Implementación y Análisis de Técnicas Híbridas de Aprendizaje Automático en la Detección Intrusos en Redes de Computadoras

Deyban Andrés Pérez Abreu

Introducción

El presente documento recopila las actividades realizadas en la elaboración del **Trabajo Especial de Grado** de mi persona (autor del documento). Este tiene cómo tema el **Análisis e Implementación de Técnicas Híbridas de Aprendizaje Automático en la Detección de Intrusos en Redes de Computadoras* haciendo uso del conjunto de datos** NSL-KDD**.

Los objetivos que se buscan a lograr con este trabajo es la implementación y análisis de modelos **basados en la firma del ataque**, cómo lo son las **Redes Neuronales** y **Máquinas de Soporte Vectorial** en conjunto con técnicas **basadas en anomalías** cómo lo es **K-Medias**. La idea de esta mezcla de paradigmas es la complementación de estos con la esperanza de mejorar el rendimiento desde un punto de vista de **eficacia** a la hora de detectar anomalías en una red de computadoras, específicamente, con la utilización de técnicas basadas en anomalías se busca detectar aquellos ataques conocidos que fueorn provistos en el conjunto de entrenamiento, y con las técnicas basadas en anomalías se busca capturar aquellas nuevas anomalías que no fueron provistas en la fase de entrenamiento al algoritmo.

El conjunto de datos **NSL-KDD** consta de un conjunto de entrenamiento y un conjunto de pruebas excluyentes, es decir, que ningún registros está duplicado entre conjuntos. Adicionalmente, el conjunto de datos de prueba posee ataques que no son proporcionados en el conjunto de entrenamiento, así que la idea es evaluar la capacidad de generalización de los modelos creados simulando un ambiente real de prueba, donde nuevos ataques surgen constantemente.

La tareas a realizar se pueden dividir en tres grandes grupos que se mencionarán a continunación:

1. La primera fase corresponde al pre-procesamiento de los datos, esto aplica tanto al conjunto de entrenamiento como al conjunto de prueba. En este paso se busca crear una vista minable que facilite la manipulación de la información y estandarice los tipos de datos a ser utilizados a lo largo de la investigación.

2. La segunda fase corresponde a la demostración de la eficacia de la propuesta planteada con aterioridad, es decir, la prueba de los modelos híbridos a la hora de realizar las tareas de detección. Esta fase se dividirá en dos conjuntos.

- **Pruebas sobre el conjunto de entrenamiento**: acá se realizarán las pruebas extrayendo un subconjunto de los datos para la prueba y el restante para el entrenamiento y se evaluará el rendimiento de cada uno de los modelos.
- **Pruebas sobre el conjunto de prueba**: acá se tomará el conjunto de entrenamiento en su totalidad para realizar las tareas de entrenamiento y se hará la prueba sobre el conjunto total de prueba provisto por el conjunto de datos NSL-KDD.

NOTA: En esta fase los modelos serán entrenados haciendo uso de parámetros por defecto.

3. La tercera fase corresponde al proceso de selección de carcaterísticas y selección de parámetros, en esta fase se analizan los resultados obtenidos del proceso de reducción de características y ajuste de los parámetros para los modelos.

Pre-Procesamiento de los datos

En esta sección se listarán las actividades realizadas concernientes al proceso de pre-procesamiento de los datos. Esta tarea aplica para los conjuntos de datos de entrenamiento y de prueba, debido a que ambos conjuntos de datos deben poseer el mismo formato para poder realizar el proceso de aprendizaje automático.

Comenzaremos con la configuración del ambiente de trabajo, donde se eliminarán las variables del ambiente de trabajo. Y se cargará un archivo con funciones llamado **functions.R**, este archivo posee una leyenda donde se explica a cabalidad el funcionamiento de cada una de las funciones ilustradas en dicho documento.

```
rm(list = ls())
source("../source/functions/functions.R")
```

A continuación se cargarán los conjuntos de prueba y de entrenamiento a ser utilizados.

```
dataset.training = read.csv(file = "../dataset/KDDTrain+.txt", sep = ",", header = FALSE)
dataset.testing = read.csv(file = "../dataset/KDDTest+.txt", sep = ",", header = FALSE)
```

En la variable **dataset.training** se encuentra cargado el conjunto de entrenamiento y en la variable **dataset.testing** se tiene cargado el conjunto de prueba. Veamos las dimensiones de los conjuntos de datos.

```
dim(dataset.training)
```

```
## [1] 125973     43
```

```
dim(dataset.testing)
```

```
## [1] 22544    43
```

El **conjunto de entrenamiento** tiene 125973 filas y 43 columnas. Por otra parte, el **conjunto de prueba** tiene 22544 filas y 43 columnas. Es importante mencionar que de las 43 columnas, la **columnas 42** corresponde a la etiqueta del ataque y la **columna 43** corresponde a la cantidad de clasificadores que acertaron a la hhora de clasificar dicho registros en el proceso de creación dle conjunto de datos NSL-KDD. En el proceso previamente mencionado se utilizaron 21 clasificadores, por dicho motivo, el rango de número en esta columna está comprendido por [0,21]. A continuación veamos si los conjuntos de datos poseen valores faltantes, para esllo haremos uso de la función **complete.cases**.

```
sum(complete.cases(dataset.training)) == nrow(dataset.training)
```

```
## [1] TRUE
```

```
sum(complete.cases(dataset.testing)) == nrow(dataset.testing)
```

```
## [1] TRUE
```

Se observa que la cantidad de casos completos es igual a la cantidad de filas de ambos conjuntos de datos, por tal motivo no existen valores faltantes. Ahora veamos los tipos de ataques por conjuntos de datos. Empezaremos por con el conjunto de entrenamiento.

```
attacks.training = unique(dataset.training$V42)
attacks.training = sort(as.character(attacks.training))
length(attacks.training)
```

```
## [1] 23
```

Se observa que el conjunto de entrenamiento consta de 23 etiquetas, donde 1 corresponde a la etiqueta de **tráfico normal**, y las otras 22 corresponden a **ataques¨. Ahora veamos el conjunto de prueba.

```
attacks.testing = unique(dataset.testing$V42)
attacks.testing = sort(as.character(attacks.testing))
length(attacks.testing)
```

```
## [1] 38
```

Se observan 38 etiquetas en el conjunto de prueba, donde 1 corresponde a la etiqueta de **tráfico normal** y las otras 37 corresponden a **ataques**. En este punto se puede observar cómo hay mayor cantidad de ataques en el conjunto de prueba que en el conjunto de entrenamiento, esto es debido a que el conjunto de prueba busca medir la habilidad del modelo de ML para generalizar ante ataques no vistos en el conjunto de entrenamiento con anterioridad.

A continuacuón se observan cuales son los ataques presentes en el conjunto de prueba que no están presentes en el conjunto de prueba y viceversa. Se empezará con examinar la cantidad total de ataques presentes entre ambos conjuntos.

```
total.attacks = sort(unique(c(attacks.training, attacks.testing)))
length(total.attacks)
```

```
## [1] 40
```

Entre ambos conjuntos se observan 40 etiquetas, donde una corresponde al **tráfico normal** y las otras 39 corresponden a etiqueras de **ataques**. De lo anterior se puede concluir que hay 17 tipos de ataques presentes en el conjunto de prueba que no están presentes en el conjunto de entrenamiento, y que hay dos tipos de ataques en el conjunto de entrenamiento que no están presentes en el conjunto de prueba. A continuación se listarán aquellas etiquetas comunes entre ambos conjuntos de datos.

```
total.attacks = sort(unique(c(attacks.training, attacks.testing)))
length(total.attacks)
```

```
## [1] 40
```

```
total.attacks
```

```
##  [1] "apache2"         "back"            "buffer_overflow"
##  [4] "ftp_write"       "guess_passwd"    "httptunnel"
##  [7] "imap"            "ipsweep"         "land"
## [10] "loadmodule"      "mailbomb"        "mscan"
## [13] "multihop"        "named"           "neptune"
## [16] "nmap"            "normal"          "perl"
## [19] "phf"             "pod"             "portsweep"
```

```
## [22] "processtable"    "ps"              "rootkit"
## [25] "saint"           "satan"           "sendmail"
## [28] "smurf"           "snmpgetattack"   "snmpguess"
## [31] "spy"             "sqlattack"       "teardrop"
## [34] "udpstorm"        "warezclient"     "warezmaster"
## [37] "worm"            "xlock"           "xsnoop"
## [40] "xterm"
```

Se observa que existen 21 etiquetas conmunes entre ambos conjuntos de datos, donde 1 corresponde a la etiqueta de **tráfico normal** y las otras 20 corresponde a *ataques**. Todas las etiquetas fueron listadas. A continuación se listarán aquellos ataques que están presentes en el conjunto de prueba y no en el conjunto de entrenamiento.

```
index.attacks = which(attacks.testing %in% attacks.training)
length(attacks.testing[-index.attacks])
```

```
## [1] 17
```

```
attacks.testing[-index.attacks]
```

```
##  [1] "apache2"       "httptunnel"    "mailbomb"      "mscan"
##  [5] "named"         "processtable"  "ps"            "saint"
##  [9] "sendmail"      "snmpgetattack" "snmpguess"     "sqlattack"
## [13] "udpstorm"      "worm"          "xlock"         "xsnoop"
## [17] "xterm"
```

Son 17 los ataques presentes en el conjunto de prueba que no están presentes en el conjunto de entrenamiento, los mismos fueron listados. A continuación se listarán aquellos ataques presentes en el conjunto de entrenamiento que no lo están en el conjunto de prueba.

```
index.attacks.training = which(attacks.training %in% attacks.testing)
length(attacks.training[-index.attacks.training])
```

```
## [1] 2
```

```
attacks.training[-index.attacks.training]
```

```
## [1] "spy"         "warezclient"
```

Son sólo 2 los ataques en el conjunto de entrenamiento que no están presentes en el conjunto de prueba. Estos corresponden a **spy** y ** warezclient**.

Extracción de características

En este documento se clasifican las anomalías en cuatro grupos **DoS**, **Probing**, **R2L** y **U2R**, es decir, habrán 5 etiquetas, donde 4 corresponden a los tipos de ataques mencionados previamente y la 5ta etiqueta corresponde a la etiqueta normal.

Para facilitar el trabajo se debe asociar cada uno de los ataques a cada una de las clases mencionadad con anterioridad. Para esto se hará uso de la función **ClassLabelAttack** que recibe cómo parámetro un **dataframe** y retorna una columna con la clase de cada tipo de ataque para cada registro. Estos nombres colocados acordes a la investigación hecha por **Bhavsar**.

```
dataset.training$V44 = ClassLabelAttack(dataset.training)
dataset.testing$V44 = ClassLabelAttack(dataset.testing)
```

De esta manera, tanto el **conjunto de entrenamiento** como el **conjunto de prueba** tienen una nueva columna en la que cada registro tiene asociada la respectiva clase a la que pertenece. Adicionalmente se agregó una nueva columna que corresponde a una nueva etiqueta que identifica a cada registro como **ataque** o **normal**. De esta manera se tiene una clase general para la asociación de los registros.

```
dataset.training$V45 = NormalAttackLabel(dataset.training)
dataset.testing$V45 = NormalAttackLabel(dataset.testing)
```

Ahora se dividirá el conjunto de datos en **dataframes** individuales para cada clase: **DoS**, **normal**, **R2L**, **U2R**.

```
training.split = split(dataset.training, dataset.training$V44)
testing.split = split(dataset.testing, dataset.testing$V44)
summary(training.split)
```

```
##         Length Class      Mode
## DoS     45     data.frame list
## normal  45     data.frame list
## Probing 45     data.frame list
## R2L     45     data.frame list
## U2R     45     data.frame list
```

```
summary(testing.split)
```

```
##         Length Class      Mode
## DoS     45     data.frame list
## normal  45     data.frame list
## Probing 45     data.frame list
## R2L     45     data.frame list
## U2R     45     data.frame list
```

Las variables **training.split** y **testing.split** contienen una lista de sub-conjuntos por eriquetas de las clases de los ataques en ambos conjuntos de datos. A continuación se listarán el número de cada clase en el conjunto de entrenamiento.

```
nrow(training.split$DoS)
```

```
## [1] 45927
```

```
nrow(training.split$normal)
```

```
## [1] 67343
```

```
nrow(training.split$Probing)
```

```
## [1] 11656
```

```r
nrow(training.split$R2L)
```

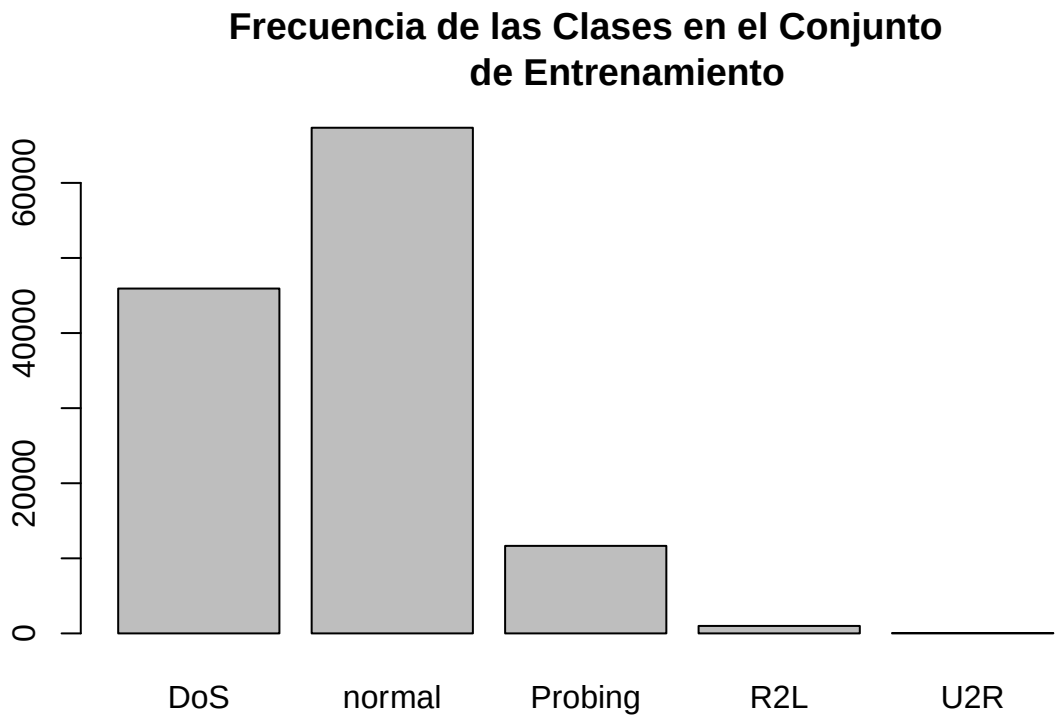
```
## [1] 995
```

```r
nrow(training.split$U2R)
```

```
## [1] 52
```

Se observa que la clase **normal** es la que más registros posee en el conjunto de datos de entrenamiento, seguido por la clase **DoS**. lo anterior nos da una idea de cuáles son las clases de ataques más comunes y menos comunes. A continuación se presenta un gráfico que ilustra lo anterior y permite visualizar mejor la distribución de las clases.

```r
barplot(table(dataset.training$V44), main = "Frecuencia de las Clases en el Conjunto
        de Entrenamiento")
```



A continuación se repiten los pasos anteriores para el conjunto de prueba.

```r
nrow(testing.split$DoS)
```

```
## [1] 7458
```

```r
nrow(testing.split$normal)
```

```
## [1] 9711
```

```r
nrow(testing.split$Probing)
```

```
## [1] 2421
```

```r
nrow(testing.split$R2L)
```

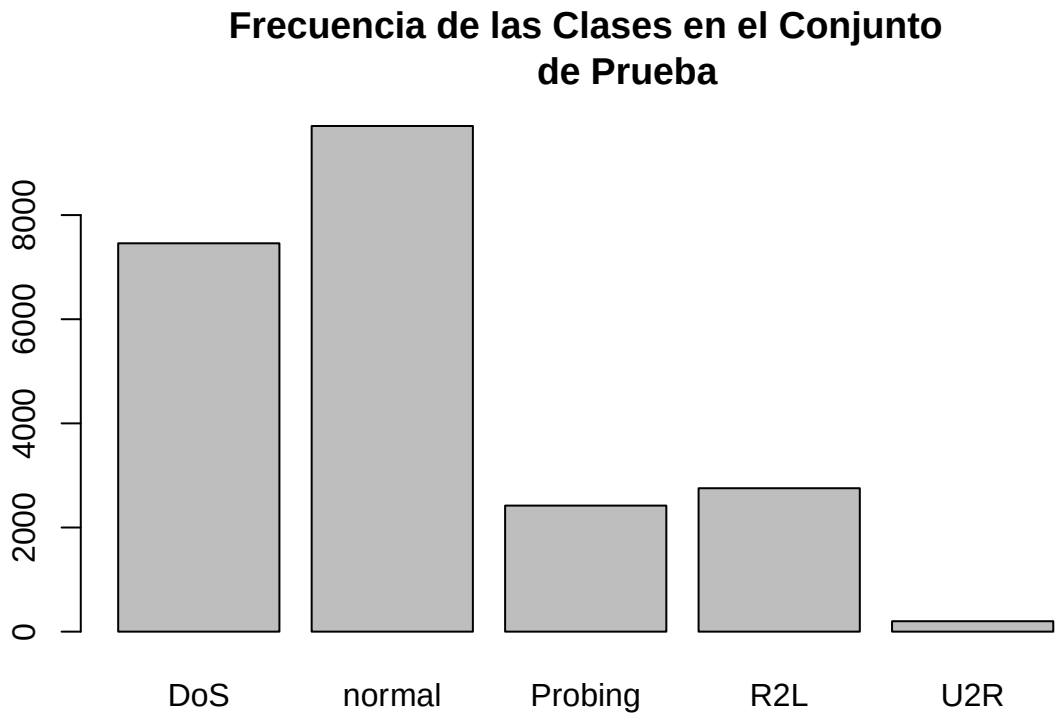
```
## [1] 2754
```

```r
nrow(testing.split$U2R)
```

```
## [1] 200
```

En esta oportunidad la clase **normal** sigue siendo la clase con mayor cantidad de registros. En contraste con el conjunto de prueba, se observa que en esta ocasión las clases **Probing** y **R2L** están más equilibradas, adicionalmente, la clase **U2R** posee una cantidad mucho mayor de registros que en el conjunto de entrenamiento. A continuación se presenta un gráfico con las distribuciones de las clases en el conjunto de prueba.

```r
barplot(table(dataset.testing$V44), main = "Frecuencia de las Clases en el Conjunto
        de Prueba")
```



Renombramiento de las columnas

Se hará uso de la función **ColumnNames** que asigna a los conjuntos de datos los nombres respectivos, estos nombres colocados acordes a la investigación hecha por **Bhavsar**.

```
dataset.training = ColumnNames(dataset.training)
dataset.testing = ColumnNames(dataset.testing)
```

Eliminación de características no importantes

En esta sección se examinaran posibles características inútiles, esto es, aquellas características que sólo tienen un nivel de valores, por ejemplo, una característica de tipo **numérico** donde en todos los registros el valor es cero (0), es decir, el rango viene dado por [0]. Para dicho propósito se utilizará la función **CheckFeaturesLevels** de toma cómo entrada un dataframe y retorna la posición (si existe) de la característica que no aporta información.

```
index.dummy.variables.training = CheckFeaturesLevels(dataset.training)
index.dummy.variables.testing = CheckFeaturesLevels(dataset.testing)
names(dataset.training)[index.dummy.variables.training]
```

```
## [1] "Num_outbound_cmds"
```

```
names(dataset.testing)[index.dummy.variables.testing]
```

```
## [1] "Num_outbound_cmds"
```

Se observa que en ambos conjuntos de datos la columna *Num_outbound_cmds* es inútil, en consecuencia, la misma será eliminada del conjunto de datos.

```
dataset.training[,index.dummy.variables.training] = NULL
dataset.testing[, index.dummy.variables.testing] = NULL
```

Tranformación de los datos

Las columnas **Protocol_type**, **Service** y **Flag** tienen tipos de datos categóricos, los mismos serán transformados a numéricos. La transformación tiene su justificación en el hecho de que los algoritmos a utilizar que son **Redes Neuronales**, **Máquinas de Soporte Vectorial** y **K-Medias** funcionan con predictores (características) numéricas. Dicho esto es obligatorio transformar las columnas de tipo categórico a tipo numérico.

1. **Protocol_type**: esta característica posee 3 niveles, que serán listados alfabeticamente a continuación.

```
sort(unique(dataset.training$Protocol_type))
```

```
## [1] icmp tcp  udp
## Levels: icmp tcp udp
```

```
sort(unique(dataset.testing$Protocol_type))
```

```
## [1] icmp tcp  udp
## Levels: icmp tcp udp
```

Los mismos se tranformarán en los valores 1,2,3 respectivamente. La función **ProtocolTranformation** es la encargada de realizar dicho trabajo.

```
dataset.training = ProtocolTransformation(dataset.training)
dataset.testing = ProtocolTransformation(dataset.testing)
```

2. **Service**: esta característica posee una mayor cantidad de niveles con respecto a **Protocol_type**, los mismo serán listados a continuación.

```
sort(unique(dataset.training$Service))
```

```
##  [1] aol         auth        bgp         courier     csnet_ns
##  [6] ctf         daytime     discard     domain      domain_u
## [11] echo        eco_i       ecr_i       efs         exec
## [16] finger      ftp         ftp_data    gopher      harvest
## [21] hostnames   http        http_2784   http_443    http_8001
## [26] imap4       IRC         iso_tsap    klogin      kshell
## [31] ldap        link        login       mtp         name
## [36] netbios_dgm netbios_ns  netbios_ssn netstat     nnsp
## [41] nntp        ntp_u       other       pm_dump     pop_2
## [46] pop_3       printer     private     red_i       remote_job
## [51] rje         shell       smtp        sql_net     ssh
## [56] sunrpc      supdup      systat      telnet      tftp_u
## [61] time        tim_i       urh_i       urp_i       uucp
## [66] uucp_path   vmnet       whois       X11         Z39_50
## 70 Levels: aol auth bgp courier csnet_ns ctf daytime discard ... Z39_50
```

```
sort(unique(dataset.testing$Service))
```

```
##  [1] auth        bgp         courier     csnet_ns    ctf
##  [6] daytime     discard     domain      domain_u    echo
## [11] eco_i       ecr_i       efs         exec        finger
## [16] ftp         ftp_data    gopher      hostnames   http
## [21] http_443    imap4       IRC         iso_tsap    klogin
## [26] kshell      ldap        link        login       mtp
## [31] name        netbios_dgm netbios_ns  netbios_ssn netstat
## [36] nnsp        nntp        ntp_u       other       pm_dump
## [41] pop_2       pop_3       printer     private     remote_job
## [46] rje         shell       smtp        sql_net     ssh
## [51] sunrpc      supdup      systat      telnet      tftp_u
## [56] time        tim_i       urp_i       uucp        uucp_path
## [61] vmnet       whois       X11         Z39_50
## 64 Levels: auth bgp courier csnet_ns ctf daytime discard ... Z39_50
```

Se observa que en el **conjunto de entrenamiento** hay un total de 70 niveles, contra 64 niveles presentes en el **conjunto de prueba**. Observemos la cantidad total de servicios uniendo ambos conjuntos.

```
sort(unique(dataset.training$Service))
```

```
##  [1] aol         auth        bgp         courier     csnet_ns
##  [6] ctf         daytime     discard     domain      domain_u
## [11] echo        eco_i       ecr_i       efs         exec
## [16] finger      ftp         ftp_data    gopher      harvest
```

```
## [21] hostnames   http        http_2784   http_443    http_8001
## [26] imap4       IRC         iso_tsap    klogin      kshell
## [31] ldap        link        login       mtp         name
## [36] netbios_dgm netbios_ns  netbios_ssn netstat     nnsp
## [41] nntp        ntp_u       other       pm_dump     pop_2
## [46] pop_3       printer     private     red_i       remote_job
## [51] rje         shell       smtp        sql_net     ssh
## [56] sunrpc      supdup      systat      telnet      tftp_u
## [61] time        tim_i       urh_i       urp_i       uucp
## [66] uucp_path   vmnet       whois       X11         Z39_50
## 70 Levels: aol auth bgp courier csnet_ns ctf daytime discard ... Z39_50
```

Se observa que el total de servicios es de 70, es decir, el conjunto de servicios en el conjunto de entrenamiento corresponde al universo de todos los servicios en los conjuntos de datos.

Los niveles serán enumerados en en rango [1,70] en orden alfabético, tal cómo se muestra a continuación.

```
sort(unique(c(as.character(unique(dataset.testing$Service)),
              as.character(unique(dataset.training$Service)))))
```

```
##  [1] "aol"         "auth"        "bgp"         "courier"     "csnet_ns"
##  [6] "ctf"         "daytime"     "discard"     "domain"      "domain_u"
## [11] "echo"        "eco_i"       "ecr_i"       "efs"         "exec"
## [16] "finger"      "ftp"         "ftp_data"    "gopher"      "harvest"
## [21] "hostnames"   "http"        "http_2784"   "http_443"    "http_8001"
## [26] "imap4"       "IRC"         "iso_tsap"    "klogin"      "kshell"
## [31] "ldap"        "link"        "login"       "mtp"         "name"
## [36] "netbios_dgm" "netbios_ns"  "netbios_ssn" "netstat"     "nnsp"
## [41] "nntp"        "ntp_u"       "other"       "pm_dump"     "pop_2"
## [46] "pop_3"       "printer"     "private"     "red_i"       "remote_job"
## [51] "rje"         "shell"       "smtp"        "sql_net"     "ssh"
## [56] "sunrpc"      "supdup"      "systat"      "telnet"      "tftp_u"
## [61] "time"        "tim_i"       "urh_i"       "urp_i"       "uucp"
## [66] "uucp_path"   "vmnet"       "whois"       "X11"         "Z39_50"
```

Se utilizará la función **ServiceTransformation** para enumerar cada uno de los servicios listados previamente.

```
dataset.training = ServiceTransformation(dataset.training)
dataset.testing = ServiceTransformation(dataset.testing)
```

3. **Flag**: es la característica categórica restante. Observemos los niveles de esta características.

```
sort(unique(dataset.training$Flag))
```

```
##  [1] OTH    REJ    RSTO   RSTOS0 RSTR   S0     S1     S2     S3     SF
## [11] SH
## Levels: OTH REJ RSTO RSTOS0 RSTR S0 S1 S2 S3 SF SH
```

```
sort(unique(dataset.testing$Flag))
```

```
##  [1] OTH    REJ    RSTO   RSTOS0 RSTR   S0     S1     S2     S3     SF
## [11] SH
## Levels: OTH REJ RSTO RSTOS0 RSTR S0 S1 S2 S3 SF SH
```

```
length(sort(unique(c(as.character(unique(dataset.testing$Flag)),
                     as.character(unique(dataset.training$Flag))))))
```

```
## [1] 11
```

Se observa que hay 11 niveles en ambos conjuntos y que la unión de los niveles de ambos conjuntos de datos arroja el mismo resultado. Dicho esto, las etiquetas serán enumeradas por orden alfabético, tal cómo se muestra a continuación.

```
sort(unique(c(as.character(unique(dataset.testing$Flag)),
              as.character(unique(dataset.training$Flag)))))
```

```
##  [1] "OTH"    "REJ"    "RSTO"   "RSTOS0" "RSTR"   "S0"     "S1"
##  [8] "S2"     "S3"     "SF"     "SH"
```

Se utilizará la función FlagTransformation para dicho propósito.

```
dataset.training = FlagTransformation(dataset.training)
dataset.testing = FlagTransformation(dataset.testing)
```

Guardando la vista minable

En este punto la vista minable ya fue creada, las columnas poseen un formato aceptable para los algoritmos que serán utilizados y se agregaron nuevas columnas que facilitarán tareas futuras en la investigación. Debido a que no hay más tareas por hacer, se procede a guardar los conjuntos de datos para cargar los datos preprocesados y no tener que repetir dicho procedimiento luego.

```
write.csv(dataset.training,
          file = "../dataset/NSLKDD_Training_New.csv", row.names = FALSE)
write.csv(dataset.testing,
          file = "../dataset/NSLKDD_Testing_New.csv", row.names = FALSE)
```

Implementación de modelos híbridos

La propuesta del **trabajo especial de grado** consta del entrenamiento de dos modelos híbridos de aprendizaje automático. El primer modelo (i) consta de una **red neuronal** en el primer nivel y **K-Medias** en el segundo, por otra parte, el segundo modelo (ii) consta de una **máquina de soporte vectorial** en el primer nivel y **K-Medias** en el segundo nivel.

En esta sección los modelos serán entrenados con los parámetros por defecto. Porteriormente se hará selección de características y parámetros y se analizará el impacto con respecto a los modelos creados en esta sección.

Esta sección será divida en dos grandes secciones, una concerniente al entrenamiento y evaluación de los modelos utilizando el **conjunto de entrnamiento** exclusivamente. En esta fase se hará uso de la **técnica de validación cruzada de 10 conjuntos** para evaluar los modelos. Posteriormente, se hará el entrenamiento haciendo uso total del conjunto de entrenamiento y se hará evaluación de los modelos haciendo uso del conjunto de prueba.

Con la estrategia descrita en el párrafo se podrán observar dos aspectos relevantes:

1. La eficacia de los modelos contra ataques conocidos.
2. La eficacia de los modelos contra ataques no conocidos.
3. Diferencias de rendimiento entre ambos modelos.

Pruebas sobre el conjunto de entrenamiento

En esta sección se listarán las actividades concernientes al entrenamiento y evaluación de los modelos híbridos haciendo uso exclusivo del conjunto de entrenamiento y de la técnica de **validación cruzada de 10 conjuntos** para la evaluación de los modelos.

Inicialmente iniciaremos con una evaluación del rendimiento de **K-Medias**, se elegirán los centroides y se evaluará su desempeño en la tarea de detección de intrusos.

K-Medias

Se empezará por establecer el ambiente de trabajo eliminando las variables parciañes, cargando el archivo de funciones y la vista minable del conjunto de entranmiento pre-procesado previamente.

```
rm(list = ls())
source("../source/functions/functions.R")
dataset.training = read.csv("../dataset/NSLKDD_Training_New.csv",
                            sep = ",", header = TRUE)
```

Para esta sección sólo se necesitaran las etiquetas **Label_Normal_ClassAttack** y **Label_Normal_or_Attack**, por lo tanto las otras etiquetas serán eliminadas.

```
dataset = dataset.training
dataset$Label_Normal_TypeAttack = NULL
dataset$Label_Num_Classifiers = NULL
```

A continuación se le asignará el tipo numérico a todas las **columnas predictoras**, y el tipo factor a las **columnas de etiquetas**.

```
for (i in 1 : (ncol(dataset) -2) )
  dataset[,i] = as.numeric(dataset[,i])

for (i in (ncol(dataset) -1):ncol(dataset) )
  dataset[,i] = as.factor(dataset[,i])
```

En este punto se crearán dos nuevos conjuntos de datos, **dataset.two** que tendrá cómo etiqueta la columna **Label_Normal_or_Attack**, columna que sólo tiene dos niveles categóricos **Attack** o **normal**. Por otra parte, se creará un segundo conjunto de datos llamado **dataset.five** el cuál contendrá cómo etiqueta la columna **Label_Normal_ClassAttack**, columna que tiene cinco niveles categóricos **DoS**, **normal**, **Probing**, **R2L** y **U2R**.

```
dataset.two = dataset[-(ncol(dataset)-1)]
dataset.two[, ncol(dataset.two)] = as.character(dataset.two[, ncol(dataset.two)])
dataset.five = dataset[-ncol(dataset)]
```

Hasta este punto de tienen tres conjuntos de datos **dataset**, **dataset.two** y **dataset-five**. El algoritmo **K-Medias** funciona utilizando medidas de distancia, portivo por el cual es necesario el escalamiento de los valores de las columnas predictoras dentro de un mismo rando de valores, de lo contrario el rendimiento del algoritmo se verá deteriorado por la diferencia entre las escalas. La función **ScaleSet** lleva todas las columnas a un rango de valores con *media* cero (0), y *desviación estándard* uno (1). Adicionalmente, del conjunto de datos **dataset** se eliminará la columna **Label_Normal_or_Attack** debido a que ya no es necesaria.

```
dataset$Label_Normal_or_Attack = NULL
dataset = ScaleSet(dataset)
dataset.two = ScaleSet(dataset.two)
dataset.five = ScaleSet(dataset.five)
```

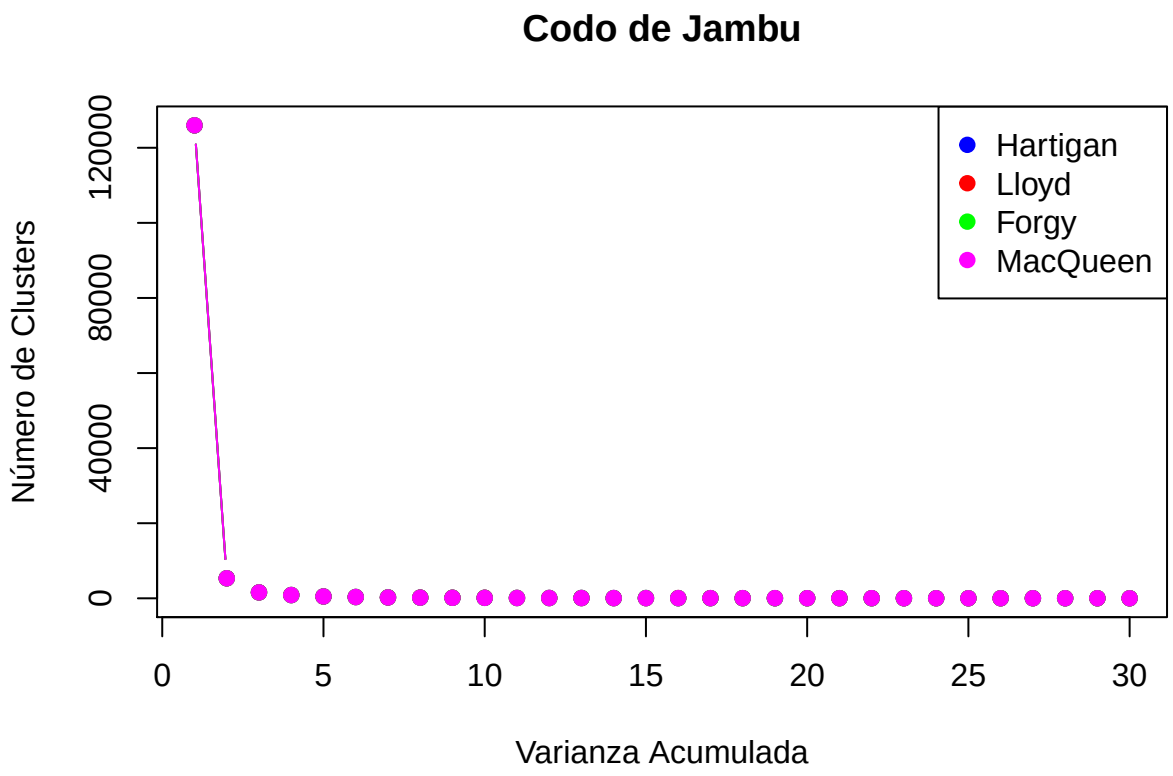
Codo de Jambu

Hasta este punto ya se tienen los conjuntos de datos listos para ser utilizados. El algoritmo **K-Medias** amerita que se le pasen cómo argumentos el número de centroides o la posición inicial de los centroides. Estos corresponden al número de conjuntos que se esperan identificar en el conjunto de datos. En el escenario que tenemos actualmente esta tarea es sencilla debido a que el conjunto de datos tiene cada uno de los registros etiquetados, sim embargo, este paso no siempre es sencillo. Adicionalmente es importante recordar que el algoritmo de **K-Medias** es un algoritmo de enfoque **no-supervisado** y no hace uso de estas etiquetas para separar los conjuntos.

Si bien es cierto que tenemos etiquetas que nos dicen de antemano cuáles son los niveles de los conjuntos, existe un dilema con respecto al rendimiento del algoritmo con **dos** o **cinco** clases objetivo. El problema de la selección del número de clusters siempre ha existido y existe un método llamado **Codo de Jambu** que es utilizado para capturar la varianza acumulada con respecto al número de clusters a usados. Al graficar la cantidad de varianza acumulada por el número de clusters, se verá que llega un punto en el que la gráfica tiene un comportamiento que emula la articulación de un codo, y es en ese punto, donde se empieza a formar la articulación donde se indica que el uso de mayor cantidad de clusters no aporta al algoritmo.

Existen 4 tipos de algoritmos para calculas las distancias de K-Medias, estas son **Hartigan-Wong**, **Lloyd**, **Forgy** y **Macqueen**. A continuación se aplicará cada una de estas técnicas varianzo la cantidad de centrpides en el rango [1,30], se graficarán los resultados y se analizarán los mismos.

```
IIC.Hartigan = vector(mode = "numeric", length = 30)
IIC.Lloyd = vector(mode = "numeric", length = 30)
IIC.Forgy = vector(mode = "numeric", length = 30)
IIC.MacQueen = vector(mode = "numeric", length = 30)
for (k in 1:30)
{
  groups = kmeans(dataset[,ncol(dataset)-2], k, iter.max = 100,
                  algorithm = "Hartigan-Wong")
  IIC.Hartigan[k] = groups$tot.withinss
  groups = kmeans(dataset[,ncol(dataset)-2], k, iter.max = 100, algorithm = "Lloyd")
  IIC.Lloyd[k] = groups$tot.withinss
  groups = kmeans(dataset[,ncol(dataset)-2], k, iter.max = 100, algorithm = "Forgy")
  IIC.Forgy[k] = groups$tot.withinss
  groups = kmeans(dataset[,ncol(dataset)-2], k, iter.max = 100, algorithm = "MacQueen")
  IIC.MacQueen[k] = groups$tot.withinss
}
plot(IIC.Hartigan, col = "blue", type = "b", pch = 19, main = "Codo de Jambu",
     xlab = "Varianza Acumulada", ylab = "Número de Clusters")
points(IIC.Lloyd, col = "red", type = "b", pch = 19)
points(IIC.Forgy, col = "green", type = "b", pch = 19)
points(IIC.MacQueen, col = "magenta", type = "b", pch= 19)
legend("topright", legend = c("Hartigan", "Lloyd", "Forgy", "MacQueen"),
       col = c("blue","red", "green", "magenta"), pch = 19)
```



Se puede observar cómo con dos clusters se alcanza el mejor resultado, debido a que cómo se observa en el gráfico, la transición entre la varianza acumulada con dos y tres clusters hace la analogía del codo que corresponde a la articulación mencionada con anterioridad. Adicionalmente se pued eobservar que todos los algoritmos se solapan entre si, este comportamiento es indicativo de que ambos se comportan de manera similar y es indiferente su uso en este conjunto de datos. Para mayor información consultar el siguiente enlace: Codo de Jambu.

En las próximas secciones se probará si estos resultados son ciertos evaluando el rendimiento del algoritmo utilizando 5 clusters y 2 clusters.

K-Medias (5 clusters)

Empezaremos con 5 clusters, debido a que aparentemente es el que tiene peor rendimeinto. La metodología es la siguiente, se aplicará 10 veces el algoritmo y se promediará la tasa de acierto para evaluar el desempeño.

```
results.five = vector(mode = "numeric", length = 10)
best.accuracy.five = 0
for (i in 1:length(results.five))
{
  set.seed(i)
  model.kmeans.five = kmeans(dataset.five[,1:(ncol(dataset.five)-1)],
                             5, iter.max = 100)

  prediction.five = OrderKmeans(model.kmeans.five)
  accuracy.five = mean(prediction.five == dataset.five$Label)

  results.five[i] = accuracy.five

  if(best.accuracy.five < accuracy.five)
```

```
  {
    best.prediction.five = prediction.five
    best.accuracy.five = accuracy.five
  }
}
```

La variable **results.five** contiene los resultados de la tasa de aciertos de cada iteración, y las variables **best.prediction.five** y **best.accuracy.five** contienen las mejores predicciones y la mejor tasa de aciertos respectivamente. Veamos los resultados.

```
results.five * 100
```

```
##  [1] 69.87688 68.85364 69.35454 77.91590 68.58771 72.29724 76.55132
##  [8] 59.90252 78.60891 76.77280
```

```
mean(results.five) * 100
```

```
## [1] 71.87215
```

Se observa que el promedio de acierto fue de 71, 87%, este rendimiento no parece estar mal, sin embargo, es necesario esperar a la comparación con el modelo de dos clusters para poder tener una mejor opinión. Mientras tanto crearemos una matriz de confusión para ilustrar el desempeño del modelo de manera gráfica.

```
confusion.matrix.five = table(Real = dataset.five$Label, Prediction = best.prediction.five)
confusion.matrix.five
```

```
##          Prediction
## Real        DoS normal Probing   R2L   U2R
##   DoS     34329   4691    6888     9    10
##   normal    115  63828     119   425  2856
##   Probing   384   5845     869  4217   341
##   R2L         3    940       0     0    52
##   U2R         0     52       0     0     0
```

La matriz de confusión se ve bastante desordenada, y no acertó en la predicción de ningún registro para las clases **R2L** y **U2R**. Veamos la mejor tasa se acierto y la peor tasa de aciertos.

```
best.accuracy.five*100
```

```
## [1] 78.60891
```

```
best.accuracy.five*100
```

```
## [1] 78.60891
```

La mejor tasa de aciertos fue de 78%, no parece ser un mal rendimiento, pero debemos esperar a la comparación con el otro modelo. Como aspecto importante a resaltar, la diferencia entre los resultados se debe a que la inicialización de los centroides en el algoritmo de **K-Medias** se hace de forma aleatoria, y dependiendo de la posición iniciales de los centroides, el algoritmo puede converger a diferentes mínimos locales. Por tal motivo, se colocaron semillas, de tal manera que las pruebas puedan ser recreables. A continuación calcularemos la eficacia por etiqueta, la salida es un vector numérico que representa a las clases ordenadas en orden alfabético de la siguiente forma: **DoS**, **normal**, **Probing**, **R2L** y **U2R**.

```
AccuracyPerLabel(confusion.matrix.five, dataset.five)
```

```
## [1] 74.746881 94.780452  7.455388  0.000000  0.000000
```

Se aprecia el hecho de que el algoritmo sólo clasifica bien las clases **DoS** y **Normal**. Estas dos clases corresponden a la mayoría de registros del conjunto de datos y por eso es que el promedio de aciertos es elevado, sin embargo, el rendimiento con las otras tres etiquetas **Probing**, **R2L** y **U2R** es pobre.

Lo siguiente será transformar la matriz de confusión de **cinco clases** a **dos clases**. Esto con la finalidad de poder calcular medidas de rendimiento binarias como lo son **sensitividad**, **especificidad** y **precisión**.

```
attack.normal.confusion.matrix.five = AttackNormalConfusionMatrix(dataset.five,
                                                                  best.prediction.five)
attack.normal.confusion.matrix.five
```

```
##         Prediction
## Real     Attack normal
##   Attack  47102  11528
##   normal   3515  63828
```

Se observa cómo hay una gran cantidad de **falsos negativos** y **falsos positivos**, a pesar de que la mayoría de los registros son clasificados de buena manera. Ahora veamos la eficacia por etiqueta. La salida corresponde a un vector numérico donde la primera posición es **Attack** y la segunda **normal**.

```
AccuracyPerLabel(attack.normal.confusion.matrix.five, dataset.two)
```

```
## [1] 80.33771 94.78045
```

La eficacia a la hora de detectar tráfico normal es bastante elevada, de 94.78%. Para la detección de los ataques es menor, esta corresponde a 80.34%, que es un número elevado, sin embargo, hayq ue recordar que este número está sesgado desde el punto de vista que sólo se clasificaron ataques de tipo **DoS**. A continuación se calcularán las medidas de rendimiento binarias.

```
Sensitivity(attack.normal.confusion.matrix.five) * 100
```

```
## [1] 80.33771
```

```
Especificity(attack.normal.confusion.matrix.five) * 100
```

```
## [1] 94.78045
```

```
Precision(attack.normal.confusion.matrix.five) * 100
```

```
## [1] 93.05569
```

Se observa que el modelo es bueno detectando tráfico normal, sin embargo, a la hora de detectar ataques el rendimiento se ve mermado.

K-Medias (2 clusters)

Ahora se implementarán los pasos realizados con cinco clusters pero ahora con dos clusters. Es decir, se correrá el algoritmo de **K-Medias** diez veces con dos clusters.

```
results.two = vector(mode = "numeric", length = 10)
best.accuracy.two = 0
for (i in 1:length(results.two))
{
  set.seed(i)
  model.kmeans.two = kmeans(dataset.two[,1:(ncol(dataset.two)-1)],
                            2, iter.max = 100)

  prediction.two = OrderKmeans(model.kmeans.two)
  accuracy.two = mean(prediction.two == dataset.two$Label)

  results.two[i] = accuracy.two

  if(best.accuracy.two < accuracy.two)
  {
    best.prediction.two = prediction.two
    best.accuracy.two = accuracy.two
  }
}
```

La variable **resulst.two** contiene la tasa de aciertos en cada iteración del algoritmo.

```
results.two
```

```
##  [1] 0.9050590 0.9050590 0.9050590 0.9050590 0.8103244 0.6087177 0.4935740
##  [8] 0.6087177 0.6087177 0.9050590
```

Se observa que la tasa de aciertos es mayor que con cinco clusters, adicionalmente, hubo iteraciones en la que los resultados se repitieron. Esto es debido a que la inicialización de los centroides al inicio del algoritmo hizo que en esas ocasiones se alcazara el mismo **mínimo local**. Este comportamiento da indicios de que la solución al conjunto de datos se representa con dos clusters y no con cinco. A continuación calculemos el promedio de acierto.

```
mean(results.two) * 100
```

```
## [1] 76.55347
```

La media de acierto es de 76.55%, este resultado es mayor al promedio con cincos clusters. Se creará una matriz de confusión para visualizar gráficamente el desempeño del algoritmo en la tarea de clasificación.

```
confusion.matrix.two = table(Real = dataset.two$Label, Prediction = best.prediction.two)
confusion.matrix.two
```

```
##         Prediction
## Real     Attack normal
##   Attack  47351  11279
##   normal    681  66662
```

Se aprecia que contiene una alto número de falsos negativos, en realidad es una cantidad similar a la matriz de confusión con cinco clusters. Por otra parte, la cantidad de falsos positivos se vio reducida notablemente. Ahora imprimiremos la tasa de aciertos y la tasa de error del mejor modelo obtenido.

```
best.accuracy.two*100
```

```
## [1] 90.5059
```

```
ErrorRate(best.accuracy.two)*100
```

```
## [1] 9.494098
```

Se obtuvo una tasa de aciertos de 90.51% una tasa bastante alta, mucho mayor que el modelo con cinco clusters. Por consecuente, la tasa de errores será también menor. Ahora veamos la tasa de aciertos por etiquetas. Es importante recordar que la salida corresponde a un vector numérico donde la primera posición corresponde a la etiqueta **Attack** y la segunda a la etiqueta **normal**.

```
AccuracyPerLabel(confusion.matrix.two, dataset.two)
```

```
## [1] 80.76241 98.98876
```

Se obtuvo una eficacia similar en la detección de ataques que en La evaluación con cinco clusters. La verdadera mejora fino en la eficacia de a la hora de clasificar el tráfico normal, donde se obtuvo un 98.99% de acierto. En esta oportunidad, la matriz de confusión ya es binaria, por consecuente se pueden calcular las medidas de rendimiento correspondientes.

```
Sensitivity(confusion.matrix.two) * 100
```

```
## [1] 80.76241
```

```
Especificity(confusion.matrix.two) * 100
```

```
## [1] 98.98876
```

```
Precision(confusion.matrix.two) * 100
```

```
## [1] 98.5822
```

La sensitividad nos dice que el modelo fue muy clasificando el **tráfico normal**, por otra parte clasificando los **ataques** es menos efectivo. La medida se sensitividad con respecto al modelo con cinco clusters se vio incrementado en 4%, por otra parte, la especificidad y la precisión es bastante parecida en ambos modelos.

Conclusión

En general ambos modelos tienen altos valores de eficacia, sin embargo, el modelo con dos clusters obtuvo mejores resultados con respecto a la **eficacia** y **sensitividad** de manera notable, y algunas mejoras simples en las medidas de especificidad y precisión. Adicionalmente, se observó que la cantidad de falsos positivos fue reducida en el modelo con dos clusters. Finalmente, el algoritmo **Codo de Jambu** es un buen método para la preselección de clusters a la hora de aplicar **K-Medias**.

Redes Neuronales

En esta sección se describirán las actividades realizadas para el entrenamiento y evaluación de redes neuronales en el ámbito de detección de intrusos en redes de computadoras. Este sección de subdivide en dos grandes partes **Entrenamiento** y **Evaluación**. Esto debido a que los pasos y observaciones serán realizadas de manera individual.

Entrenamiento

Para el entrenamiento de la red neuronal se propone una arquitectura con cuarenta neuronas en la **capa de entrada**, una **capa intermedia** con veinte neuronas, y cinco neuronas para la **capa de salida**. La razón para la selección de la arquitectura descrita previamente corresponde a que inicialmente se tienen cuarenta **variables predictoras** que corresponden a la capa de entrada; por otra parte, se tienen cinco clases objetivo que corresponden a las cinco neuronas de la **capa de salida**. El número de capas intermedias y el número de neuronas por capas que debe poseer una red neuronal no está ormado en ningún lugar, sólo existen recomendaciones hechas por expertos. **Andrew Ng** profesor de la *Universidad de Stanford*, menciona en uno de sus cursos de aprendizaje automático en **Coursera** que un modelo con una capa intermedia es suficiente para resolver una gran cantidad de problemas, adicionalmente comenta que en caso de querer utilizar una segunda capa intermedia, es **recomendable** que ambas capas posean igual cantidad de neuronas.

En este modelo se utilizaron 20 neuronas debido a que al haber una gran cantidad de neuronas de entrada, entonces la cantidad de neuronas intermedias de la mitad de las neuronas de entrada parece suficiente. El paque utilizado para la implementación de redes neuronales se llama **nnet**. Se probó otro paquete llamado **neuralnet**, a diferencia de **nnet**, **neuralnet** permite crear modelos con múltiples capas intermedias, pero es mucho más lento y para las tareas de clasificación hace el proceso más engorroso. Por otra parte, **nnet** a pesar de que sólo permite hacer uso de una capa intermedia, es mucho más rápido para el entrenamiento y el proceso de preparación de los datos para ser pasados cómo parámetros es más directo.

Se empezará por establecer el ambiente de trabajo eliminando variables parciales, cargando el archivo de funciones y la vista minable del conjunto de entrenamiento.

```
rm(list = ls())
dataset.training = read.csv("../dataset/NSLKDD_Training_New.csv",
                            sep = ",", header = TRUE)
source("../source/functions/functions.R")
```

Es importante mencionar que en esta sección se hará el entrenamiento y la evaluación del modelo haciendo uso únicamente del **conjunto de entrenamiento** haciendo uso de la técnica de validación de modelos llamada **validación cruzada de 10 conjuntos**.

Cómo se mencionó previamente el paquete utilizado es **nnet**, a continuación se procederá a instalarlo y a cargarlo en el ámbiente de trabajo.

```
install.packages("nnet")
library("nnet")
```

Una vez que tenemos nuestro ambiente de trabajo preparado se eliminarán aquellas etiquetas del conjunto de datos que no van a ser utilizadas a lo largo del proceso de entrenamiento del modelo. Este nivel posee cinco clases objetivos que son **DoS**, **normal**, **Probing**, **R2L** y **U2R**, esto con la finalidad de que la salida para el especialista sea más entendible y pueda identificar la falla se seguridad acontándola dentro de estas cuatro clase de ataques. Dicho esto eliminaremos el resto de las etiquetas.

```
dataset = dataset.training
dataset$Label_Normal_TypeAttack = NULL
dataset$Label_Num_Classifiers = NULL
dataset$Label_Normal_or_Attack = NULL
```

Es obligatorio para el uso de las redes neuronales que todas las variables predictoras sean de tipo **numérico**. Por tal motivo, las columnas serán transformadas a tipo numérico. Adicionalmente, cómo haremos tareas de clasificación, se establecerá la columna objetivo como tipo **factor**.

```
for (i in 1 : (ncol(dataset) -1) )
  dataset[,i] = as.numeric(dataset[,i])

dataset[,ncol(dataset)] = as.factor(dataset[,ncol(dataset)])
```

Para reducir el tiempo de entrenamiento y mejorar la precisión de las predicciones es buena práctica escalar el conjunto de datos a valores que posean *media* cero (0) y *desviación estándar* uno (1).

```
dataset = ScaleSet(dataset)
```

La estrategia adoptada para la evaluación del modelo será la utilización de **validación cruzada de 10 conjuntos**. La función **CVSet** toma un conjunto de datos y establece 10 divisiones de igual longitud del conjunto de datos y las devuelve en una **lista de dataframes**.

```
cv.sets = CVSet(dataset, k = 10, seed = 22)
length(cv.sets)
```

```
## [1] 10
```

Se observa que la longitud de la lista es de diez posiciones, esto debido a que en cada posición se posee un dataframe que corresponde a un subconjunto del conjunto de datos original. Todos los registros entre los diferentes dataframes son diferentes debido a que el muestreo se hizo sin reemplazo. Para seguir con las tareas se procederá a inicializar algunas variables.

```
results = vector(mode = "numeric", length = 10)
list.results = list(0, 0, 0, 0)
names(list.results) = c("results", "best_model", "best_testing_set", "best_predictions")
best.accuracy = 0
```

El proceso de entrenamiento y de validación del modelo es bastante largo y por esto se almacenarán en una lista los **resultados** correspondiendes a la eficacia de cada modelo en cada iteración, el **mejor modelo**, el **conjunto de datos de prueba** que originó la predicción, y el mejor conjunto de **predicciones**. De esta manera la lista puede ser guardada como un objeto y ser esportada a un archivo que posteriormente puede cargarse y no es necesario esperar a la realización de este paso cada vez que se desee analizar los reusltados. El siguiente fragmento de código es el encargado de realizar las tareas mencionadas con anterioridad.

```
for (i in 1:10)
{
  #Extrayendo el conjunto de datos
  testingset = as.data.frame(cv.sets[[i]])
  trainingset = cv.sets
  trainingset[[i]] = NULL
  trainingset = do.call(rbind, trainingset)

  #Entrenamiento de la red neuronal
  model = nnet(Label ~ .,
                      data = trainingset,
```

```
                    size = 20,
                    maxit = 100)

  #Realizando las predicciones
  predictions = predict(model, testingset[, 1:(ncol(testingset)-1)], type = "class")


  #Calculando la tasa de aciertos
  accuracy = mean(testingset[, ncol(testingset)] == predictions)

  #Almacenando el resultado
  results[i] = accuracy

  #Almacenando el mejor resultado
  if(best.accuracy < accuracy)
  {
    list.results$best_model = model
    list.results$best_testing_set = testingset
    list.results$best_predictions = predictions
    best.accuracy = accuracy
  }
}
```

Una vez que se culmina el proceso de entrenamiento, se almacena en la lista los resultados parciales de cada iteración y se exporta el modelo.

```
list.results$results = results
saveRDS(list.results, "../source/normal_model/NN/Tests/list_results.rds")
```

Evaluación