [==============================] - 10s 96us/step - loss: 0.0102 - acc: 0.9977
Epoch 7/10
101909/101909 [==============================] - 12s 115us/step - loss: 0.0093 - acc: 0.9978
Epoch 8/10
101909/101909 [==============================] - 11s 108us/step - loss: 0.0090 - acc: 0.9979
Epoch 9/10
101909/101909 [==============================] - 10s 95us/step - loss: 0.0084 - acc: 0.9982
Epoch 10/10
101909/101909 [==============================] - 10s 100us/step - loss: 0.0083 - acc: 0.9982
```

Slika 2: Izvršavanje epoha za drugi model

Različiti tipovi napada imaju različiti nivo zastupljenosti u skupu podataka, što dovodi do toga da se manje zastupljeni napadi slabije klasifikuju. Iz tog razloga pokušali smo da koristimo "cross validation" sa metodom podele "stratified n fold" koja deli skup podataka na n podskupova u kojima su zastupljene sve klase. Problem predstavljaju retke klase koje nemaju čak ni n instanci. Njih ne možemo da predvidimo i o njima ne možemo ništa da naučimo. Nismo našli funkciju koja radi ovaj metod za nas pa smo iskoristili petlju. U petlji smo išli po svakom izdvojenom podskupu i na kraju smo izvojili srednju vrednost preciznosti i udela stvarno pozitivnih instanci. Mesta koja imaju nulu upravo jesu retke klase sa nedovoljnim brojem instanci koje nismo izbacili sa namerom da prikažemo koje su to klase.

```
1000 kfold = StratifiedKFold(n_splits=20, shuffle=True, random_state=seed)
```

Listing 12: StratifiedKFold funkcija

3 Karakteristike sistema

Programi su izvršavani na sistemu sa sledećim karakteristikama:

- Processor: Intel Core i3-6100U CPU @ 2.30GHz x 4
- Graphics: Intel HD Graphics 520 (Skylake GT2)

- Memory: 7,2 GiB
- GNOME: 3.28.2
- OS: Ubuntu 18.04 LTS
- OS type: 64-bit
- Disk: 130,1 GB

4 Eksperimentalni rezultati

Da bismo uvideli u kojim slučajevima neuronska mreža najbolje radi, istraživali smo šta se dešava kada promenimo i kombinujemo neke parametre. Menjali smo broj slojeva, čvorova u sloju, iteracija, kao i funkciju aktivacije. Uočili smo da sigmoidna funkcija daje bolje rezultate u slučaju manjeg broja slojeva, a lošije u prisustvu više unutrašnjih slojeva zbog svoje osobine "nestajućeg gradijenta". Ta osobina smanjuje učenje u unutrašnjim slojevima. Međutim, relu funkcija radi dobro za mreže sa više unutrašnjih slojeva i prevazilazi ovaj problem. Mi smo koristili dva unutrašnja sloja i iz tog razloga i sigmoidnu funkciju pri određivanju da li je konekcija napad ili ne. Za predviđanja koji tip napada se desio koristili smo funkciju softmax koja najbolje radi kada imamo više klasa koje smo kodirali preko OneHotEncoder-a. Broj čvorova u sloju kao i broj iteracija ne utiče znatno na preciznost i udeo stvarno pozitivnih vrednosti u mreži.

5 Zaključak

Veštačke neuronske mreže su izuzetno moćan alat za klasifikaciju podataka. Mogu se primeniti da veliku količinu različitih podataka i napraviti model koji veoma precizno predviđa vrednosti. Međutim, potrebno je biti oprezan u pravljenju modela da ne bi došlo do preprilagođavanja i potprilagođavanja modela. Postoji mnogo vrsta mreža i načina za primenu i potrebno je dobro poznavanje parametara i načina funkcionisanja mreža da bi rezultati bili zadovoljavajući. Iako je prošlo mnogo vremena otkako su prikupljeni podaci iz Network Intrusion Data i dalje je jedan on najpopularnijih i najboljih skupova podataka za detekciju upada na mreže. Zahvaljujući dobro raspoređenim i sveobuhvatnim podacima primena veštačkih neuronskih mreža na njih je dala izuzetno dobre rezultate u razlikovanju napada od normalne konekcije. Neki veoma retki napadi su slabo klasifikovani, što je logično uzevši u obzir da mreža nije imala dovoljno informacija o njima da bi ih prepoznala, ali celokupna klasifikacija je dala zadovoljavajuće rezultate. Tako da se može zaključiti da je primena veštačkih neuronskih mreža na Network Intrusion Data bila uspešna.

Literatura

[1] Atributi. on-line at: https://kdd.ics.uci.edu/databases/kddcup99/task.html.

[2] Napadi. on-line at: https://web.cs.dal.ca/ zincir/bildiri/pst05-gnm.pdf.

[3] Po uzoru na kod. on-line at:https://github.com/vinayakumarr/Network-Intrusion-Detection/blob/master/KDDCup

[4] Muniyandi R. C. Azzawi M. A. Abdulmajed, E. S. A Novel Method of Preprocessing and Evaluating the KDD CUP 99 Data Set. *Adv Research in Dynamical Control Systems*, 9(10), 2017.

[5] A. Engelbrecht. *Computational Intelligence - An Introduction.* John Willey Sons, 2007.