تمرین

اینترنت اشیا

استاد

محمد زارع

به کوشش

مهدی جوانشیر

تاریخ

آذر 1403

عنوان تمرین نرم افزاری: تمرین شناسایی حمله باRNN

برای شناسایی حملات سایبری و تحلیل رفتارهای نامتعارف از طریق شبکه‌های عصبی بازگشتی (RNN)به تمرین زیر بپردازید

هدف

شناسایی رفتارهای غیرطبیعی در ترافیک شبکه (شامل حملات مثلDDoS ، اسکن پورت و غیره) با استفاده از RNN.

مراحل انجام تمرین

1. جمع آوری داده ها

- از یک دیتاست مناسب برای ترافیک شبکه استفاده کنید. دیتاست‌هایی مانند Cup KDD 99 یا CICIDS2017 مناسب هستند.

- این دیتاست‌ها باید شامل ویژگی‌هایی مانند زمان، منبع، مقصد، نوع پروتکل و نشانگر حمله یا عدم حمله باشند.

2. پیش‌پردازش داده‌ها

- داده‌ها را پاک‌سازی کنید و ویژگی‌های لازم را انتخاب کنید.
- داده‌ها را به شکل زمان‌بندی آماده کنید. به عنوان مثال، هر 100 نمونه از ترافیک را به عنوان یک ورودی برای شبکه RNN در نظر بگیرید.
- داده‌ها را به دو بخش آموزش و تست تقسیم کنید.

3. طراحی مدل RNN

- از یک معماری ساده RNN یا LSTM (Long Short- Term Memory) استفاده کنید.
- ورودی به مدل باید شامل ویژگی‌های ترافیک به همراه برچسب حمله یا عدم حمله باشد.
- می‌توانید از کتابخانه‌های TensorFlow یا PyTorch برای پیاده‌سازی استفاده کنید.

4. آموزش مدل

- مدل را با داده‌های آموزشی آموزش دهید.
- از تکنیک‌های تنظیم هایپرپارامتر و بهینه‌سازی استفاده کنید تا دقت مدل را افزایش دهید.

5. ارزیابی مدل

- مدل را با داده‌های تست ارزیابی کنید.
- از معیارهای مختلفی مانند دقت، یادآوری، F1- score و ROC-AUC برای اندازه‌گیری عملکرد مدل استفاده کنید.

6. تحلیل نتایج

- نتایج را تحلیل کنید و بررسی کنید که کجا مدل به درستی یا نادرستی پیش‌بینی کرده است.
- می‌توانید مهمترین ویژگی‌ها را شناسایی کنید که به شناسایی حمله کمک کرده‌اند.

نکات اضافی

- برای بهبود دقت، می‌توانید از تکنیک‌های متنوعی مانند Dropout یا Regularization استفاده کنید.
- همچنین می‌توانید با استفاده از شبکه‌های عصبی پیچشی (CNN)به عنوان ورودی به RNN، ویژگی‌های فضایی را نیز استخراج کنید.

```
1  # وارد کردن کتابخانه‌های مورد نیاز
2  import pandas as pd  # برای پردازش داده‌ها به صورت ساختاریافته
3  import numpy as np  # برای عملیات عددی و کار با آرایه‌ها
4  import tensorflow as tf  # برای طراحی و اجرای مدل‌های یادگیری عمیق
5  from sklearn.preprocessing import StandardScaler  # برای نرمال‌سازی داده‌ها
6  from sklearn.model_selection import train_test_split  # برای تقسیم داده‌ها به مجموعه‌های آموزش و آزمایش
7  from sklearn.metrics import classification_report, accuracy_score  # برای ارزیابی مدل
8  import matplotlib.pyplot as plt  # برای رسم نمودارها
9  import dask.dataframe as dd  # برای پردازش موازی داده‌های بزرگ
10
```

کتاب خانه ها و ماژول ها

(pd) pandas

- برای مدیریت داده‌ها در قالب DataFrame که دسترسی و عملیات روی داده‌ها را ساده‌تر می‌کند.
- در این کد از pandas برای انجام تغییرات روی ستون‌های داده (مانند تبدیل برچسب‌ها و ویژگی‌ها به مقادیر عددی) استفاده شده است.

(np) numpy

- برای انجام عملیات عددی کارآمد، مانند تغییر شکل داده‌ها و بررسی وجود مقادیر NaN.
- آرایه‌های numpy به دلیل کارایی بالا در پردازش داده‌های عددی برای آماده‌سازی ورودی مدل‌ها استفاده می‌شوند.

(tf) tensorflow

- ساخت مدل RNN با لایه LSTM برای شناسایی الگوهای زمانی در داده‌ها.
- ابزارهای قدرتمندی برای طراحی لایه‌ها، کامپایل مدل، و آموزش آن فراهم می‌کند.

- sklearn.preprocessing.StandardScaler: برای نرمال‌سازی و استانداردسازی داده‌ها به کار می‌رود.
- sklearn.model_selection.train_test_split: تقسیم داده‌ها به مجموعه‌های آموزش و آزمایش.
- sklearn.metrics.classification_report و accuracy_score: برای ارائه معیارهای دقت، یادآوری (recall)، و F1-score. برای محاسبه درصد پیش‌بینی‌های صحیح.

(plt) matplotlib.pyplot

- رسم نمودار برای تحلیل و تجسم نتایج.
- برای نمایش گرافیکی تغییرات دقت مدل در طول epochها.

dask.dataframe

- داده‌های حجیم می‌توانند پردازش را کند کنند؛ dask امکان بارگذاری داده‌ها به صورت موازی و مدیریت منابع بهینه‌تر را فراهم می‌کند.

```
11  # مرحله 1: جمع‌آوری داده‌ها
12  print("مرحله 1: جمع‌آوری داده‌ها")
13
14  # مسیر فایل داده‌ها (فایل ورودی شامل داده‌های شبکه برای شناسایی نفوذ)
15  file_path = 'C:/Users/Mahdi/Desktop/KDDCup99.txt'
16
17  # بارگذاری داده‌ها با استفاده از Dask (پردازش سریع‌تر داده‌های بزرگ) برای پردازش موازی
18  data = dd.read_csv(file_path, delimiter=',', na_values='?', assume_missing=True)
19
20  # استخراج 10000 ردیف داده‌ها برای کاهش حجم پردازش
21  data = data.head(10000)
22
23  # نمایش پیام موفقیت در جمع‌آوری داده‌ها
24  print("داده‌ها با موفقیت جمع‌آوری شدند")
25
26  # نمایش چند ردیف اول داده‌ها برای بررسی
27  print(data.head())
```

جمع آوری داده ها

در این کد، داده‌ها از یک فایل متنی با نام KDDCup99.txt جمع‌آوری شده‌اند. فرآیند جمع‌آوری داده‌ها به این صورت انجام شده است:

بارگذاری داده‌ها با استفاده از Dask

- dask داده‌ها را به صورت موازی بارگذاری و پردازش می‌کند، بنابراین برای فایل‌های بزرگ کارایی بیشتری نسبت به pandas دارد.
- این قابلیت کمک می‌کند که با فایل‌های بسیار بزرگ بدون بارگذاری کامل در حافظه، کار کنیم.

عملکرد

- فایل ورودی با استفاده از dask.dataframe.read_csv بارگذاری می‌شود.
  - delimiter=',' : مشخص می‌کند که داده‌ها با کاما جدا شده‌اند (فرمت CSV).
  - na_values='?': مشخص می‌کند که مقادیر علامت ? به عنوان مقادیر ناشناخته (NaN) در نظر گرفته شوند.
  - assume_missing=True: برای اطمینان از اینکه ستون‌ها با داده‌های ناقص به درستی پردازش شوند.

کاهش حجم داده‌ها

- از کل مجموعه داده، فقط 10,000 ردیف انتخاب می‌شود
- کاهش حجم داده‌ها به منظور ساده‌تر و سریع‌تر کردن پردازش اولیه (به خصوص در مراحل تست و دیباگ)..

```
# مرحله 2: پیش‌پردازش داده‌ها
print("\nمرحله 2: پیش‌پردازش داده‌ها")

# تعریف نقشه‌برداری برای تبدیل برچسب‌های متنی (labels) به مقادیر عددی
label_mapping = {
    'normal': 0,  # رفتار عادی
    'buffer_overflow': 1, 'loadmodule': 1, 'perl': 1, 'neptune': 1, 'smurf': 1,
    'guess_passwd': 1, 'pod': 1, 'teardrop': 1, 'portsweep': 1, 'ipsweep': 1,
    'land': 1, 'ftp_write': 1, 'back': 1, 'imap': 1, 'satan': 1, 'phf': 1,
    'nmap': 1, 'multihop': 1, 'warezmaster': 1, 'warezclient': 1, 'spy': 1,
    'rootkit': 1  # رفتارهای غیرعادی (حملات)
}

# اعمال نقشه‌برداری به ستون برچسب‌ها
data['label'] = data['label'].map(label_mapping)

# حذف ردیف‌هایی که برچسب آنها مقدار Null دارد
data = data.dropna(subset=['label'])

# تبدیل ویژگی‌های متنی به کدهای عددی (برای استفاده در مدل یادگیری ماشین)
data['protocol_type'] = data['protocol_type'].astype('category').cat.codes
data['service'] = data['service'].astype('category').cat.codes
data['flag'] = data['flag'].astype('category').cat.codes

# جدا کردن ویژگی‌ها (X) و برچسب‌ها (y)
X = data.drop(columns=['label'])  # حذف ستون برچسب‌ها برای استفاده به‌عنوان ویژگی
y = data['label']  # برچسب‌ها (هدف)

# نرمال‌سازی داده‌های ویژگی‌ها برای بهبود عملکرد مدل
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

پیش پردازش داده ها

داده‌ها را برای استفاده در مدل یادگیری عمیق آماده می‌کند. هدف این مرحله اطمینان از این است که داده‌ها به فرم مناسب برای مدل‌های یادگیری ماشین و یادگیری عمیق تبدیل شوند.

تعریف نقشه‌برداری برای برچسب‌ها

- این نقشه‌برداری برچسب‌های متنی موجود در داده‌ها را به مقادیر عددی تبدیل می‌کند.
- مقدار 0 برای رفتارهای عادی (Normal). مقدار 1 برای رفتارهای غیرعادی (حملات).
- با استفاده از label_mapping، مقادیر ستون label به مقادیر عددی (0 یا 1) تبدیل می‌شود.

تبدیل ویژگی‌های متنی به مقادیر عددی

- ستون‌های متنی (مانند protocol_type، service و flag) به کدهای عددی تبدیل می‌شوند.
- مدل‌ها فقط می‌توانند مقادیر عددی را پردازش کنند، بنابراین مقادیر متنی باید به کدهای عددی تبدیل شوند.

نرمال‌سازی داده‌ها باStandardScaler

- نرمال‌سازی داده‌ها باعث می‌شود که میانگین هر ویژگی برابر صفر و انحراف معیار برابر یک باشد.
- این کار به مدل کمک می‌کند تا بهتر یاد بگیرد، به خصوص وقتی که مقادیر ویژگی‌ها در مقیاس‌های مختلف باشند..

```
# مرحله 3: طراحی مدل
print("\nمرحله 3: طراحی مدل")
# تقسیم داده‌ها به مجموعه‌های آموزش (80%) و آزمایش (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# بررسی عدم وجود مقادیر NaN در برچسب‌های آموزشی و آزمایشی
assert not np.any(np.isnan(y_train)), "y_train contains NaN values"
assert not np.any(np.isnan(y_test)), "y_test contains NaN values"

# تغییر شکل داده‌ها به 3 بعد برای ورودی مدل RNN
X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))  # تغییر شکل داده‌های آموزشی
X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))  # تغییر شکل داده‌های آزمایشی

# تعریف مدل RNN با استفاده از LSTM
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(X_train.shape[1], X_train.shape[2])),  # لایه ورودی با شکل داده
    tf.keras.layers.LSTM(64, return_sequences=False),  # لایه LSTM با 64 واحد
    tf.keras.layers.Dense(32, activation='relu'),  # لایه Dense با 32 نرون و فعال‌سازی ReLU
    tf.keras.layers.Dense(1, activation='sigmoid')  # لایه خروجی با فعال‌سازی سیگموئید (برای طبقه‌بندی دودویی)
])

# کامپایل مدل با تابع هزینه و بهینه‌ساز
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# تعریف Early Stopping برای جلوگیری از آموزش بیش از حد (Overfitting)
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

# تعریف Learning Rate Scheduler برای کاهش تدریجی نرخ یادگیری در هر epoch
lr_scheduler = tf.keras.callbacks.LearningRateScheduler(
    lambda epoch: 1e-3 * 0.7 ** epoch, verbose=True)
```

طراحی مدل

این تکه کد به مرحله طراحی مدل مربوط است که در آن یک مدل یادگیری عمیق بر پایه شبکه عصبی بازگشتی (RNN) با استفاده از لایه‌های LSTM طراحی و آماده استفاده می‌شود.

بررسی و تغییر شکل داده‌ها برای استفاده در RNN

داده‌ها به شکل سه‌بعدی تبدیل می‌شوند که ساختار موردنیاز برای ورودی به مدل‌های RNN است:

- بعد اول: تعداد نمونه‌ها (Samples).
- بعد دوم: تعداد گام‌های زمانی (Time Steps). در اینجا برابر 1 است، زیرا هر نمونه یک بردار ویژگی است.
- بعد سوم: تعداد ویژگی‌ها (Features) که در هر گام زمانی ورودی به مدل هستند.

تعریف مدل شبکه عصبی بازگشتی(RNN)

- تعریف شکل ورودی مدل به صورت (گام‌های زمانی، ویژگی‌ها). اینجا گام‌های زمانی برابر 1 و تعداد ویژگی‌ها برابر تعداد ستون‌های X_train است.
- لایه LSTM: برای پردازش داده‌های ترتیبی یا سری‌های زمانی بسیار مناسب است. یک لایه Long Short-Term Memory (LSTM) با 64 واحد تعریف می‌شود.
- تابع فعال‌سازی (ReLU): غیرخطی بودن را به مدل اضافه می‌کند و به آن کمک می‌کند تا روابط پیچیده‌تری را یاد بگیرد.
- لایه خروجی با فعال‌سازی Sigmoid: مقادیر خروجی را به بازه [0, 1] محدود می‌کند که برای طبقه‌بندی دودویی مناسب است.

کامپایل مدل

کامپایل مدل: مشخص می‌کند که مدل چگونه یاد می‌گیرد و ارزیابی می‌شود.

- بهینه‌ساز (Optimizer)

• لایه خروجی با فعال‌سازی Sigmoid: مقادیر خروجی را به بازه [0, 1] محدود می‌کند که برای طبقه‌بندی دودویی مناسب است.

• تابع هزینه (Loss Function):binary_crossentropy برای مسائل طبقه‌بندی دودویی استفاده می‌شود.
• معیار ارزیابی (Metrics):accuracy برای محاسبه درصد پیش‌بینی‌های صحیح.

تعریف Learning Rate Scheduler

عملکرد: نرخ یادگیری مدل را به صورت پویا در هر دوره کاهش می‌دهد.

- تابع کاهش نرخ یادگیری: کاهش تدریجی نرخ یادگیری به مدل کمک می‌کند تا به یک مینیمم پایدارتر برسد.

```
90  # مرحله 4: آموزش مدل
91  print("\nمرحله 4: آموزش مدل")
92
93  # آموزش مدل با داده‌های آموزشی و اعتبارسنجی بر اساس داده‌های آزمایشی
94  history = model.fit(X_train, y_train, epochs=20, batch_size=128, validation_data=(X_test,
    y_test),
95                      callbacks=[early_stopping, lr_scheduler])
96
97  # مرحله 5: ارزیابی مدل
98  print("\nمرحله 5: ارزیابی مدل")
99
100 # پیش‌بینی بر روی داده‌های آزمایشی
101 y_pred = (model.predict(X_test) > 0.5).astype("int32")  # تبدیل احتمال به مقادیر 0 یا 1
102
103 # گزارش طبقه‌بندی (شامل دقت، یادآوری، و F1-Score)
104 print(classification_report(y_test, y_pred))
105
106 # محاسبه و نمایش دقت مدل
107 accuracy = accuracy_score(y_test, y_pred)
108 print(f"دقت مدل: {accuracy * 100:.2f}%")
```

آموزش و ارزیابی مدل

داده‌های آموزشی و اعتبارسنجی

- X_train و y_train: داده‌های آموزشی که مدل بر اساس آنها یادگیری انجام می‌دهد.
- validation_data=(X_test, y_test): داده‌های آزمایشی که برای بررسی عملکرد مدل در طول آموزش استفاده می‌شوند. این داده‌ها در حین آموزش برای تنظیم مدل استفاده شده، اما در یادگیری مستقیم دخالت ندارند.

تعداد دوره‌ها:(Epochs)

- epochs=20: مدل در حداکثر 20 دوره (دوره‌های کامل بر داده‌های آموزشی) آموزش داده می‌شود.

اندازه دسته (Batch Size):

- batch_size=128: داده‌های آموزشی در گروه‌های 128تایی پردازش می‌شوند. این به کاهش مصرف حافظه و تسریع آموزش کمک می‌کند.

پیش‌بینی بر روی داده‌های آزمایشی

- model.predict(X_test): مدل، احتمال پیش‌بینی‌شده برای هر نمونه در داده‌های آزمایشی را برمی‌گرداند (مقدار بین 0 و 1).
- (> 0.5): اگر احتمال پیش‌بینی‌شده بزرگتر از 0.5 باشد، مقدار 1 (حمله) و در غیر این صورت مقدار 0 (عادی) تعیین می‌شود.
- astype("int32"): مقادیر پیش‌بینی‌شده به نوع عدد صحیح تبدیل می‌شوند.

گزارش طبقه‌بندی

یک گزارش کامل از معیارهای عملکرد مدل تولید و نمایش داده می‌شود.:

- Precision (دقت): نسبت پیش‌بینی‌های صحیح از میان تمام پیش‌بینی‌های مثبت.
- Recall (بازیابی): نسبت نمونه‌های مثبت که به‌درستی شناسایی شده‌اند.
- F1-Score: میانگین موزون دقت و بازیابی.
- Support: تعداد نمونه‌های هر کلاس.

محاسبه دقت مدل

- محاسبه دقت:accuracy_score(y_test, y_pred) : درصد نمونه‌های آزمایشی که به‌درستی پیش‌بینی شده‌اند.

```
109 # مرحله 6: تحلیل نتایج
110 print("\nمرحله 6: تحلیل نتایج")
111
112 # رسم نمودار دقت در طول epochها برای آموزش و اعتبارسنجی
113 plt.figure(figsize=(10, 6)) # تنظیم ابعاد نمودار
114 plt.plot(history.history['accuracy'], label='Training Accuracy') # نمودار دقت آموزشی
115 plt.plot(history.history['val_accuracy'], label='Validation Accuracy') # نمودار دقت اعتبارسنجی
116 plt.title('Model Accuracy Over Epochs') # عنوان نمودار
117 plt.xlabel('Epochs') # برچسب محور افقی
118 plt.ylabel('Accuracy') # برچسب محور عمودی
119 plt.legend() # افزودن راهنما به نمودار
120 plt.show() # نمایش نمودار
```

نتیجه گیری

جمع‌آوری داده‌ها

- داده‌ها از فایل KDDCup99.txt با استفاده از Dask بارگذاری شدند.
- برای کاهش بار پردازشی، تنها 10,000 نمونه داده مورد استفاده قرار گرفت.
- این داده‌ها شامل ویژگی‌های مختلف ترافیک شبکه و نوع رفتار (عادی یا غیرعادی) بودند.

پیش‌پردازش داده‌ها

- تبدیل برچسب‌ها: انواع حملات به مقدار عددی 1 (غیرعادی) و رفتارهای عادی به 0 تبدیل شدند.
- پردازش ویژگی‌ها:
- داده‌های متنی (مثل پروتکل‌ها) به مقادیر عددی تبدیل شدند.
- داده‌ها با استفاده از StandardScaler نرمال‌سازی شدند.
- تقسیم داده‌ها: داده‌ها به مجموعه‌های آموزشی (80٪) و آزمایشی (20٪) تقسیم شدند.
- تغییر شکل: داده‌ها به قالب سه‌بعدی تبدیل شدند تا با معماری شبکه‌های عصبی بازگشتی (RNN) سازگار باشند.

طراحی مدل

- یک مدل RNN با لایه‌های زیر طراحی شد:
- لایه LSTM: برای یادگیری توالی‌های زمانی داده‌ها.
- لایه Dense: برای ترکیب ویژگی‌های یادگرفته شده.
- لایه خروجی: با فعال‌سازی سیگموئید برای پیش‌بینی طبقه‌بندی دودویی.
- از Early Stopping و Learning Rate Scheduler برای بهبود فرآیند آموزش و جلوگیری از Overfitting استفاده شد.

آموزش مدل

- مدل با 20 دوره (epoch) و حجم دسته‌ای (batch size) برابر 128 آموزش داده شد.
- داده‌های آزمایشی برای اعتبارسنجی استفاده شدند.
- یادگیری نرخ (Learning Rate) به‌تدریج کاهش یافت.

ارزیابی مدل

- گزارش طبقه‌بندی (Classification Report):
- مدل عملکرد بسیار خوبی داشته است:
- Precision (دقت): درصد پیش‌بینی‌های درست در مقایسه با تمام پیش‌بینی‌های مثبت.
- برای هر دو کلاس (عادی و غیرعادی) برابر 1.00 (100٪) است.
- Recall (بازخوانی): درصد نمونه‌های مثبت که به‌درستی شناسایی شده‌اند.
- برای کلاس عادی: 1.00.
- برای کلاس غیرعادی: 0.99.
- F1-Score: میانگین هارمونیک دقت و بازخوانی، که برای هر دو کلاس نزدیک به 1.00 است.
- Overall Accuracy (دقت کل): دقت نهایی مدل 99.85٪ است که نشان‌دهنده عملکرد عالی مدل است.

مدل طراحی‌شده به‌طور موفقیت‌آمیزی حملات شبکه را شناسایی می‌کند.

دقت 99.85٪ بیانگر این است که مدل در یادگیری رفتارهای ترافیک شبکه و شناسایی نفوذ بسیار قوی عمل کرده است.مراحل مختلف از جمع‌آوری داده‌ها تا پیش‌پردازش، طراحی، آموزش، و ارزیابی بدون خطا اجرا شده‌اند.این سیستم می‌تواند به‌عنوان بخشی از یک راه‌حل امنیت سایبری در شناسایی نفوذها و رفتارهای غیرعادی در شبکه استفاده شود.