Detector de Botines

Sergio Rosello

Instalación y configuración de entorno

Mi idea inicial era hacer la práctica en la máquina virtual de Ubuntu. Así que he procedido a hacer eso. El problema con el que me he topado es que me he quedado sin espacio en el disco, así que he tenido que volver a configurar todo el entorno en mi sistema operativo host.

Pasos iniciales

Se ha creado un `virtualenv` con el que gestionar todos los módulos extras que necesita esta práctica. (`pandas`, `sklearn`) Al acabar, se ha descargado el dataset del caso 8 exclusivamente, ya que no me cabía en el ordenador todo el dataset entero.

Realización de la práctica

Al inicio, revisamos que el dataset esté correcto y además revisamos las cabeceras que tiene. En nuestro caso, se pueden observar en la imagen inferior.

```
>>> import pandas as pd
>>> data = pd.read_csv("../../Dataset/CTU-13-Dataset/8/capture20110816-3.binetflow")
>>> data['Label'] = data.Label.str.contains("Botnet")
>>> data.columns
Index(['StartTime', 'Dur', 'Proto', 'SrcAddr', 'Sport', 'Dir', 'DstAddr',
       'Dport', 'State', 'sTos', 'dTos', 'TotPkts', 'TotBytes', 'SrcBytes',
       'Label'],
      dtype='object')
>>>
```

Figure 1: Import Pandas

Para cargar el archivo `capture20110816-3.binetflow` tenemos que añadirlo al main de `LoadData.py` de la siguiente forma:

```
if __name__ == "__main__":
    loaddata('../../CTU-13-Dataset/9/capture20110817.binetflow')
```

Esto nos carga los datos del dataset y nos los guarda formateados ya en el archivo `flowdata.pickle`.

A partir de aquí, cargamos el archivo binario en la consola de python, importamos todos los módulos requeridos para analizar el tráfico de red, preparamos los datos para su análisis y empezamos con ello:

Decision tree model Aunque se ha separado el dataset en datos de prueba y datos para generar el modelo, se ha generado el modelo con los datos del modelo

```
>>> Xdata
array([[9.7656000e-01, 0.0000000e+00, 2.2540000e+03, ..., 4.0000000e+00,
        2.4400000e+02, 6.1105000e+04],
       [9.5066800e-01, 0.0000000e+00, 6.4136000e+04, ..., 4.0000000e+00,
        2.4400000e+02, 6.1105000e+04],
       [1.0069080e+00, 0.0000000e+00, 3.8440000e+03, ..., 4.0000000e+00,
        2.7600000e+02, 6.1105000e+04],
       ...,
       [1.1748079e+01, 0.0000000e+00, 2.3400000e+03, ..., 1.0000000e+01,
        1.3260000e+03, 3.5460000e+04],
       [4.6680000e-02, 1.0000000e+00, 1.5970000e+03, ..., 2.0000000e+00,
        3.2300000e+02, 2.1905070e+06],
       [8.9325760e+00, 0.0000000e+00, 1.5500000e+03, ..., 3.0000000e+00,
        1.8600000e+02, 6.8710000e+04]])
>>> XdataT
array([[1.880000e-04, 1.000000e+00, 5.932700e+04, ..., 2.000000e+00,
        2.140000e+02, 2.190507e+06],
       [2.380000e-04, 1.000000e+00, 3.665900e+04, ..., 2.000000e+00,
        2.140000e+02, 2.190507e+06],
       [2.630000e-04, 1.000000e+00, 4.033900e+04, ..., 2.000000e+00,
        2.140000e+02, 2.190507e+06],
       ...,
       [0.000000e+00, 0.000000e+00, 3.575000e+03, ..., 1.000000e+00,
        6.200000e+01, 6.871000e+04],
       [1.473169e+00, 0.000000e+00, 2.679000e+03, ..., 2.500000e+01,
        1.305600e+04, 2.163410e+05],
       [4.810000e-04, 1.000000e+00, 1.491000e+03, ..., 2.000000e+00,
        3.170000e+02, 2.190507e+06]])
```

Figure 2: Xdata, XdataT

```
>>> from sklearn.linear_model import *
>>> from sklearn.tree import *
>>> from sklearn.naive_bayes import *
>>> from sklearn.neighbors import *
>>> DataPreparation.Prepare(Xdata, Ydata, XdataT, YdataT)
<Prepare(Thread-1, initial)>
```

Figure 3: Import modules y prepare

y luego se han revisado con los datos de prueba para ver si efectivamente analiza bien los datos, se puede llegar a razonar que estos datos están sobreaprendidos.

Para validar el resultado, cabría la posibilidad de revisar los datos con otro sector del dataset original, ya que hemos consumido únicamente aproximadamente 30000 entradas de las casi tres millones de entradas del dataset.

Aún así, de acuerdo a lo que he mencionado anteriormente, esto es simplemente un paso de cautela, ya que se han seguido las buenas prácticas de generación y revisión de modelos.

```
<Prepare(Thread-1, initial)>
>>> clf = DecisionTreeClassifier()
>>> clf.fit(Xdata, Ydata)
DecisionTreeClassifier()
>>> Prediction = clf.predict(XdataT)
>>> Score = clf.score(XdataT, YdataT)
>>> print("El resultado de la clasificación con árbol de decisiones es: ", Score * 100)
El resultado de la clasificación con árbol de decisiones es:  99.89001099890011
```

Figure 4: Decision tree model

Las ventajas de los árboles de decisiones son:

- No requiere la normalización de datos
- No requiere el escalado de información
- Los datos se pueden usar sin estar validados, ya que no afectan demasiado el resultado de árbol
- Modelo intuitivo y sencillo de explicar

Los inconvenientes de los árboles de decisión son:

- Pueden producir árboles excesivamente complejos (sobreaprendizaje)
- A mayor número de datos, mayor complejidad (Deriva en sobreaprendizaje)
- Los puntos de decisión se basan en los "padres", no en el conjunto global de datos
- Pequeñas variaciones en los datos pueden crear árboles completamente distintos

Logistic regression model Este modelo es una extensión de algoritmo de regresión lineal, principalmente usado en estadística, pero extendido para usar la función sigmoide.

A diferencia de una función lineal, esta nos permite variar su forma para adaptarse a los datos.

Las ventajas de los modelos de regression logística son:

- Si se configuran bien, producen resultados excelentes
- Se puede evaluar de forma visual los resultados con la curva ROC (Receiver Operating Characteristics) y llegar a un análisis mejor si se interpreta dicha curva y varían los datos de configuración del algoritmo

```
>>> clf = LogisticRegression(C=10000)
>>> clf.fit(Xdata, Ydata)
LogisticRegression(C=10000)
>>> Prediction = clf.predict(XdataT)
>>> Score = clf.score(XdataT, YdataT)
>>> print("El resultado de la clasificación con Logistic regression es: ", Score * 100)
El resultado de la clasificación con Logistic regression es:  96.25037496250374
```

Figure 5: Logistic Regression

Las desventajas de los modelos de regression logística son:

- Para producir resultados excelentes, los datos a analizar tienen que ser linealmente clasificables
- La clasificación es más complicada que en otros algoritmos. Se tienen que tener en cuenta los "outliers" ya que pueden producir resultados poco optimizados

Gaussian Naive Bayes model El teorema de Bayes calcula la probabilidad de que ocurra B cuando A es verdadero. En un problema de clasificación, el teorema de Bayes nos permite calcular la probabilidad condicional de que una muestra pertenece a una clase en concreto dados sus atributos.

El problema con el teorema de Bayes es que calcula las relaciones probabilísticas entre todos lo atributos en un dataset.

El teorema Naive Bayes simplifica el proceso al asumir independencia entre clases. Esto quiere decir que no se calculará la probabilidad de los atributos entre clases.

```
>>> clf = GaussianNB()
>>> clf.fit(Xdata, Ydata)
GaussianNB()
>>> Prediction = clf.predict(XdataT)
>>> Score = clf.score(XdataT, YdataT)
>>> print("El resultado de la clasificación con Gaussian Naive Bayes es: ", Score * 100)
El resultado de la clasificación con Gaussian Naive Bayes es:  67.83321667833216
```

Figure 6: Gaussian Naive Bayes

Las ventajas de los modelos de regression logística son:

- Sencillo de implementar
- Escalable
- Sirve tanto para clasificación binaria o multi-clase

Las desventajas de los modelos de regression logística son:

- Asume que los atributos son condicionalmente independientes
- Necesita bastantes datos para poder calcular las combinaciones clase/atributos que existen en el dataset.

K-nearest Neighbors model Es un algoritmo de clasificación que discrimina los datos según la distancia cada uno de los mismos a cada uno de los puntos predefinidos. Posteriormente, agrega el punto analizado al punto predefinido más cercano.

```
>>> clf = KNeighborsClassifier()
>>> clf.fit(Xdata, Ydata)
KNeighborsClassifier()
>>> Prediction = clf.predict(XdataT)
>>> Score = clf.score(XdataT, YdataT)
>>> print("El resultado de la clasificación con K-Nearest Neghbors es: ", Score * 100)
El resultado de la clasificación con K-Nearest Neghbors es:  96.84031596840316
```

Figure 7: K-Nearest Neighbors

Las ventajas de los modelos de regression logística son:

- Algoritmo sencillo de implementar

Las desventajas de los modelos de regression logística son:

- Computacionalmente exigente

Neural Network model La particularidad de las redes neuronales es que a partir de una configuración inicial, ellas van generando sus configuraciones, para "Aprender". Poco a poco, van evolucionando y ajustando sus pesos con distintas técnicas, como backpropagation.

Para poder calcular el modelo probabilístico de la red neuronal, se han importado los módulos `TensorFlow` y `keras`.

Más adelante, se han seguido los siguientes comandos (Sin captura de pantalla porque la salida de la red neuronal evita que se vean correctamente) y se ha llegado al peor resultado de todos los modelos probabilísticos.

```
from keras.models import *
from keras.layers import Dense, Activation
from keras.optimizers import *


model = Sequential()
model.add(Dense(10, input_dim=9, activation="sigmoid"))
model.add(Dense(10, activation='sigmoid'))
model.add(Dense(1))
sgd = SGD(lr=0.01, decay=0.000001, momentum=0.9, nesterov=True)
model.compile(optimizer=sgd, loss='mse')
model.fit(Xdata, Ydata, nb_epoch=200, batch_size=100)
Score = model.evaluate(XdataT, YdataT, verbose=0)
print("El resultado de la clasificación con Red Neuronal es: ", Score * 100)
```

El resultado del modelo probabilístico ha dado `24.0171`.

Puede que añadir una capa oculta a la red neuronal ayude a la clasificación de la red.

Las ventajas de las redes neuronales son:

- Ajustando unos parámetros iniciales, ellas mismas se configuran para obtener el mejor resultado posible
- No solo sirven para clasificar datos, se pueden usar para una gran cantidad de propósitos

Las desventajas de las redes neuronales son:

- Una vez configurada la red neuronal, no podemos saber que hacen las capas ocultas.
- Son mas Computacionalmente caras que algunos otros modelos
- Si no se configuran correctamente, pueden llegar a sobreaprender

Problemas encontrados

En un principio, he tenido el problema de la máquina virtual, ya que al descargar el archivo de tamaño tan grande, se ha quedad sin memoria el disco lógico de Ubuntu. Esto ha provocado que, de alguna forma que no comprendo aún se rompa la máquina virtual.

Habiendo aprendido la lección, decidí hacer la práctica con la máquina virtual de Kali, que tenía dedicados `80GB` de almacenamiento virtual. Mientras se descargaba el Dataset, he empezado a actualizar el sistema, ya que era una máquina que hacía tiempo que tenía. A mediados de la descarga e instalación, se ha llenado el disco físico de mi ordenador, dejando la máquina virtual a medias. He tenido que apagar la máquina a mitad de instalación y descarga y esto ha provocado que se rompa.

Logré recuperar la máquina, pero estaba tan rota de el fallo de actualización y apagado a medias, que no tenía remedio, o al menos, no tenía remedio rápido.

A falta de espacio en mis disco físico, he descargado únicamente el contenido de la sección 8 desde la página web, pero intentando realizar la práctica con este contenido ha sido imposible, ya que no tenía todos los archivos que necesitaba.

Además, el archivo proporcionado en la fase 8 no cumple los requisitos necesarios por el script de carga de datos. A las 26127 líneas de análisis, el programa sale. Esto es porque los checks de calidad de los registros hacen que del archivo tan grande, solo queden 26127 líneas válidas. Esto hace que no llegue a generar los registros de datos de pruebas. Este problema lo he tenido tanto con el dataset 5 como con el dataset que viene de ejemplo en el código de GitHub.