DataFusion 2023

Meetup 16.02.2023

Кратко о задачах

В процессе атаки

Подменяем у 10 транзакций каждого пользователя mcc код и сумму транзакции (в допустимых рамках для нового mcc кода и тем же знаком, что и исходная транзакция)

В процессе защиты

Защищаем как можно лучше модель от любых искажений данных

Дополнение про данные

4200 пользователей для атаки
7080 пользователей с разметкой для дообучения защиты

Суммы немного отличаются от реальных
У каждого пользователя гарантированно 300 транзакций
Время для всех транзакций одинаковое (вне зависимости от часового пояса)

Как обрабатываются данные?

.dropna() – в данном случае пережиток прошлого, NaN в ваших данных нет

```
df_transactions = (
    pd.read_csv(
        source_file,
        parse_dates=["transaction_dttm"],
        dtype={"user_id": int, "mcc_code": int, "currency_rk": int, "transaction_amt": float},
    )
    .dropna()
    .assign(
        hour=lambda x: x.transaction_dttm.dt.hour,
        day=lambda x: x.transaction_dttm.dt.dayofweek,
        month=lambda x: x.transaction_dttm.dt.month,
        number_day=lambda x: x.transaction_dttm.dt.day,
    )
)
```

Как обрабатываются данные?

nn_bins – сформирован во время тренировки. Для суммы – 100 бакетов (квантили), остальное – уникальные значения

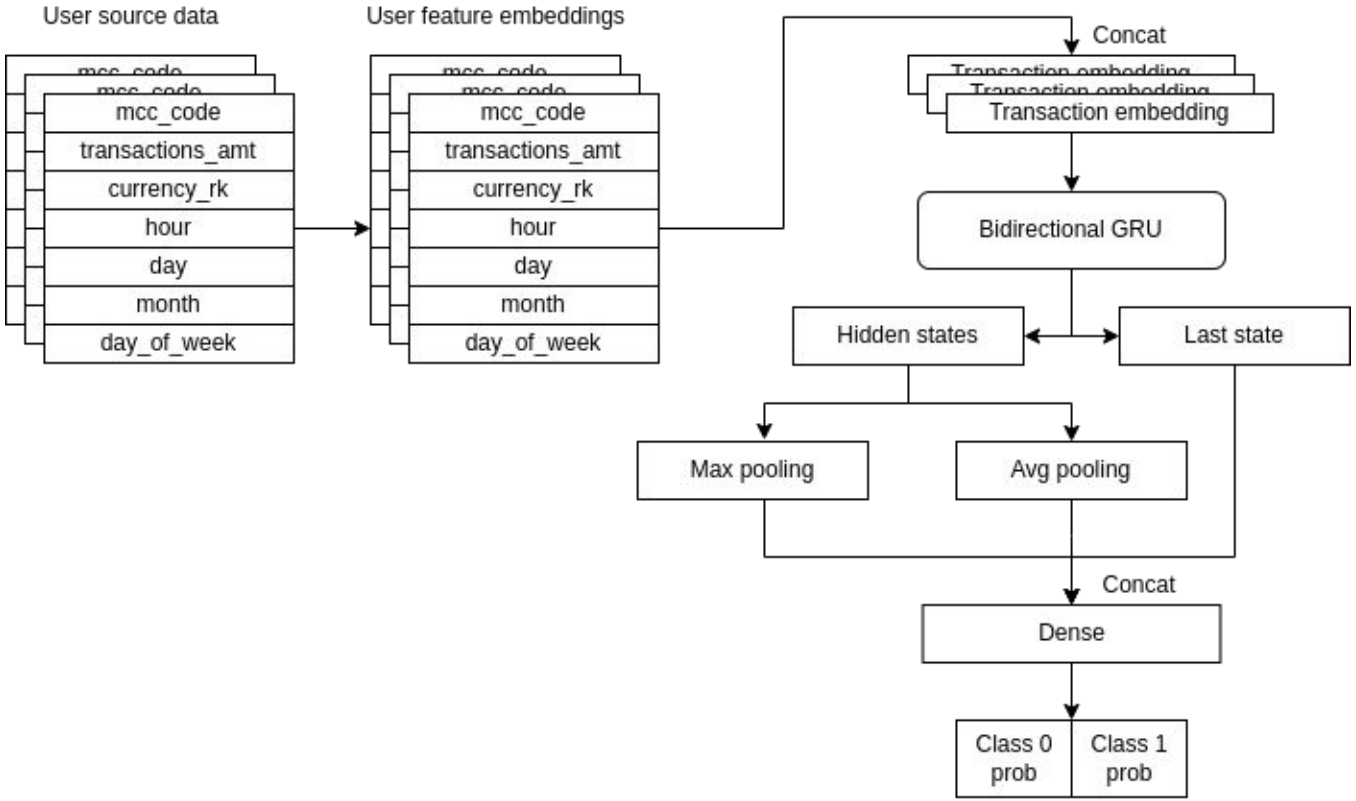
```python
def process_for_nn(transactions_frame, features, nn_bins, *, need_padding=True):
    transactions_frame = transactions_frame[transactions_frame["mcc_code"].notna()]

    assert set(nn_bins.keys()) <= set(transactions_frame.columns), "Something wrong"

    def digitize_cat(df):
        for dense_col in nn_bins.keys():
            if dense_col == "transaction_amt":
                df[dense_col] = pd.cut(df[dense_col], bins=nn_bins[dense_col], labels=False).astype(int)
            else:
                df[dense_col] = pd.cut(
                    df[dense_col].astype(float).astype(int), bins=nn_bins[dense_col], labels=False,
                ).astype(int)

        return df

    after_digitize = transactions_frame.pipe(digitize_cat)
```

Как обрабатываются данные?

Обрезаем данные и делаем паддинг до 300 транзакций
Сейчас это не нужно, т.к. для каждого пользователя гарантированно есть 300 транзакций

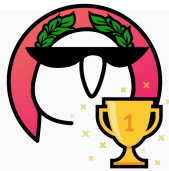
```
num_transactions = 300
if not need_padding:
    num_transactions = 1
return (
    after_digitize.groupby(["user_id"])[features]
    # take last 300 transactions
    .apply(lambda x: x.values.transpose()[:, -num_transactions:].tolist())
    # additional padding to 300
    .apply(lambda x: np.array([list(i) + [0] * int(num_transactions - len(x[0])) for i in x]))
    .reset_index()
    .rename(columns={0: "sequences"})
)
```

Атака! Защита! Модель?



Атака

Материалы задачи Атака



1. Неразмеченные тестовые данные для атаки `sample_submission.csv` (**57 MB**)
2. Файл с лимитами на измененные суммы транзакций по MCC кодам `quantiles.json` (**32 KB**)
3. Пример наивного baseline решения `naive_submission.csv` (**55 MB**)
4. Архив с кодом наивного baseline решения для воспроизведения участниками `naive_baseline.zip` (**1 MB**)

sample_submission.csv – меняем этот файл
quantiles.json – допустимые суммы для каждого mcc_code, используется при проверке ограничений на бюджет (**check_budget.py** в **naive_baseline.zip**)
naive_submission.csv – из ноутбука в **naive_baseline.zip**
naive_baseline.zip рассмотрим далее

Ссылка на страницу с данными на атаку

naive_baseline.zip

model.py – предикт модели на транзакциях (.csv файл). На выходе .csv файл
check_budget.py – проверка бюджета атаки. Используется внутри системы
quantiles.json – границы для каждого mcc_code

```
# для каждого кода заданы лимиты положительных и отрицательных значений
# Вот, например, диапазон, в котором должны лежать суммы для mcc_code 4111
quantiles["positive"]["min"]["4111"], quantiles["positive"]["max"]["4111"]
```

```
(8.145073890686035, 45117.69001)
```

nn_bins.pickle, **nn_weights.ckpt** – файлы для работы предобученной модели

naive_baseline.zip – naive_submission.ipynb

Стратегия примерно следующая:

1) Крафтим атаку
2) Проверяем функцией check_budget валидность атаки
3) Оцениваем насколько уронили ROC AUC
4) Крутой результат? Отправляем!

- Метрика соревнования — **ROC-AUC Diff** между результатом модели на исходных данных и ее результатом на вашем решении.

In 1

```python
import json

import numpy as np
import pandas as pd
from tqdm import tqdm
import random
from model import predict # Функция, позволяет получить предсказание нейронки.
from check_budget import check_budget # функция проверки бюджета. Проверяйте допустимость решения до сабмита
```

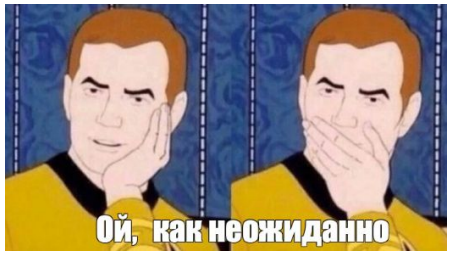
Как сделать атаку?

Идея бейзлайна атаки

- Берутся самые яркие представители 0 и 1 классов
- Берутся их **последние** транзакции
- Подставляются остальным в **конец** (с учетом знаков)

Плюсы: просто

Минусы: просто



naive_baseline.zip – naive_submission.ipynb

Как делаем атаку?

– Получаем разметку из модели
– Берём самого яркого представителя из 0 и 1 классов
– Берём у них последние транзакции,
— для тех, у кого класс 0, подставляем транзакции класса 1 и наоборот
— учитываем знак транзакции исходного пользователя

WARNING: при смене знака следите за соблюдением ограничений. Лучше сразу сделать **генератор валидных сумм** для mcc кода и знака

naive_baseline.zip – naive_submission.ipynb

Правим транзакции для каждого пользователя

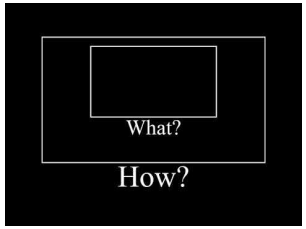
```
for user in tqdm(users):
    if user in one_users:
        copy_from = poor_user # похожим на Героя скопируем 10 последних транзакций Неудачника
    else:
        copy_from = hero_user # А похожим на Неудачника наоборот

    idx_to = df_transactions.index[df_transactions.user_id == user][-BUDGET:]
    idx_from = df_transactions.index[df_transactions.user_id == copy_from][-BUDGET:]
    sign_to = np.sign(df_transactions.loc[idx_to, "transaction_amt"].values)
    sign_from = np.sign(df_transactions.loc[idx_from, "transaction_amt"].values)
    sign_mask = (sign_to == sign_from)
    df_transactions.loc[idx_to[sign_mask], "mcc_code"] = df_transactions.loc[idx_from[sign_mask],
     "mcc_code"].values
    df_transactions.loc[idx_to[sign_mask], "transaction_amt"] = df_transactions
     .loc[idx_from[sign_mask], "transaction_amt"].values
df_transactions.to_csv(output_path, index=False)
```

Улучшаем бейзлайн атаки

- Нужно освежить в памяти как работает Bidirectional GRU
- Разные последовательности для разных типов пользователей
- Ищем паттерны каждого пользователя
- Сделать генератор сумм

Куда копать дальше

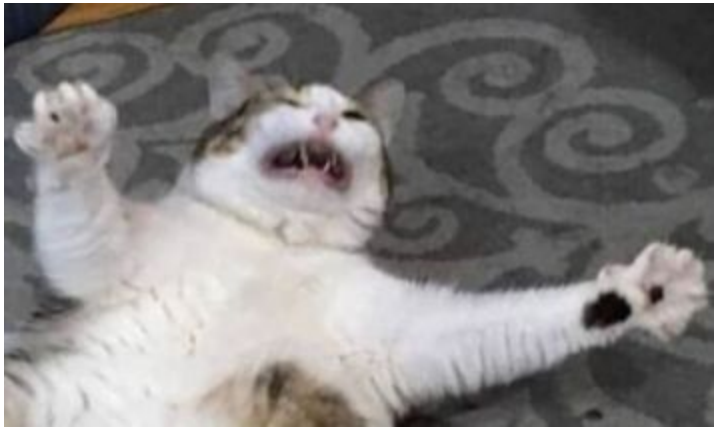
Есть два типа статей про атаку:
- **White-box – знаем архитектуру, есть доступ к параметрам**
- Black-box – модель неизвестна, видим только предикты



Рекомендуем:

- FGSM Sampling Fool и их вариации
- FGSM для рекуррентной сети
- Описание других видов атак и защит

Защита



Материалы задачи Защита

1. Пример решения `sample_submission.zip` **(1 MB)**
2. Пример решения `submit_defence.zip` **(1 MB)**

sample_submission.zip – пример как надо

submit_defence.zip – наш бейзлайн защиты – рассмотрим далее

Ссылка на страницу с данными на защиту

submit_defence.zip

– **metadata.json** – какой образ взять и как запустить скрипт

```
{
    "image": "odsai/vtb23-data-fusion:0.0.1",
    "entry_point": "python -u model.py"
}
```

– **model.py** – скрипт, который сделает предикт. На вход имя .csv файла для предикта и имя выходного .csv файла с результатом работы

– **nn_bins.pickle nn_weights.ckpt** – файлы, необходимые для работы бейзлайна

submit_defence.zip – metadata.json

– **metadata.json** – какой образ взять и как запустить скрипт

```
{
    "image": "odsai/vtb23-data-fusion:0.0.1",
    "entry_point": "python -u model.py"
}
```

Будут вызовы вашего скрипта по типу

python -u model.py attacked.csv out.csv

И мы оценим **out.csv**

Структуру результата смотреть в sample_submission.zip

submit_defence.zip – model.py

– **model.py** – скрипт, который сделает предикт. На вход имя .csv файла для предикта и имя выходного .csv файла с результатом работы модели

Общая структура:

```
def main():
    source_file, output_path = sys.argv[1:]
    bins_path = "nn_bins.pickle"
    model_path = "nn_weights.ckpt"
    result = reliable_predict(source_file, bins_path, model_path)
    result.to_csv(output_path, index=False)
```

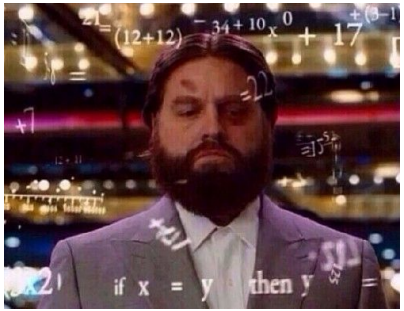
Data Fusion

Как сделать защиту?

Идея бейзлайна защиты

Test-time augmentation

- Семплируем 90% транзакций каждого пользователя
- Делаем предикт
- Делаем так 9 раз
- Усредняем результат для каждого пользователя



submit_defence.zip model.py подготовка

```python
def reliable_predict(source_file, bins_path, model_path, random_seed=20230206):
    REPETITIONS = 9  # Сколько повторений делаем
    pl.seed_everything(random_seed)

    df_transactions = (
        pd.read_csv(
            source_file,
            parse_dates=["transaction_dttm"],
            dtype={"user_id": int, "mcc_code": int, "currency_rk": int, "transaction_amt": float},
        )
        .dropna()
        .assign(
            hour=lambda x: x.transaction_dttm.dt.hour,
            day=lambda x: x.transaction_dttm.dt.dayofweek,
            month=lambda x: x.transaction_dttm.dt.month,
            number_day=lambda x: x.transaction_dttm.dt.day,
        )
    )
```

submit_defence.zip model.py подготовка

```python
with open(bins_path, "rb") as f:
    bins = pickle.load(f)

features = bins.pop("features")

model = TransactionsRnn()
model.load_state_dict(torch.load(model_path))
model.to(device)
model.eval()
```

submit_defence.zip model.py начинка

```
results = []
for i in range(REPETITIONS):
    # Сэмплируем данные. Каждый раз - разный random_seed
    df = process_for_nn(df_transactions.sample(frac=0.9, random_state=random_seed+i, replace=True),
                        features, bins)
    dataset = TransactionsDataset(df)
    dataloader = get_dataloader(dataset, device, is_validation=True)
    preds = []
    users = []
    for data, target in dataloader:
        y_pred = model(data)
        preds.append(y_pred.detach().cpu().numpy())
        users.append(target.detach().cpu().numpy())
    preds = np.concatenate(preds)
    users = np.concatenate(users)
    results.append(pd.DataFrame({"user_id": users, "target": preds[:, 1]}))
results[0]['target'] = np.mean([x.target.values for x in results], axis=0)  # усредняем предсказания
return results[0]
```

Улучшаем бейзлайн защиты

- Дообучите на тех данных, которые мы дали
- Можно оставить модель немного недообученной, пожертвовав точностью
- Попробуйте дообучить модель на данных с аугментацией

Куда копать дальше

1) Обрабатываем пачки (nonlocal denoising method), а не каждую конкретную транзакцию
2) Клёво решаем задачу атаки и тренируемся на атакованных данных (adversarial training)
3) Разные подходы к защите (урезаем пространство фичей, делаем ансамбль, обучаем GAN, определяем adversarial пример и т.д.)
4) Принимаем решение по-другому (сэмплинг + выбираем как-нибудь по особому класс)
5) А-ля учебник по тому, как противостоять разным атакам
6) Можно обучить другую модель

Вопросы, обсуждение

Meetup 16.02.2023