AVAZU CTR PREDICTION

Jiani XU

06/12/2024

Content

1. Objective and Context

Objective: use Random Forest to build a model that can predict user behavior effectively.

Describe: user-click records with various categorical and numerical features.

Data size: more than 4,000k rows
25 columns

	id	click	hour	C1	banner_pos	site_id	site_domain	site_category	app_id	app_domain	...	device_type	device_conn_type	C14	C15	C16	C17	C18	C19	C20	C21
0	3926754196077507072	1	14102100	1005	0	f84e52b6	d7e2f29b	28905ebd	ecad2386	7801e8d9	...	1	0	20346	300	250	2331	2	39	-1	23
1	4544181857823990784	1	14102100	1005	1	e151e245	7e091613	f028772b	ecad2386	7801e8d9	...	1	0	17037	320	50	1934	2	39	-1	16
2	12744445434029488128	0	14102100	1005	0	1fbe01fe	f3845767	28905ebd	ecad2386	7801e8d9	...	1	0	15699	320	50	1722	0	35	100083	79
3	4063799378381284352	0	14102100	1005	0	1fbe01fe	f3845767	28905ebd	ecad2386	7801e8d9	...	1	2	15699	320	50	1722	0	35	-1	79
4	6357775859397618688	1	14102100	1005	0	5b08c53b	7687a86e	3e814130	ecad2386	7801e8d9	...	1	0	17654	300	250	1994	2	39	100083	33
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
404285	9823785328714326016	0	14103023	1005	0	1fbe01fe	f3845767	28905ebd	ecad2386	7801e8d9	...	1	0	22254	320	50	2545	0	431	100084	221
404286	5655954353396606976	1	14103023	1005	1	17caea14	0dde25ec	f028772b	ecad2386	7801e8d9	...	1	0	21412	320	50	2467	2	167	100081	23
404287	5360327371237894144	0	14103023	1002	0	61a8c644	c4e18dd6	50e219e0	ecad2386	7801e8d9	...	0	0	21789	320	50	2512	2	291	-1	52
404288	8782688772576775168	1	14103023	1005	0	1fbe01fe	f3845767	28905ebd	ecad2386	7801e8d9	...	1	0	22254	320	50	2545	0	431	-1	221
404289	4657897218921320448	1	14103023	1005	0	1fbe01fe	f3845767	28905ebd	ecad2386	7801e8d9	...	1	0	22257	320	50	2545	0	431	100084	221

2. Data Processing

01. Select Data

```
# read the data in chunk

chunk_size = 100000
train_chunks = pd.read_csv('train.csv', chunksize=chunk_size)

for chunk in train_chunks:
    print(chunk.info(memory_usage='deep'))
    print(chunk.head())
    break
```

```
# random select
sample_ratio = 0.01  # select 1% each time

# sampling
sampled_data = pd.DataFrame()
for chunk in pd.read_csv('train.csv', chunksize=chunk_size):
    sampled_chunk = chunk.sample(frac=sample_ratio, random_state=42)
    sampled_data = pd.concat([sampled_data, sampled_chunk], ignore_index=True)

print(f"the size of sampled data: {sampled_data.shape}")
```

```
the size of sampled data: (404290, 24)
```

```python
# optimization

def optimize_memory(df):
    for col in df.columns:
        if df[col].dtype == 'int64' or df[col].dtype == 'float64':
            df[col] = pd.to_numeric(df[col], downcast='unsigned')
        elif df[col].dtype == 'object':
            df[col] = df[col].astype('category')
    return df

sampled_data = optimize_memory(sampled_data)

print(sampled_data.info(memory_usage='deep'))
```

```python
# save as a csv file
sampled_data.to_csv('train_sample.csv', index=False, compression='gzip')
```

```python
df = pd.read_csv('train_sample.csv', compression='gzip')
df
```

02. Process Data

Time

```
# get information of time

df['hour'] = pd.to_datetime(df['hour'].astype(str), format='%y%m%d%H')

df['day'] = df['hour'].dt.day

df['weekday'] = df['hour'].dt.weekday
df['is_weekend'] = df['hour'].dt.isocalendar().day >= 6

df['hour'] = df['hour'].dt.hour
df['is_morning'] = df['hour'].between(6, 12)
df['is_afternoon'] = df['hour'].between(12, 18)
df['is_evening'] = df['hour'].between(18, 24)

df['hour_sin'] = np.sin(2 * np.pi * df['hour'] / 24)
df['hour_cos'] = np.cos(2 * np.pi * df['hour'] / 24)
```

Combination Features

```
df['site_app_combination'] = df['site_category'] * 1000 + df['app_category']
df['device_interaction'] = df['device_type'] * 10 + df['device_conn_type']
```

Statistical

```
df['site_category_count'] = df.groupby('site_category')['click'].transform('count')
df['app_category_mean_click'] = df.groupby('app_category')['click'].transform('mean')
```

Ratio features and Logarithmic transformation

```
df['C14_log'] = np.log1p(df['C14'])
df['C15_C16_ratio'] = df['C15'] / (df['C16'] + 1)
```

3. Data Training

```python
# The basic model

ctr = RandomForestClassifier(
    n_estimators=100,
    max_depth=10,
    random_state=42,
    n_jobs=-1
)

ctr.fit(X_train, y_train)

y_pred = ctr.predict(X_test)
y_pred_proba = ctr.predict_proba(X_test)[:, 1]

auc = roc_auc_score(y_test, y_pred_proba)
print(f"AUC: {auc:.4f}")
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
AUC: 0.8150
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.99      0.92     67228
           1       0.84      0.20      0.33     13630

    accuracy                           0.86     80858
   macro avg       0.85      0.60      0.62     80858
weighted avg       0.86      0.86      0.82     80858
```

```python
# class weight

ctr_balanced = RandomForestClassifier(
    n_estimators=100,
    max_depth=10,
    random_state=42,
    n_jobs=-1,
    class_weight='balanced'
)

ctr_balanced.fit(X_train, y_train)

y_pred_balanced = ctr_balanced.predict(X_test)
y_pred_proba_balanced = ctr_balanced.predict_proba(X_test)[:, 1]

auc_balanced = roc_auc_score(y_test, y_pred_proba_balanced)
print(f"AUC: {auc_balanced:.4f}")
print("Classification Report:")
print(classification_report(y_test, y_pred_balanced))
```

```
AUC: 0.8163
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.64      0.76     67228
           1       0.32      0.82      0.46     13630

    accuracy                           0.67     80858
   macro avg       0.63      0.73      0.61     80858
weighted avg       0.84      0.67      0.71     80858
```

```python
# ROC
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba_balanced)
optimal_idx = np.argmax(tpr - fpr)
optimal_threshold = thresholds[optimal_idx]
print(f"The best: {optimal_threshold:.2f}")

y_pred_adjusted = (y_pred_proba_balanced >= optimal_threshold).astype(int)

print("Classification Report:")
print(classification_report(y_test, y_pred_adjusted))
```

```
The best: 0.52
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.66      0.78     67228
           1       0.33      0.80      0.46     13630

    accuracy                           0.69     80858
   macro avg       0.63      0.73      0.62     80858
weighted avg       0.84      0.69      0.73     80858
```

```python
feature_importances = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': ctr_balanced.feature_importances_
}).sort_values(by='Importance', ascending=False)

print("Feature Importance(Top 10):")
print(feature_importances.head(10))

low_importance_features = feature_importances[feature_importances['Importance'] < 0.01]['Feature'].tolist()
print(f"Low Feature Importance: {low_importance_features}")
```

```
Feature Importance(Top 10):
                     Feature  Importance
44        device_click_rate    0.330231
38          site_click_rate    0.164740
39           app_click_rate    0.109305
45             device_count    0.027800
40               site_count    0.026068
34  app_category_mean_click    0.020690
22                      C21    0.020470
10                device_id    0.019548
46      site_app_click_rate    0.019021
17                      C16    0.017736
Low Feature Importance: ['app_conn_combination', 'site_id', 'C14_log', 'C14', 'C18', 'device_interaction', 'C15', 'site_app_c
4_C17_product', 'C1', 'device_type', 'C20', 'C17', 'site_category', 'app_category', 'app_domain', 'C19', 'site_category_count
er_pos', 'device_model', 'id', 'device_ip', 'hour_sin', 'device_conn_type', 'device_type_conn_interaction', 'hour_cos', 'day'
'weekday', 'is_afternoon', 'is_morning', 'is_weekend']
```

```python
# remove the Low Feature Importance
X_train_optimized = X_train.drop(columns=low_importance_features)
X_test_optimized = X_test.drop(columns=low_importance_features)
```

```python
ctr_optimized = RandomForestClassifier(
    n_estimators=200,
    max_depth=10,
    random_state=42,
    class_weight='balanced',
    n_jobs=-1
)

ctr_optimized.fit(X_train_optimized, y_train)

# 预测
y_pred_optimized = ctr_optimized.predict(X_test_optimized)
y_pred_proba_optimized = ctr_optimized.predict_proba(X_test_optimized)[:, 1]

print("Classification Report:")
print(classification_report(y_test, y_pred_optimized))
print(f"AUC: {roc_auc_score(y_test, y_pred_proba_optimized):.4f}")
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.62      0.75     67228
           1       0.31      0.84      0.45     13630
```

```
time_features = ['hour_sin', 'hour_cos', 'hour', 'weekday', 'is_weekend']
X_train_restored = X_train_optimized.join(X_train[time_features])
X_test_restored = X_test_optimized.join(X_test[time_features])
```

```
ctr_restored = RandomForestClassifier(
    n_estimators=200,
    max_depth=10,
    random_state=42,
    class_weight='balanced',
    n_jobs=-1
)
ctr_restored.fit(X_train_restored, y_train)

y_pred_restored = ctr_restored.predict(X_test_restored)
y_pred_proba_restored = ctr_restored.predict_proba(X_test_restored)[:, 1]

print("Classification Report:")
print(classification_report(y_test, y_pred_restored))
print(f"AUC: {roc_auc_score(y_test, y_pred_proba_restored):.4f}")
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.64      0.76     67228
           1       0.32      0.82      0.46     13630

    accuracy                           0.67     80858
   macro avg       0.63      0.73      0.61     80858
weighted avg       0.84      0.67      0.71     80858

AUC: 0.8156
```

```
X_train_interactions = X_train_restored.copy()
X_test_interactions = X_test_restored.copy()

X_train_interactions['click_rate_interaction'] = X_train_interactions['device_click_rate'] * X_train_interactions['site_click_rate']
X_test_interactions['click_rate_interaction'] = X_test_interactions['device_click_rate'] * X_test_interactions['site_click_rate']
```

```
ctr_interactions = RandomForestClassifier(
    n_estimators=200,
    max_depth=10,
    random_state=42,
    class_weight='balanced',
    n_jobs=-1
)
ctr_interactions.fit(X_train_interactions, y_train)

y_pred_interactions = ctr_interactions.predict(X_test_interactions)
y_pred_proba_interactions = ctr_interactions.predict_proba(X_test_interactions)[:, 1]

print("Classification Report:")
print(classification_report(y_test, y_pred_interactions))
print(f"AUC: {roc_auc_score(y_test, y_pred_proba_interactions):.4f}")
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.64      0.76     67228
           1       0.32      0.82      0.46     13630

    accuracy                           0.67     80858
   macro avg       0.63      0.73      0.61     80858
weighted avg       0.84      0.67      0.71     80858

AUC: 0.8164
```

```
X_train_interactions = X_train_restored.copy()
X_test_interactions = X_test_restored.copy()

X_train_interactions['click_rate_interaction'] = X_train_interactions['device_click_rate'] * X_train_interactions['site_click_rate']
X_test_interactions['click_rate_interaction'] = X_test_interactions['device_click_rate'] * X_test_interactions['site_click_rate']
```

```
ctr_interactions = RandomForestClassifier(
    n_estimators=200,
    max_depth=10,
    random_state=42,
    class_weight='balanced',
    n_jobs=-1
)
ctr_interactions.fit(X_train_interactions, y_train)

y_pred_interactions = ctr_interactions.predict(X_test_interactions)
y_pred_proba_interactions = ctr_interactions.predict_proba(X_test_interactions)[:, 1]

print("Classification Report:")
print(classification_report(y_test, y_pred_interactions))
print(f"AUC: {roc_auc_score(y_test, y_pred_proba_interactions):.4f}")
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.64      0.76     67228
           1       0.32      0.82      0.46     13630

    accuracy                           0.67     80858
   macro avg       0.63      0.73      0.61     80858
weighted avg       0.84      0.67      0.71     80858

AUC: 0.8164
```

```python
rf_complex = RandomForestClassifier(
    n_estimators=300,
    max_depth=15,
    random_state=42,
    class_weight='balanced',
    n_jobs=-1
)
rf_complex.fit(X_train_interactions, y_train)

y_pred_complex = rf_complex.predict(X_test_interactions)
y_pred_proba_complex = rf_complex.predict_proba(X_test_interactions)[:, 1]


print("Classification Report:")
print(classification_report(y_test, y_pred_complex))
print(f"AUC: {roc_auc_score(y_test, y_pred_proba_complex):.4f}")
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.67      0.78     67228
           1       0.33      0.79      0.46     13630

    accuracy                           0.69     80858
   macro avg       0.63      0.73      0.62     80858
weighted avg       0.84      0.69      0.73     80858

AUC: 0.8161
```

```python
#validat by using RandomizedSearchCV

param_dist = {
    'n_estimators': [100, 200, 300, 400],
    'max_depth': [10, 15, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'class_weight': ['balanced']
}

random_search = RandomizedSearchCV(
    estimator=RandomForestClassifier(random_state=42, n_jobs=-1),
    param_distributions=param_dist,
    n_iter=10,  # 控制随机搜索的次数
    scoring='roc_auc',
    cv=3,
    verbose=2,
    random_state=42
)

random_search.fit(X_train_interactions, y_train)

best_rf = random_search.best_estimator_
print("The best parameters:")
print(random_search.best_params_)

y_pred_rs = best_rf.predict(X_test_interactions)
y_pred_proba_rs = best_rf.predict_proba(X_test_interactions)[:, 1]

from sklearn.metrics import classification_report, roc_auc_score
print("Classification Report:")
print(classification_report(y_test, y_pred_rs))
print(f"AUC: {roc_auc_score(y_test, y_pred_proba_rs):.4f}")
```

```
The best parameters:
{'n_estimators': 400, 'min_samples_split': 5, 'min_samples_leaf': 4, 'max_depth': 15, 'class_weight': 'balanced'}
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.67      0.78     67228
           1       0.33      0.79      0.46     13630

    accuracy                           0.69     80858
   macro avg       0.63      0.73      0.62     80858
weighted avg       0.84      0.69      0.73     80858

AUC: 0.8178
```

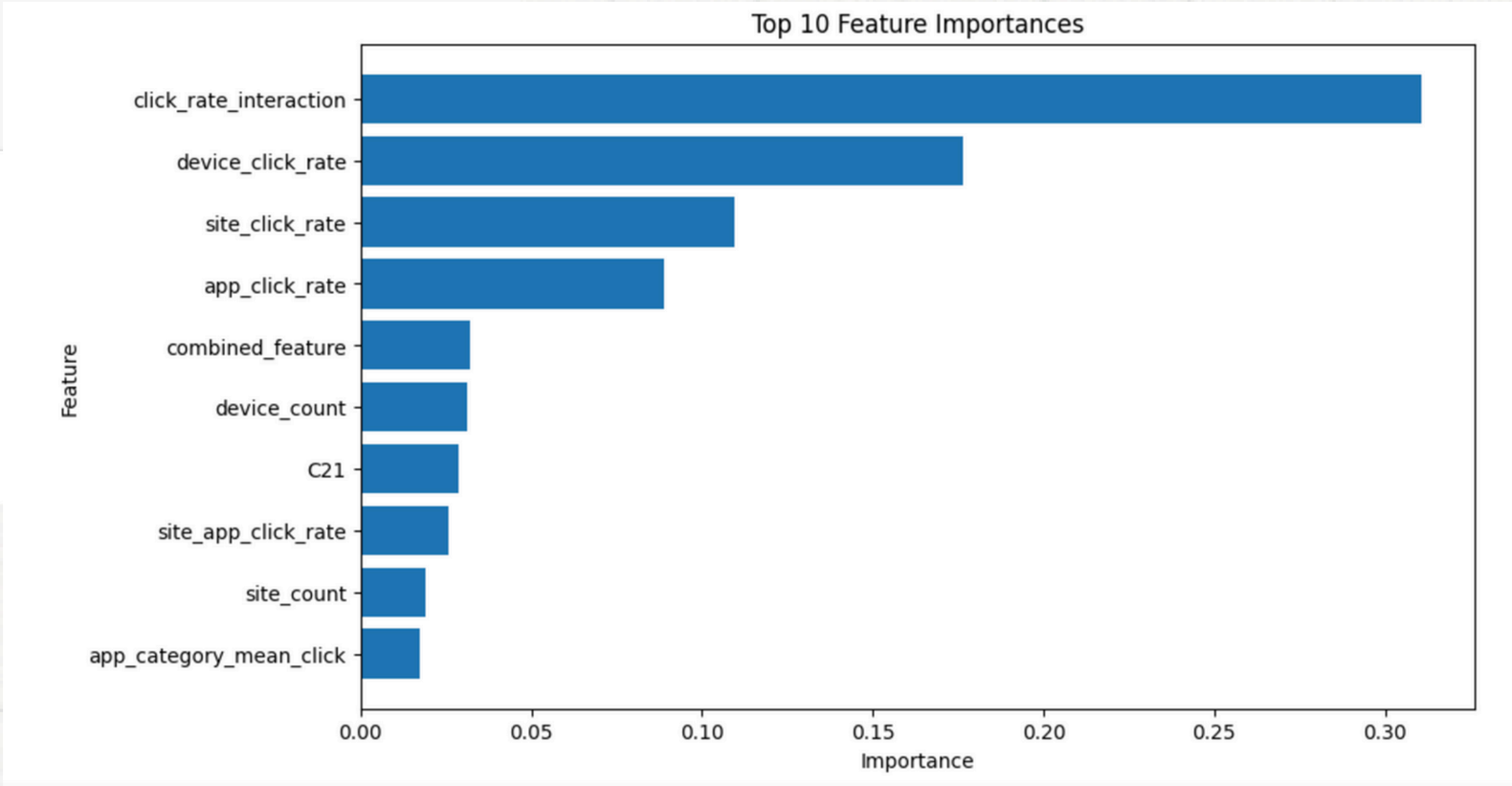
Model Interpretation

```
feature_importances = pd.DataFrame({
    'Feature': X_train_interactions.columns,
    'Importance': best_rf.feature_importances_
}).sort_values(by='Importance', ascending=False)

print(feature_importances.head(10))

plt.figure(figsize=(10, 6))
plt.barh(feature_importances['Feature'][:10], feature_importances['Importance'][:10])
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Top 10 Feature Importances')
plt.gca().invert_yaxis()
plt.show()
```

	Feature	Importance
22	click_rate_interaction	0.310644
12	device_click_rate	0.176517
8	site_click_rate	0.109475
9	app_click_rate	0.088704
15	combined_feature	0.032176
13	device_count	0.031367
4	C21	0.028850
14	site_app_click_rate	0.025695
10	site_count	0.018876
5	app_category_mean_click	0.017297



4. Demo

Thank you very much!