### **Big Picture**

* Week of 1/27:
  + setup github repo, google colab, game plan
* Week of 2/3: analysis > [**slides**](https://docs.google.com/presentation/d/1eGvioOmmYqEIkIjLWEPR3mkFwEQJX08aCSwGZXsV_G4/edit#slide=id.p)
* Week of 2/10: polish slides, practice presenting
* 2/13 midterm presentation day
  + 10min including q&a
  + Aim for 8-9min

### **Notes**

#### 2025-02-13

* Review updates / team decisions
* practice presentation! \*time it

#### 2025-02-12

* Todo:
  + AR, JY, work on HW
  + Test on test set (SL, HS)
  + Finalize/polish ipynb
    - SL still need to import a few more plot code chunks i think
    - Generally tidy
  + Finalize slides
    - SL Training metrics vizs
      * One slide w three confusion matrices
      * One slide w three roc curves
      * One slide w feature engineering \*hide
  + Schedule time to practice presenting tmr \*time it
    - AR anytime after 11:00am
    - JY pref before 12:30, otherwise 1:30-3:00
    - SL anytime

#### 2025-02-07

* This weekend: JY, SL, and HS
* Model implementation: HS and AR
* [TA office hour update](https://www.notion.so/Meeting-with-Stat414-TA-team-discussion-193aec3513f380c39bcdde8323ba45a6)

#### 2025-02-04:

* Does drive mount work for you guys - new solution ✅
* Data label classification ?
  + Will be covered in discussion
* Any model, in or out of class
  + Try XGboost\* and log reg
* SL: Colab crashing on model.fit
  + fit on sample of data?
  + Sgd classifier - for large datasets
  + Sparsify data first?
* HS: training data split ratio? 80/20?
* Todo:

1. AR: Clean / feature engineer data + visualization

2. JY: Correlation test to determine most important features  
3. HS: XGboost  
4.. SL: Logistic regression

- Only w important features > XGboost + Logistic regression  
- compare AUC

* Start Slides (SL)
* Ask TA how to handle columns(HS)
* Set Scale\_pos\_weight = 70 in xgboost

#### 2025-01-27:

* Clean data / basic EDA
  + Correlation matrix
  + multicollinearity?
* Predicting Click-Through-Rate (CTR)
* algorithm(s) / strategies to use…
  + Logistic regression
  + Adaboost / random forests?
  + **Xgboost**? decision tree

### **Assignment**

* [Data Link](https://www.kaggle.com/datasets/xiaojiu1414/digix-global-ai-challenge/data)
  + About Dataset: Ad recommendation models are usually built based on historical *ad impressions, clicks, and other user behavior data*. If only data from the ads domain is used, user behavior data will be sparse, and the user behavior types that can be identified will be limited. However, if a user's behavior data in other domains from the same app is explored, the user's interests and behavior characteristics can be better identified. Of course, introducing user behavior data from other apps can also help enrich the data of user behavior characteristics and ad performance.You are expected to enhance ads click-through rate (**CTR**) prediction accuracy by leveraging ad logs, user profiles, and cross-domain data. With ads as the target domain and news feeds as the source domain, you should build user interest models through impressions, clicks, and other user behavior data obtained from the news feeds domain, thus improving the CTR prediction performance of the ads domain.
  + Source domain:
    - news feed data (28 fields)
  + Target domain:
    - CTR prediction performance

### **Field Descriptions**

label (target, Y)

* **Description**: The target variable you’re trying to predict.
* **Purpose**: This could represent whether a user clicked on an ad (binary classification) or some other metric like conversion rate or engagement level.
* **Target Variable**: The label column is your target variable, so focus on understanding how other columns relate to it.

++++ Deepseek recommend ++++++

number of clicks = aggregate of like ad\_click\_list

* **Feature Engineering**: Use columns like ad\_click\_list and ad\_close\_list to create features like "number of clicks" or "number of closes."

number of clicks = aggregate of ad\_close\_list

* **Feature Engineering**: Use columns like ad\_click\_list and ad\_close\_list to create features like "number of clicks" or "number of closes."

1. log\_id

* **Description**: A unique identifier for each log entry or record.
* **Purpose**: Used to uniquely identify each row in the dataset.

3. user\_id

* **Description**: A unique identifier for each user.
* **Purpose**: Tracks which user performed the action (e.g., clicked an ad).

4. age

* **Description**: The age of the user.
* **Purpose**: Used to analyze how age correlates with ad engagement or clicks.

5. gender

* **Description**: The gender of the user (likely encoded as integers, e.g., 1 for male, 2 for female).
* **Purpose**: Helps in understanding gender-based trends in ad interactions.

6. residence

* **Description**: The user’s residence or location (likely encoded as integers).
* **Purpose**: Used to analyze regional trends in ad engagement.

7. city

* **Description**: The city where the user is located.
* **Purpose**: Provides more granular location data for analysis.

8. city\_rank

* **Description**: A ranking or tier of the city (e.g., based on population or economic activity).
* **Purpose**: Helps in understanding how city size or importance affects ad engagement.

9. series\_dev

* **Description**: Likely refers to the series or version of the device the user is using.
* **Purpose**: Used to analyze how device type affects ad interactions.

10. series\_group

* **Description**: A grouping or category of the device series.
* **Purpose**: Provides a higher-level categorization of devices.

11. emui\_dev

* **Description**: Likely refers to the EMUI version (a custom Android skin by Huawei) on the user’s device.
* **Purpose**: Used to analyze how the device’s software version affects ad interactions.

12. device\_name

* **Description**: The name or model of the user’s device.
* **Purpose**: Helps in understanding how specific devices affect ad engagement.

13. device\_size

* **Description**: The screen size or storage size of the device.
* **Purpose**: Used to analyze how device specifications affect ad interactions.

14. net\_type

* **Description**: The type of network the user is connected to (e.g., Wi-Fi, 4G, 5G).
* **Purpose**: Helps in understanding how network conditions affect ad engagement.

15. task\_id

* **Description**: A unique identifier for the task or ad campaign.
* **Purpose**: Tracks which ad campaign the interaction is associated with.

16. adv\_id

* **Description**: A unique identifier for the advertiser.
* **Purpose**: Tracks which advertiser the ad belongs to.

17. creat\_type\_cd

* **Description**: The type of creative used in the ad (e.g., image, video, text).
* **Purpose**: Helps in understanding how different ad formats perform.

18. adv\_prim\_id

* **Description**: A primary identifier for the ad or creative.
* **Purpose**: Tracks specific ads or creatives within a campaign.

19. inter\_type\_cd

* **Description**: The type of interaction (e.g., click, view, close).
* **Purpose**: Tracks how users interacted with the ad.

20. slot\_id

* **Description**: The slot or position where the ad was displayed (e.g., top banner, sidebar).
* **Purpose**: Helps in understanding how ad placement affects engagement.

21. site\_id

* **Description**: The site or platform where the ad was displayed.
* **Purpose**: Tracks which website or app the ad was shown on.

22. spread\_app\_id

* **Description**: The app or platform used to spread the ad.
* **Purpose**: Tracks the distribution channel for the ad.

23. hispace\_app\_tags

* **Description**: Tags or categories associated with the app (e.g., gaming, social media).
* **Purpose**: Helps in understanding how app categories affect ad engagement.

24. app\_second\_class

* **Description**: A secondary classification or subcategory of the app.
* **Purpose**: Provides more granular app categorization.

25. app\_score

* **Description**: A score or rating of the app.
* **Purpose**: Used to analyze how app quality affects ad interactions.

26. ad\_click\_list\_v001, ad\_click\_list\_v002, ad\_click\_list\_v003

* **Description**: Lists of ad IDs that the user clicked on, separated by a caret (^).
* **Purpose**: Tracks the user’s click behavior across multiple ads.

27. ad\_close\_list\_v001, ad\_close\_list\_v002, ad\_close\_list\_v003

* **Description**: Lists of ad IDs that the user closed, separated by a caret (^).
* **Purpose**: Tracks the user’s behavior of closing ads.

28. pt\_d

* **Description**: The date and time of the interaction (e.g., 202206051230 for June 5, 2022, 12:30 PM).
* **Purpose**: Tracks when the interaction occurred.

29. u\_newsCatInterestsST

* **Description**: The user’s interests in news categories, separated by a caret (^).
* **Purpose**: Helps in understanding how user interests affect ad engagement.

30. u\_refreshTimes

* **Description**: The number of times the user refreshed the page or app.
* **Purpose**: Tracks user engagement with the platform.

31. u\_feedLifeCycle

* **Description**: The lifecycle stage of the user’s feed (e.g., new, active, dormant).
* **Purpose**: Helps in understanding how user activity levels affect ad engagement.

Summary of Key Groups

1. **User Metadata**: user\_id, age, gender, residence, city, city\_rank.
2. **Device Metadata**: series\_dev, series\_group, emui\_dev, device\_name, device\_size.
3. **Ad Metadata**: task\_id, adv\_id, creat\_type\_cd, adv\_prim\_id, inter\_type\_cd, slot\_id, site\_id, spread\_app\_id.
4. **Behavioral Data**: ad\_click\_list\_v001, ad\_close\_list\_v001, u\_newsCatInterestsST, u\_refreshTimes, u\_feedLifeCycle.
5. **App Metadata**: hispace\_app\_tags, app\_second\_class, app\_score.
6. **Timestamp**: pt\_d.

How to Use This Information

* **Feature Engineering**: Use columns like ad\_click\_list and ad\_close\_list to create features like "number of clicks" or "number of closes."
* **Target Variable**: The label column is your target variable, so focus on understanding how other columns relate to it.
* **Categorical Data**: Columns like gender, city, and creat\_type\_cd are categorical and may need encoding (e.g., one-hot encoding).
* **Behavioral Insights**: Columns like u\_newsCatInterestsST and u\_feedLifeCycle can help you understand user preferences and activity levels.

Juyi Notes:

Feature Engineering thought process:

We wanted to determine the specific hour or time for each record, but due to inconsistencies in time zones, we cannot ensure accuracy. Instead, we began by identifying broader temporal patterns, such as the day of the week, the week of the month, and similar time-based groupings.

3 common mistakes in churn analysis

1. Seasonality
2. Value-based churn rate, how important is this user to the company
3. User segmentation

# Define categorical columns for different encoding strategies

one\_hot\_categorical\_columns = [

"age",

"gender",

"city\_rank",

"series\_group",

"net\_type",

"creat\_type\_cd",

"inter\_type\_cd",

"app\_score"

]

target\_encoding\_categorical\_columns = [

"residence",

"city",

"series\_dev",

"emui\_dev",

"device\_name",

"device\_size",

"task\_id",

"adv\_id",

"adv\_prim\_id",

"slot\_id",

"spread\_app\_id",

"hispace\_app\_tags",

"app\_second\_class"

]

# Copy the data and drop unnecessary columns

df = train\_ads.copy()

df.drop(columns=['site\_id', 'log\_id'], inplace=True, errors='ignore')

# Separate features and target variable

X = df.drop(columns=['label'])

y = df['label']

# Split the dataset into train and test sets (ensure stratification for balanced class distribution)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.2, stratify=y, random\_state=42

)

# ===============================

# Step 1: One-Hot Encoding for Low Cardinality Categorical Features

# ===============================

onehot\_encoder = OneHotEncoder(drop='first', sparse=False)

X\_train\_onehot = onehot\_encoder.fit\_transform(X\_train[one\_hot\_categorical\_columns])

X\_test\_onehot = onehot\_encoder.transform(X\_test[one\_hot\_categorical\_columns])

# Convert One-Hot Encoded features to DataFrame with correct column names

onehot\_feature\_names = onehot\_encoder.get\_feature\_names\_out(one\_hot\_categorical\_columns)

X\_train\_onehot\_df = pd.DataFrame(X\_train\_onehot, index=X\_train.index, columns=onehot\_feature\_names)

X\_test\_onehot\_df = pd.DataFrame(X\_test\_onehot, index=X\_test.index, columns=onehot\_feature\_names)

# Remove original categorical columns used for One-Hot Encoding

X\_train\_remaining = X\_train.drop(columns=one\_hot\_categorical\_columns)

X\_test\_remaining = X\_test.drop(columns=one\_hot\_categorical\_columns)

# Merge One-Hot Encoded features with the remaining dataset

X\_train\_processed = pd.concat([X\_train\_onehot\_df, X\_train\_remaining], axis=1)

X\_test\_processed = pd.concat([X\_test\_onehot\_df, X\_test\_remaining], axis=1)

# ===============================

# Step 2: K-Fold Target Encoding for High Cardinality Categorical Features

# ===============================

X\_train\_te = X\_train\_processed.copy()

X\_test\_te = X\_test\_processed.copy()

# Define number of folds for K-Fold Target Encoding

n\_splits = 5

kf = KFold(n\_splits=n\_splits, shuffle=True, random\_state=42)

global\_mean = y\_train.mean()

# Apply K-Fold Target Encoding for each high-cardinality categorical feature

for col in target\_encoding\_categorical\_columns:

# Create a new column for target encoding

X\_train\_te[col + '\_te'] = np.nan

# Perform K-Fold Encoding

for train\_index, valid\_index in kf.split(X\_train\_te):

X\_tr, X\_val = X\_train\_te.iloc[train\_index], X\_train\_te.iloc[valid\_index]

y\_tr = y\_train.iloc[train\_index]

# Compute mean target value for each category

mapping = y\_tr.groupby(X\_tr[col]).mean()

# Map the computed values to the validation fold

X\_train\_te.loc[X\_train\_te.index[valid\_index], col + '\_te'] = X\_val[col].map(mapping)

# Fill missing values (new categories) with the global mean

X\_train\_te[col + '\_te'].fillna(global\_mean, inplace=True)

# Encode the test set using the entire training data mapping

mapping\_full = y\_train.groupby(X\_train\_processed[col]).mean()

X\_test\_te[col + '\_te'] = X\_test\_te[col].map(mapping\_full)

X\_test\_te[col + '\_te'].fillna(global\_mean, inplace=True)

# Drop the original categorical columns used for Target Encoding

X\_train\_te.drop(columns=target\_encoding\_categorical\_columns, inplace=True)

X\_test\_te.drop(columns=target\_encoding\_categorical\_columns, inplace=True)

# ===============================

# Step 3: Compute and Visualize Feature Correlations with Label

# ===============================

# Add the target variable back to the training data for correlation analysis

train\_final = X\_train\_te.copy()

train\_final['label'] = y\_train

# Compute the correlation matrix for numerical features

corr\_matrix = train\_final.corr()

# Print feature correlations with the label, sorted by correlation strength

label\_corr = corr\_matrix['label'].drop('label').sort\_values(ascending=False)

print("Feature correlations with label:")

print(label\_corr)

# Plot the correlation heatmap

plt.figure(figsize=(12, 10))

sns.heatmap(corr\_matrix, annot=True, fmt=".2f", cmap="coolwarm")

plt.title("Feature Correlation Heatmap")

plt.show()