Úvod do strojového učenia

Strojové učenie a umelá inteligencia patrí medzi aktuálne najviac sa rozvíjajúce oblasti vedy a techniky. Umelú inteligenciu je možné využívať v rôznych situáciách a doménach, medzi ktoré patria medicína, strojárstvo, priemysel, vzdelávanie, kyberbezpečnosť, informatika ale aj počítačové siete.

V rámci tohto blogu budú vysvetlené dôležité pojmy z tejto oblasti, základné delenie a vzorové zdrojové kódy. Pre lepšiu predstavu, komunite sieťového akademického programu NetAcad, bude využitie strojového učenia ukázané na klasifikačnej úlohe klasifikácie paketov na základe jeho charakteristík, pričom tá istá úloha bude riešená rôznymi typmi strojového učenia. Z pohľadu implementácie bude využitý programovací jazyk Python.

Delenie strojového učenia

Strojové učenie (množina algoritmov) je možné rozdeliť do troch hlavných kategórií a to:

- **Učenie s učiteľom** (angl. Supervised Learning)
- **Učenie bez učiteľa** (angl. Unsupervised Learning)
- **Učenie s posilňovaním** (angl. Reinforcement Learning)

Podkategóriou strojového učenia je tzv. **Hlboké učenie** (angl. Deep Learning), ktoré využíva neurónové siete (analogický prístup učenia, aký sa odohráva v ľudskom mozgu).

Nasledujúci zoznam znázorňuje vzájomný súvis uvedených pojmov:

- Umelá inteligencia
    - Strojové učenie
        - Hlboké učenie

Dátová sada

Strojové učenie vyžaduje dátovú sadu, ktorá bude slúžiť na trénovanie a testovanie modelu strojového učenia. Dátovú sadu si je možné predstaviť ako 2D pole, resp. tabuľku, ktorá obsahuje množinu riadkov (záznamy) a množinu stĺpcov (charakteristiky popisujúce daný záznam). Zdrojové dáta pred ich využitím v rámci strojového učenia je potrebné pred spracovať, kde medzi hlavné možnosti predspracovania patrí:

- **Imputácia** = vloženie/doplnenie chýbajúcich hodnôt. Štandardne sa chýbajúce hodnoty nahrádzajú priemernou alebo najčastejšie používanou hodnotou v danom stĺpci.

- **Kódovanie** = nahradenie reťazcov číselnými hodnotami.
- **Normalizácia / Škálovanie** = transformácia číselných hodnôt do malého rozsahu <0 , 1>.
- **Delenie** = rozdelenie dát na trénovaciu sadu a testovaciu sadu. Štandardne v pomere 80:20 / 70:30.

Dátová sada zvyčajne obsahuje X časť (2D pole obsahujúce záznamy a ich charakteristiky) a y časť (vektor obsahujúci označenia daných dát). V našom prípade klasifikačnej úlohy bude dátová sada vyzerať nasledovne:

```
Source,Destination,Protocol,Length
10.22.22.22,224.0.0.9,RIPv2,66
10.22.22.2,224.0.0.5,OSPF,94
10.22.22.22,224.0.0.10,EIGRP,74
1.1.1.1,22.22.22.22,ICMP,114
```

Celkový počet záznamov je 830. V praxi je potrebné mať podstatne väčšie dátové sady, no pre demonštračný účel tohto blogu je 830 postačujúce.

Ako si je možné všimnúť, tak dátová sada má nasledujúce stĺpce:

- Source = zdrojová adresa
- Destination = cieľová adresa
- Length = dĺžka paketu
- Protocol = značka

Jednotlivé riadky predstavujú záznamy - v našom prípade pakety a stĺpce predstavujú ich charakteristiky. **X** časť dátovej sady pozostáva zo stĺpcov **Source**, **Destination** a **Length**. **y** časť dátovej sady pozostáva z vektora (stĺpec) **Protocol**.

Predspracovanie dát bude vykonané nasledujúcim spôsobom:

1. Kódovanie stĺpcov Source, Destination a Protocol
2. Rozdelenie dát na trénovacie a testovacie
3. Normalizácia / Škálovanie stĺpca Length

Model strojového učenia

Strojové učenie je množina algoritmov, ktoré dokážu vytvoriť model, naučiť ho istému správaniu (trénovacia sada dát), vyhodnotiť správnosť správania (testovacia sada dát) a následne ho použiť na vyhodnocovanie novo získaných dát. Modelov strojového učenia existuje viacero, každý z modelov využíva na pozadí “inú matematiku” a algoritmy. Modely je možné aj ladiť úpravou hodnôt ich hyperparametrov.

Učenie s učiteľom

Učenie s učiteľom je založené na množine označených dát.

Scenár:

Majme dieťa, ktoré chceme naučiť rozoznávať ovocie. Učenie s učiteľom znamená, že mu postupne budeme ukazovať ovocie (objekt = množina atribútov - tvar, veľkosť, farba, …) a hneď mu povieme, čo za ovocie to je (značka = jablko, hruška, banán, …). Čím viac druhov ovocia ukážeme, tým lepšie bude dieťa rozoznávať dané ovocie. Ak mu však ukážeme nejaké ovocie, ktoré ešte nevidelo, tak nedokáže povedať, čo to je, ale len to, že to nie je žiadne z ovocí, ktoré videlo. V prípade strojového učenia bude výsledok nesprávnej klasifikácie dát.

V našom prípade budeme učiť model strojového učenia klasifikovať pakety (OSPF, RIP, EIGRP a ICMP) na základe ich charakteristík (Zdrojová adresa, Cieľová adresa a dĺžka paketu). Nasledujúci zdrojový kód znázorňuje ukážku učenia s učiteľom:

```
# import kniznice pre pracu s datami
import pandas as pd
import numpy as np

# nacitanie datovej sady
data = pd.read_csv('cls.csv')

# vytvorenie X dat (pouzivane na trenovanie) a y dat (predikovana
znacka)
X = data.drop('Protocol', axis=1)  # odstranenie stlpca Protocol a
ulozenie vyslednej datovej sady
y = data['Protocol'] # vlozenie stlpca Protocol do premennej y

# zobrazenie vytvorenych premennych
print(X)
print(y)

# importovanie modulu LabelEncoder
from sklearn.preprocessing import LabelEncoder

# vytvorenie objektu
c1 = LabelEncoder()
# trenovanie a transformacia stlpca source
X["Source"] = c1.fit_transform(X["Source"])

# vytvorenie objektu - trenovanie a transformacia stlpca
destination
c2 = LabelEncoder()
X["Destination"] = c2.fit_transform(X["Destination"])

# vytvorenie objektu - trenovanie a transformacia stlpca Protocol
c3 = LabelEncoder()
# prevedie y hodnoty na rozsah 0-3
```

```
y = c3.fit_transform(y)

# importovanie modulu pre rozdelenie dat
from sklearn.model_selection import train_test_split
# rozdelenie dat na trenovacie a testovacie
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=1)

# importovanie modulu pre skalovanie
from sklearn.preprocessing import MinMaxScaler
# preskalovanie dat dlzky paketu do rozsahu 0-1
scaler = MinMaxScaler()
X_train["Length"] = scaler.fit_transform(X_train[["Length"]])
X_test["Length"] = scaler.transform(X_test[["Length"]])

# importovanie modulu pre klasifikator
from sklearn.tree import DecisionTreeClassifier
# vytvorenie objektu klasifikatora
clf = DecisionTreeClassifier()

# trenovanie modelu na trenovacich datach
clf.fit(X_train, y_train)

# importovanie modulu pre odhad presnosti modelu
from sklearn.metrics import accuracy_score

# predikovanie znaciek pre testovacie data
y_pred = clf.predict(X_test)

# porovnanie predikovanych dat a realnych testovacich
accuracy = accuracy_score(y_test, y_pred)
print("Presnost natrenovaneho modelu je: {}".format(accuracy))


# transformacia dat noveho paketu
new_source = c1.transform(["10.22.22.2"]).reshape(-1, 1)
new_dest = c2.transform(["224.0.0.5"]).reshape(-1, 1)
new_length = scaler.transform([[94]]).reshape(-1, 1)
# vytvorenie noveho riadku, ktory bude klasifikovany neuronovou
sietou
new_data = np.concatenate((new_source, new_dest, new_length),
axis=1)

# predikcia na novych datach a zobrazenie znacky
new_predict = clf.predict(new_data)
print("Nove data patria protokolu s indexom:
{}".format(new_predict))

# spatna konverzia indexu predikcie na textovu podobu
predicted_protocol = c3.inverse_transform(np.array([new_predict]))
print("Nove data patria protokolu:
```

```
{}".format(predicted_protocol[0]))
```

Očakávaným výstupom programu je nasledujúci výpis:

```
Presnost natrenovaneho modelu je: 0.9919678714859438
Nove data patria protokolu s indexom: [2]
Nove data patria protokolu: OSPF
```

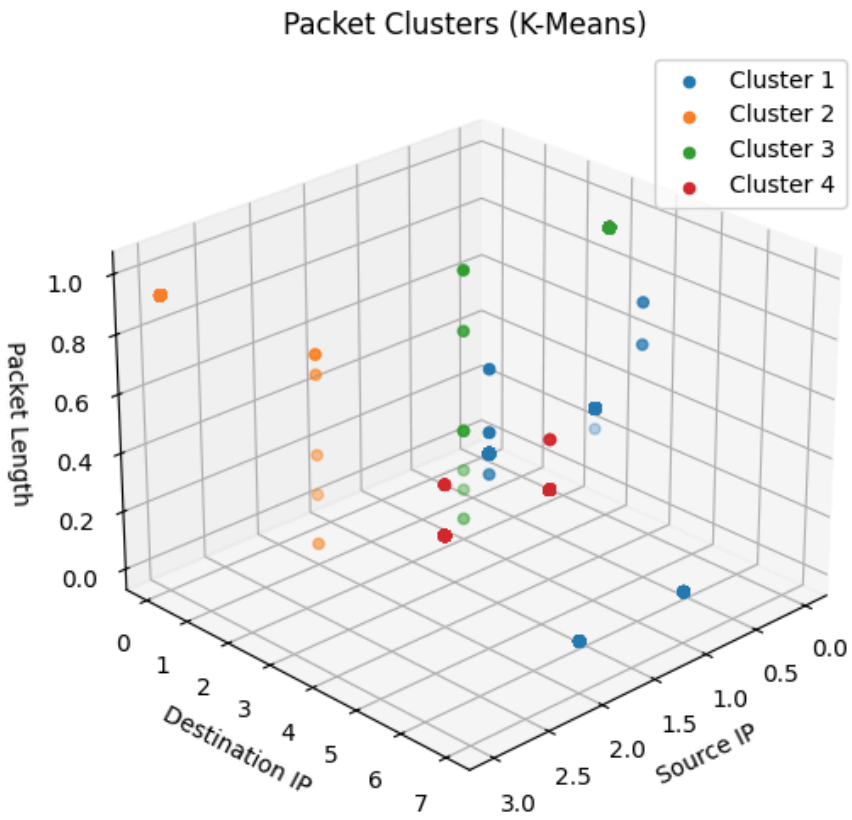
Učenie bez učiteľa

Učenie bez učiteľa je založené na množine neoznačených dát:

Scenár:

Majme dieťa, ktoré chceme naučiť rozoznávať ovocie. Učenie bez učiteľa znamená, že mu na stôl položíme niekoľko druhov ovocia (náhodne napr. 5 jablk, 6 hrušiek, …), pričom úlohou je teraz rozdeliť dané ovocie na kôpky, čím vytvoríme niekoľko zhlukov (klastrov).

Tento prístup umožňuje detegovať rôzne typy dát - v našom prípade vzniknú 4 kôpky (pre každý protokol jedna). Tento prístup má využitie napr. pri detekcii anomálií. V praxi, pri budovaní dátovej sady, sa využije tento prístup, následne sa vytvorené klastre manuálne vyhodnotia a priradí sa im značka. V prípade našej dátovej sady je možné vytvorené klastre vizualizovať, čo znázorňuje nasledujúci obrázok.



Zdrojový kód pre túto úlohu vyzerá nasledovne:

```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
import matplotlib.pyplot as plt

# nacitanie dat
data = pd.read_csv("cls.csv")

# vytvorenie datovej sady
features = ["Source", "Destination", "Length"]
X = data[features]

# kodovanie a skalovanie dat
c1 = LabelEncoder()
X["Source"] = c1.fit_transform(X["Source"])
c2 = LabelEncoder()
X["Destination"] = c2.fit_transform(X["Destination"])
scaler = MinMaxScaler()
X["Length"] = scaler.fit_transform(X[["Length"]])
```

```
# definovanie poctu ocakavanych clustrov
n_clusters = 4

#vykonaie K-means clustering-u
kmeans = KMeans(n_clusters = n_clusters)
kmeans.fit(X)

# ulozenie objavenych oznaceni clustrov do povodnych dat
X["Cluster"] = kmeans.labels_

# vykreslenie vytvorenych clustrov v 3D zobrazeni
fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(111, projection='3d')

# prechod cez vsetky clustre a zobrazenie ich hodnot roznymi
farbami
for cluster in range(n_clusters):
     cluster_data = X[X["Cluster"] == cluster]
     ax.scatter(cluster_data["Source"],
cluster_data["Destination"], cluster_data["Length"],
label=f"Cluster {cluster+1}")

# pridanie oznaceni - os x,y,z a nazvu grafu
ax.set_xlabel("Source IP")
ax.set_ylabel("Destination IP")
ax.set_zlabel("Packet Length")
plt.title("Packet Clusters (K-Means)")

# povolenie legendy a rotacie
plt.legend()
ax.view_init(elev=15, azim=60)

# zobrazenie grafu
plt.show()

# zobrazenie povodnych, upravenych dat a znacky clustru
final = data
final["src_encode"] = X["Source"]
final["dst_encode"] = X["Destination"]
final["length_encode"] = X["Length"]
final["cluster"] = X["Cluster"]
print(final)

# ulozenie hodnot do csv suboru
df1 = pd.DataFrame(final)
df1.to_csv('result.csv', index=False)
print("Done")
```

Očakávaný výstup programu je nasledovný:

	Source	Destination	Protocol	...	dst_encode	length_encode	cluster
0	10.22.22.22	224.0.0.9	RIPv2	...	7	0.103448	0
1	10.22.22.2	224.0.0.5	OSPF	...	5	0.586207	0
2	10.22.22.22	224.0.0.10	EIGRP	...	4	0.241379	3
3	10.22.22.2	224.0.0.5	OSPF	...	5	0.517241	0
4	10.22.22.22	224.0.0.5	OSPF	...	5	0.517241	0
..	...	...	...	...	...	...	...
825	10.22.22.2	224.0.0.5	OSPF	...	5	0.586207	0
826	10.22.22.22	224.0.0.9	RIPv2	...	7	0.103448	0
827	10.22.22.22	224.0.0.5	OSPF	...	5	0.586207	0
828	10.22.22.2	224.0.0.10	EIGRP	...	4	0.241379	3
829	10.22.22.22	224.0.0.10	EIGRP	...	4	0.241379	3

Učenie s posilňovaním

Učenie s posilňovaním je založené na existencii agenta, ktorý sa vloží do prostredia. Agent má k dispozícii množinu akcií, ktoré môže vykonať a hodnotu úžitku, ktorú po vykonaní akcie získa. Pričom platí, že pri použití vhodnej akcie dostane agent kladnú odmenu a v prípade menej efektívnej akcie dostane agent zápornú odmenu. Agent sa v niekoľkých iteráciách snaží upraviť svoje správanie tak, aby maximalizoval úžitok svojej aktivity a získal najväčšiu odmenu. Tento typ učenia je analogický s učením dieťaťa založenom na pochvale a pokarhaní.

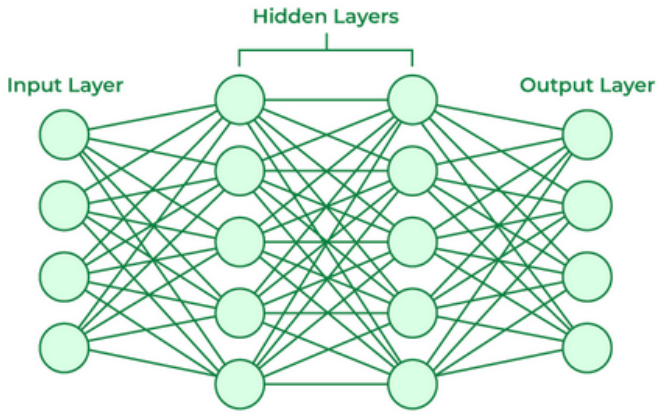
Učenie s posilňovaním umožňuje riešiť problémy bez nutnosti trénovacej sady dát. Na druhej strane vyžaduje množinu algoritmov a funkcii, ktore dokážu pozorovať správanie agenta a priradzovať mu "odmenu". Vzhľadom k uvedenému, je zdrojový kód podstatne zložitejší a preto jeho ukážka nie je súčasťou tohto blogu.

Neurónové siete

Neurónová sieť je univerzálny aproximátor funkcie, pričom pozostáva z 3 typov vrstiev a to:

- **Vstupná vrstva** = vstupné dáta - v našom prípade množina atribútov charakterizujúcich pakety (zdrojová adresa, cieľová adresa, dĺžka paketu) = potrebujeme 3 neuróny
- **Istý počet skrytých vrstiev** = množina neurónov a prepojení, ktoré slúžia na úpravu váh jednotlivých prepojení s cieľom získania správneho výstupu
- **Výstupná vrstva** = množina očakávaných značiek, pričom na základe vstupu dá neurónová sieť na výstup pravdepodobnosti, ktorá značka prislúcha daným vstupným dátam

Nasledujúci obrázok graficky znázorňuje prepojenia medzi jednotlivými vrstvami.



Pri trénovaní neurónovej siete je potrebné mať označenú dátovú sadu, pričom myšlienka je podobná ako v prípade učenia s učiteľom v štandardnom strojovom učení.

Zdrojový kód vytvorenia jednoduchej neurónovej siete je znázornený v nasledujúcom výpise:

```
from sklearn.model_selection import train_test_split
from tensorflow import keras
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
import numpy as np


def load_data(filename):
      data = pd.read_csv(filename)
      features = data.drop('Protocol', axis=1)
      labels = data['Protocol']
      return features, labels


# nacitanie dat
X, y = load_data("cls.csv")
# kodovanie X retazcov na cisla
c1 = LabelEncoder()
X["Source"] = c1.fit_transform(X["Source"])
c2 = LabelEncoder()
X["Destination"] = c2.fit_transform(X["Destination"])
c3 = LabelEncoder()
# prevedie y hodnoty na rozsah 0-3
y = c3.fit_transform(y)
# prevedie rozsah 0-3 na [0.0.0.1. - 1. 0. 0. 0.]
y = keras.utils.to_categorical(y, 4)

# rozdelenie dat na testovacie a trenovacie 80:20
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=1)
```

```
# preskalovanie dat dlzky paketu do rozsahu 0-1
scaler = MinMaxScaler()
X_train["Length"] = scaler.fit_transform(X_train[["Length"]])
X_test["Length"] = scaler.transform(X_test[["Length"]])
print(X_train)
print(y_train)

# vytvorenie modelu neuronovej siete - neuronova siet je
univerzalny aproximator funkcie
model = keras.Sequential([
     keras.layers.Input(shape=(3,)),  # vstupna vrstva pre
zdroj/ciel adresu a dlzku paketu = 3 neurony
     keras.layers.Dense(100, activation='leaky_relu'),  # mnozina
skrytych vrstiev
     keras.layers.Dropout(0.2),  # dropout vrstva pre predchadzanie
pretrenovania
     keras.layers.Dense(16, activation='relu'),
     keras.layers.Dense(32, activation='relu'),
     keras.layers.Dense(4, activation='softmax')  # vystupna vrstva
- softmax pre multi-class clasifikaciu = 4 neurony
])

# vytvorenie modelu
model.compile(loss="categorical_crossentropy", optimizer="adam",
metrics=["accuracy"])
# trenovanie modelu, kde batch_size = pocet nahodne zvolenych
prvkov (postupne iterovane v ramci 1 epochy X_size/batch_size)
# epochs = pocet kolko krat sa prejde celou datovou sadou
model.fit(X_train, y_train, epochs=10, batch_size=32)

# vyhodnotenie modelu
loss, accuracy = model.evaluate(X_test, y_test)
print("Loss:", loss)
print("Accuracy:", accuracy)

# transformacia dat noveho paketu
new_source = c1.transform(["10.22.22.2"]).reshape(-1, 1)
new_dest = c2.transform(["224.0.0.5"]).reshape(-1, 1)
new_length = scaler.transform([[94]]).reshape(-1, 1)
# vytvorenie noveho riadku, ktory bude klasifikovany neuronovou
sietou
new_data = np.concatenate((new_source, new_dest, new_length),
axis=1)

# predikovanie vyuzitim modelu - vrati pole 4 hodnot -
pravdepodobnost kazdeho vystupu
prediction = model.predict(new_data)
print("Prediction:", prediction)

# vratenie hodnoty indexu predikovanej znacky 0-3
```

```
(najpravdepodobnejsia znacka)
predicted_protocol = np.argmax(prediction[0])
print("Predicted protocol - index:", predicted_protocol)

# spatna konverzia indexu predikcie na textovu podobu
predicted_protocol =
c3.inverse_transform(np.array([predicted_protocol]))
print("Predicted protocol - string:", predicted_protocol[0])
```

Očakávaným výstupom programu je: