jupyter 02_preprocessing_feature_eng Last Checkpoint: 6 hours ago

File Edit View Run Kernel Settings Help — Not Trusted

Markdown — JupyterLab — Python 3 (ipykernel)

Cell 1 – Load Parsed Sessions from Pickle File

This cell loads the parsed SLAM sessions (slam_sessions.pkl) from the data/interim folder. These are the raw token-level session logs we will use to extract metadata such as user ID, session type, client platform, time spent, and activity days

[33]:
```python
import pickle
from pathlib import Path
import pandas as pd
import numpy as np

# Set paths
project_root = Path.cwd().parent
parsed_sessions_path = project_root / "data" / "interim" / "slam_sessions.pkl"

# Load pickle file
with open(parsed_sessions_path, "rb") as f:
    slam_sessions = pickle.load(f)

print(f"✅ slam_sessions loaded. Total sessions: {len(slam_sessions)}")
```

```
✅ slam_sessions loaded. Total sessions: 824012
```

Cell 2 – Extract Metadata to DataFrame

This cell extracts key metadata from each session such as user_id, session_type, client, days, time, etc. We create a structured DataFrame df_sessions and handle missing/null values for numeric fields

[34]:
```python
def extract_metadata(session):
    metadata = {
        "prompt": "", "user_id": "", "countries": "", "days": 0,
        "client": "", "session_type": "", "format": "", "time": 0
    }
    for line in session:
        if line.startswith("# prompt:"):
            metadata["prompt"] = line.replace("# prompt:", "").strip()
        elif line.startswith("# user:"):
            parts = line.replace("# user:", "").strip().split()
            metadata["user_id"] = parts[0]
            for part in parts[1:]:
                if ":" in part:
                    key, value = part.split(":")
                    if key == "session":
                        metadata["session_type"] = value
                    elif key == "days":
                        metadata["days"] = value
                    elif key in metadata:
                        metadata[key] = value
    return metadata

# Extract metadata only for sessions with a prompt
parsed_metadata = [
    extract_metadata(s) for s in slam_sessions
    if any("# prompt:" in line for line in s)
]

# Create DataFrame
df_sessions = pd.DataFrame(parsed_metadata)

# Clean 'time' and 'days' columns
df_sessions['time'] = pd.to_numeric(df_sessions['time'].replace(['null', '', None], 0))
df_sessions['days'] = pd.to_numeric(df_sessions['days'].replace(['null', '', None], 0))

# Save to CSV
output_path = project_root / "data" / "processed" / "parsed_sessions.csv"
output_path.parent.mkdir(parents=True, exist_ok=True)
df_sessions.to_csv(output_path, index=False)

# Print summaries
print(f"📁 Metadata saved to: {output_path}")
print(f"✅ Total rows saved: {len(df_sessions)}\n")
print("📊 Session Type Breakdown:\n", df_sessions['session_type'].value_counts())
print("\n💻 Client Breakdown:\n", df_sessions['client'].value_counts())
print("\n📈 Numeric Stats:\n", df_sessions[['days', 'time']].describe())
```

```
📁 Metadata saved to: f:\Bachleros Research\Rsearch thesis\New folder\Predicting-Churn-using-ML-and-DL\data\processed\parsed_sessions.csv
✅ Total rows saved: 595100

📊 Session Type Breakdown:
 session_type
lesson      493236
practice     93907
test          7957
Name: count, dtype: int64

💻 Client Breakdown:
 client
android    416645
ios        107590
web         70865
Name: count, dtype: int64

📈 Numeric Stats:
                 days           time
count  595100.000000  595100.000000
mean        6.072470      24.693122
std         5.631776     766.344798
min         0.000000    -156.000000
25%         1.307000       5.000000
50%         4.355500       9.000000
75%         9.284000      17.000000
max        28.042000  330554.000000
```

Cell 3 – Inspect Raw Lines with Session-Type Information

This helps verify that session types like correctly tagged in the logs

[35]:
```python
# ------------------- CELL 3: Inspect Session Types -------------------
print("🔍 Checking session lines that might contain 'session_type':\n")

for idx, session in enumerate(slam_sessions[:10]):
    print(f"\n🆕 Session {idx + 1}:")
    for line in session:
        if "session_type" in line.lower() or "session" in line.lower():
            print("  ", line)
```

```
🔍 Checking session lines that might contain 'session_type':


🆕 Session 1:
   # user:XEinXf5+  countries:CO  days:0.003  client:web  session:lesson  format:reverse_translate  time:9

🆕 Session 2:
   # user:XEinXf5+  countries:CO  days:0.005  client:web  session:lesson  format:reverse_translate  time:12

🆕 Session 3:
   # user:XEinXf5+  countries:CO  days:0.008  client:web  session:lesson  format:reverse_translate  time:6

🆕 Session 4:
   # user:XEinXf5+  countries:CO  days:0.008  client:web  session:lesson  format:reverse_translate  time:13

🆕 Session 5:
   # user:XEinXf5+  countries:CO  days:0.008  client:web  session:lesson  format:reverse_translate  time:16

🆕 Session 6:
   # user:XEinXf5+  countries:CO  days:0.011  client:web  session:lesson  format:reverse_translate  time:10

🆕 Session 7:
   # user:XEinXf5+  countries:CO  days:0.011  client:web  session:lesson  format:reverse_translate  time:5

🆕 Session 8:
   # user:XEinXf5+  countries:CO  days:0.016  client:web  session:lesson  format:reverse_translate  time:11

🆕 Session 9:
   # user:XEinXf5+  countries:CO  days:0.016  client:web  session:lesson  format:reverse_translate  time:10

🆕 Session 10:
   # user:XEinXf5+  countries:CO  days:0.018  client:web  session:lesson  format:listen  time:5
```

Cell 4 – Filter Early Sessions (≤14 Days)

We focus only on sessions from the first 14 days of activity per user. This matches our proposal's scope for predicting churn based on early behavior.

[36]:
```python
# Remove sessions with missing or non-positive time
df_cleaned = df_sessions.dropna(subset=["time"])
df_cleaned = df_cleaned[df_cleaned["time"] > 0]

# IQR filtering for outlier removal (based on session time)
Q1 = df_cleaned["time"].quantile(0.25)
Q3 = df_cleaned["time"].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
df_cleaned = df_cleaned[(df_cleaned["time"] >= lower_bound) & (df_cleaned["time"] <= upper_bound)]

# Normalize session type text
df_cleaned["session_type"] = df_cleaned["session_type"].astype(str).str.lower()

# Filter for early sessions (within first 14 days)
early_sessions = df_cleaned[df_cleaned['days'] <= 14].copy()

print(f"✅ Filtered early sessions: {early_sessions.shape[0]} rows")
```

```
✅ Filtered early sessions: 481271 rows
```

🧮 Cell 5 – Feature Engineering: Aggregate Session Data Per User

We create user-level features to summarize behavior across sessions. These include:

- Average session time, total time, and count of sessions
- First and last active day, and list of unique active days
- Counts of different `session_type` and `client` (platforms)

These features are essential for churn modeling and behavioral analysis.

[37]:
```python
user_features = early_sessions.groupby('user_id').agg({
    'time': ['mean', 'sum', 'count'],
    'days': ['min', 'max', 'unique'],
    'session_type': lambda x: x.value_counts().to_dict(),
    'client': lambda x: x.value_counts().to_dict()
}).reset_index()

# Flatten MultiIndex column names
user_features.columns = [
    'user_id', 'avg_time', 'total_time', 'session_count',
    'first_day', 'last_day', 'active_days',
    'session_type_counts', 'client_counts'
]

print("✅ User features created:")
user_features.head()
```

```
✅ User features created:
```

[37]:

	user_id	avg_time	total_time	session_count	first_day	last_day	active_days	session_type_counts	client_counts
0	++j955YG	9.407895	1430	152	0.004	13.102	[0.004, 1.059, 1.063, 1.065, 1.925, 1.93, 1.93...	{'lesson': 131, 'practice': 21}	{'android': 142, 'web': 10}
1	+/iDvu/I	9.309973	3454	371	0.369	12.905	[0.369, 0.371, 0.379, 2.315, 2.317, 2.319, 2.3...	{'lesson': 328, 'practice': 43}	{'android': 371}
2	+0UEF02n	12.648649	1404	111	0.006	1.443	[0.006, 0.01, 0.015, 0.019, 0.029, 0.032, 0.03...	{'lesson': 107, 'practice': 4}	{'ios': 111}
3	+197nchq	17.761468	1936	109	0.023	13.013	[0.023, 1.026, 1.046, 1.88, 1.885, 1.894, 3.00...	{'lesson': 107, 'practice': 2}	{'android': 109}
4	+7lbKZrn	10.307339	2247	218	0.469	13.451	[0.469, 1.414, 1.418, 1.422, 1.491, 1.494, 1.5...	{'lesson': 122, 'practice': 96}	{'android': 218}

Cell 6 – Label Churn (User Inactivity After Day 14)

Churn is defined as no activity after day 14 in full user timeline. This uses the cleaned df_cleaned which includes all valid session data

[38]:
```python
# Full user last active day across the full timeline
user_last_day = df_cleaned.groupby('user_id')['days'].max()

# Define churn: users whose last day is ≤ 14 are considered churned
churn_labels = (user_last_day <= 14).astype(int)  # 1 = Churned, 0 = Retained

# Map churn labels back to the early session-based user features
user_features['churned'] = user_features['user_id'].map(churn_labels).fillna(0).astype(int)

print("✅ Churn labels assigned.")
print(user_features['churned'].value_counts())
```

```
✅ Churn labels assigned.
churned
1    1647
0     933
Name: count, dtype: int64
```

Cell 7 – Flatten Nested Dictionaries to Columns

This cell converts session_type_counts and columns like client_counts into separate session_type_lesson , client_android , etc. This prepares the features for ML modeling.

[39]:
```python
# Expand session type and client counts
session_df = pd.json_normalize(user_features['session_type_counts']).fillna(0)
client_df = pd.json_normalize(user_features['client_counts']).fillna(0)

# Add proper column prefixes
session_df = session_df.add_prefix('session_type_')
client_df = client_df.add_prefix('client_')

# Merge with user_features
df_expanded = pd.concat([
    user_features.drop(['session_type_counts', 'client_counts'], axis=1),
    session_df,
    client_df
], axis=1)

print("✅ Expanded user features shape:", df_expanded.shape)
df_expanded.head()
```

```
✅ Expanded user features shape: (2580, 14)
```

[39]:

	user_id	avg_time	total_time	session_count	first_day	last_day	active_days	churned	session_type_lesson	session_type_practice	session_type_test	client_android
0	++j955YG	9.407895	1430	152	0.004	13.102	[0.004, 1.059, 1.063, 1.065, 1.925, 1.93, 1.93...	0	131.0	21.0	0.0	142.0
1	+/iDvu/I	9.309973	3454	371	0.369	12.905	[0.369, 0.371, 0.379, 2.315, 2.317, 2.319, 2.3...	1	328.0	43.0	0.0	371.0
2	+0UEF02n	12.648649	1404	111	0.006	1.443	[0.006, 0.01, 0.015, 0.019, 0.029, 0.032, 0.03...	1	107.0	4.0	0.0	0.0
3	+197nchq	17.761468	1936	109	0.023	13.013	[0.023, 1.026, 1.046, 1.88, 1.885, 1.894, 3.00...	0	107.0	2.0	0.0	109.0
4	+7lbKZrn	10.307339	2247	218	0.469	13.451	[0.469, 1.414, 1.418, 1.422, 1.491, 1.494, 1.5...	0	122.0	96.0	0.0	218.0

Cell 8 – Save Final Dataset:

user_features_expanded.csv The final dataset contains 14+ features per user, including churn labels. This will be used in EDA and ML notebooks.

[40]:
```python
# Save the final expanded dataset
final_output_path = project_root / "data" / "processed" / "user_features_expanded.csv"
df_expanded.to_csv(final_output_path, index=False)

print(f"✅ Final dataset saved to: {final_output_path}")
print(f"📊 Total users: {df expanded.shape[0]}")
print("📋 Schema:")
print(df_expanded.info())
print("\n🕳 Missing values:")
print(df_expanded.isnull().sum())
```

```
✅ Final dataset saved to: f:\Bachleros Research\Rsearch thesis\New folder\Predicting-Churn-using-ML-and-DL\data\processed\user_features_expanded.csv
📊 Total users: 2580
📋 Schema:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2580 entries, 0 to 2579
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   user_id                2580 non-null   object
 1   avg_time               2580 non-null   float64
 2   total_time             2580 non-null   int64
 3   session_count          2580 non-null   int64
 4   first_day              2580 non-null   float64
 5   last_day               2580 non-null   float64
 6   active_days            2580 non-null   object
 7   churned                2580 non-null   int64
 8   session_type_lesson    2580 non-null   float64
 9   session_type_practice  2580 non-null   float64
 10  session_type_test      2580 non-null   float64
 11  client_android         2580 non-null   float64
 12  client_web             2580 non-null   float64
 13  client_ios             2580 non-null   float64
dtypes: float64(9), int64(3), object(2)
memory usage: 282.3+ KB
None

🕳 Missing values:
user_id                  0
avg_time                 0
total_time               0
session_count            0
first_day                0
last_day                 0
active_days              0
churned                  0
session_type_lesson      0
session_type_practice    0
session_type_test        0
client_android           0
client_web               0
client_ios               0
dtype: int64
```