<Machine Learning Term Project - Flight Recommendation System>

Group C
202034916 Park Seonjun
202234911 Rhyu Yeonyi
202235118 Jung Soyeon

* Source code&Dataset

1. Project Overview

1.1 Background and Objectives

There are ongoing reports of passengers receiving unfair or unsatisfactory treatment from airlines. Since the in-flight experience strongly affects overall trip satisfaction, this project aims to develop a system that recommends the most suitable airline or flight for each user based on user ratings and service factors (e.g., seat comfort, cabin crew service).

1.2 Datasets

This project utilizes two airline review datasets

Dataset 1:

Source: Kaggle Airlines Review Dataset (airline.csv)

Samples: 41,396

Columns: 20

Key Fields: Author, airline_name, overall_rating

Dataset 2:

Source: Skytrax Reviews Dataset (airlines_reviews.csv)

Samples: 8,100

Columns: 17

Key Fields: Name, Airline, Overall Rating

2. Data Preprocessing

Steps 1. Column Removal: Removed columns absent in test data or irrelevant to similarity computation (review dates etc.)

Steps 2. Missing Values: Test dataset had no missing values from the start

[Train Data set] (Raw data & Result)

airline2.csv
Raw data

	Column	Missing Values
0	airline_name	0
1	link	0
2	title	0
3	author	0
4	author_country	1591
5	date	0
6	content	0
7	aircraft	40118
8	type_traveller	39018
9	cabin_flown	2876
10	route	39055
11	overall_rating	4535
12	seat_comfort_rating	7690
13	cabin_staff_rating	7688
14	food_beverages_rating	8132
15	inflight_entertainment_rating	10282
16	ground_service_rating	39193
17	wifi_connectivity_rating	40831
18	value_money_rating	1673
19	recommended	0

Samples: 41396

airline2.csv
Preprocessed

	Column	Missing Values
0	user_id	0
1	item_id	0
2	Class	0
3	Seat Comfort	0
4	Staff Service	0
5	Food & Beverages	0
6	Inflight Entertainment	0
7	Value For Money	0
8	Recommended	0
9	Overall Rating	0

Samples: 34853

[Test Data set] (Raw data & Result)

airline1.csv
Raw data

	Column	Missing Values
0	Title	0
1	Name	0
2	Review Date	0
3	Airline	0
4	Verified	0
5	Reviews	0
6	Type of Traveller	0
7	Month Flown	0
8	Route	0
9	Class	0
10	Seat Comfort	0
11	Staff Service	0
12	Food & Beverages	0
13	Inflight Entertainment	0
14	Value For Money	0
15	Overall Rating	0
16	Recommended	0

Samples: 8100

airline1.csv
Preprocessed

	Column	Missing Values
0	user_id	0
1	item_id	0
2	Class	0
3	Seat Comfort	0
4	Staff Service	0
5	Food & Beverages	0
6	Inflight Entertainment	0
7	Value For Money	0
8	Recommended	0
9	Overall Rating	0

Samples: 8100

3. Methodology: (1) Hybrid User-Based Collaborative Filtering

3.1 Algorithm Overview

This project employs a Hybrid User-Based Collaborative Filtering approach with the following characteristics:

1. Only consider users who appear more than three times: To ensure reliable recommendations, only users with at least 3 rating records are included in the analysis
2. Model knows other users' scores and recommendation status: The model has prior knowledge of other users' ratings and recommendations (Warm Start scenario)
3. Users provide one of their own evaluation records: Users must submit one of their rating records as input
4. Compute similarity by comparing the provided records with the records of other users: Similarity is calculated between the user's provided record and records from other users
5. Provide airline recommendations to user: Predict whether an item that a user has rated will be recommended, and measure performance based on this prediction accuracy

3.2 Core Algorithm Steps

Step 1. User Preference Learning : For each training user, a personalized preference weight vector is learned using Ridge Regression.

```
def train_and_save(df: pd.DataFrame, out_prefix: str,
                   k: int, shrink: float, ridge: float, folds: int):
    model = HybridUserCFNoProfile(
        neighbour_k=k, shrink_alpha=shrink, ridge_alpha=ridge, content_weight=0.0
    )
    # 1) 스케일러: train의 raw factor 전체로 적합
    model.fit_scaler_on_factors(df)
    # 2) 사용자 선호(가중치) 학습
    model.fit_user_preferences_train(df)
    # 3) K-fold로 Recommended 분류기 학습(Δ/ρ)
    model.train_classifier_with_kfold(df, n_splits=folds)

    out_path = f"{out_prefix}_{k}_{shrink}_{ridge}.pkl"
    with open(out_path, "wb") as f:
        cloudpickle.dump(model, f)
    print(f"[OK] Saved: {out_path}")
```

```python
# -------------------- Train user preferences (no item-profile) --------------------
def fit_user_preferences_train(self, df: pd.DataFrame):
    """
    Per-user Ridge: y = b_u + w_u · (scaled raw factor vector of that row).
    """
    self.global_b = float(df[TARGET_COL_R].mean()) if len(df) else 0.0

    user_w, user_b, user_n = {}, {}, {}
    Xw_all, yw_all = [], []

    for uid, g in tqdm(df.groupby("user_id"), desc="Train: user preferences", total=df["user_id"].nunique()
        X = self._tx(g[FACTOR_COLS].to_numpy(dtype=float))
        y = g[TARGET_COL_R].to_numpy(dtype=float)

        if len(y) >= 2:
            m = Ridge(alpha=self.ridge_alpha, fit_intercept=True, random_state=RANDOM_SEED)
            m.fit(X, y)
            w = m.coef_.astype(float); b = float(m.intercept_)
        elif len(y) == 1:
            w = np.zeros(len(FACTOR_COLS), dtype=float); b = float(y[0])
        else:
            w = np.zeros(len(FACTOR_COLS), dtype=float); b = 0.0

        user_w[uid], user_b[uid], user_n[uid] = w, b, int(len(y))
        if len(y) >= 2:
            Xw_all.append(X); yw_all.append(y)

    if Xw_all:
        Xgl = np.vstack(Xw_all); ygl = np.concatenate(yw_all)
        m = Ridge(alpha=self.ridge_alpha, fit_intercept=True, random_state=RANDOM_SEED)
        m.fit(Xgl, ygl)
        self.global_w = m.coef_.astype(float)
    else:
        self.global_w = np.zeros(len(FACTOR_COLS), dtype=float)

    self.user_w_train, self.user_b_train, self.user_n_train = user_w, user_b, user_n
```

- Each training user learns how much weight to give to the six service factors (Class, Seat Comfort, Staff Service, Food & Beverages, Inflight Entertainment, Value For Money).
- w_u (weight vector): Represents user preference patterns, e.g., "User A values seat comfort, User B values food."
- b_u (bias/intercept): The user's rating tendency (whether they tend to give high or low ratings).
- global_w, global_b: The average preferences of all users.

Step 2. Test User Preference Estimation (1-Shot Learning)

```python
if x_seed is not None:
    denom = self.ridge_alpha + float(x_seed.dot(x_seed))
    adj = (y_seed - b_prior - float(w_prior.dot(x_seed))) / denom
    w_u = w_prior + adj * x_seed
    b_u = (b_prior*self.ridge_alpha + y_seed) / (self.ridge_alpha + 1.0)
else:
    w_u, b_u = w_prior, b_prior
```

- The test user is a new user not present in the training data. Their preferences are estimated using only a single rating (seed).
- It predicts, "This test user is likely to have the following preferences."

Step 3. Neighbor Discovery with Shrinkage

```python
# 이웃(top-K, shrink)
neigh = []
for v, wv in Wn.items():
    if v == uid:
        continue
    s = float(w_u.dot(wv))
    if s <= 0: # 유사도가 0 이하인 사용자 제외
        continue
    s *= _shrink(user_n_f.get(v,0), self.shrink_alpha) * _shrink(1, self.shrink_alpha) # 나머지 사용자에 대해 이웃 숫자로 shrink 설정
    if s > 0:
        neigh.append((v, s)) #userId와 shrink 반영된 유사도 저장
neigh.sort(key=lambda x: -x[1])
neigh = neigh[: self.neighbour_k]
```

- Reliable neighbors are found using cosine similarity and shrinkage.
- Shrinkage meaning:
  (1) Users with many ratings → higher reliability → larger weight
  (2) Users with few ratings → lower reliability → smaller weight
      * Example: For item B, User1 has 2 ratings, User3 has 1 rating.
      Similarity between User2 and User1 = 70%
      Similarity between User2 and User3 = 70%
      After applying shrinkage: User1 > User3 (users with more data are considered more reliable).
- Determines "how influential this user is for recommendations."

Step 4. Collaborative Filtering Features

```
for v, s in neigh:
    if (v, iid) in rating_label:
        den += abs(s)
        num += s * (1 if rec_label[(v,iid)]==1 else 0)
        # y=0 → r0_v ≈ b_v (x_i 불필요)
        r0_v = user_b_f.get(v, 0.0)
        num_res += s * (rating_label[(v,iid)] - r0_v)
        den_res += abs(s)
rho = (num/den) if den>0 else 0.0
delta = (num_res/den_res) if den_res>0 else 0.0
```

- Rho (ρ): Recommendation ratio. How much did similar users recommend this item? – "How similar must a user be for their recommendation preferences to align with mine?"
- Delta (Δ): Rating residual. Meaning: adjustment considering users' rating tendencies (strict vs. lenient).
- r0_v: User's overall rating mean → baseline rating tendency.

Step5. Recommendation Classifier

```
self.clf = LogisticRegression(
    max_iter=300, solver="lbfgs", class_weight="balanced", random_state=RANDOM_SEED
)
self.clf.fit(Xf, Y)
```

The previously calculated features are fed into a Logistic Regression model to predict the final recommendation outcome.

Step6. Evaluation

```
# compute metrics (micro)
tp = sum(1 for yt, yp in zip(all_y_true_cls, all_y_pred_cls) if yt==1 and yp==1)
fp = sum(1 for yt, yp in zip(all_y_true_cls, all_y_pred_cls) if yt==0 and yp==1)
fn = sum(1 for yt, yp in zip(all_y_true_cls, all_y_pred_cls) if yt==1 and yp==0)
tn = sum(1 for yt, yp in zip(all_y_true_cls, all_y_pred_cls) if yt==0 and yp==0)
prec = tp / max(tp+fp, 1)
rec  = tp / max(tp+fn, 1)
acc  = (tp+tn) / max(tp+tn+fp+fn, 1)
f1   = (2*prec*rec) / max(prec+rec, 1e-12)
```

- The model's predictions are compared with the items the test users actually recommended to measure performance.
- Metrics: Precision, Recall, Accuracy, F1.

[Result]

Model	Precision	Recall	Accuracy	F1	Samples
hybrid_user_100_100.0_1.0.pkl	0.9054	0.7546	0.7129	0.8232	6332
hybrid_user_100_20.0_1.0.pkl	0.9049	0.7549	0.7127	0.8231	6332
hybrid_user_100_50.0_1.0.pkl	0.9048	0.7525	0.7107	0.8216	6332
hybrid_user_50_100.0_1.0.pkl	0.9109	0.7551	0.7178	0.8257	6332
hybrid_user_50_20.0_1.0.pkl	0.9123	0.7735	0.7336	0.8372	6332
hybrid_user_50_50.0_1.0.pkl	0.9116	0.7630	0.7246	0.8307	6332

3. Methodology: (2) Content-Based

3.1 Algorithm Overview

This project employs a Content-Based approach with the following characteristics:

1. Build item profiles from users' aspect ratings: Item profiles are created from users' aspect ratings and are smoothed using Empirical Bayes to ensure reliability.
2. Users provide one of their own evaluation records: Users must submit one of their rating records as input.
3. Model knows other users' scores and recommendation status: The model has prior knowledge of other users' ratings and recommendations (Warm Start scenario).
4. Compute item similarity as the cosine similarity: Item similarity is calculated as the cosine similarity between the constructed item profiles.
5. Provide airline recommendations to user: Predict whether an item that a user has rated will be recommended, and measure performance based on this prediction accuracy.

3.2 Core Algorithm Steps

Step 1. Item Profile Construction & Normalization (Scaled + EB Smoothing): The feature vector (service factors) for each item (flight/airline) is constructed.

```
for iid in sum_w.keys():
    if sum_w[iid] > 0:
        m_i = sum_ (variable) iid: Any
        item_mean[iid] = m_i
        item_n[iid] = int((all_item_ids == iid).sum())
    else:
        continue

# 4) Empirical Bayes smoothing in scaled space
profiles = {}
a = float(self.alpha)
for iid, m_i in item_mean.items():
    n_i = item_n[iid]
    prof = (n_i/(n_i+a))*m_i + (a/(n_i+a))*self.mu_vec
    profiles[iid] = prof.astype(float)

self.item_profiles = profiles
self.item_counts   = item_n
```

- The entire set of raw factors (ratings) is first transformed into a common scale space

using Standard Scaler.

- A weighted mean vector is computed for each item, using user ratings as weights.
- Empirical Bayes Smoothing is applied: items with few reviews are smoothed toward the global mean, while items with many reviews are dominated by their own mean rating to enhance profile reliability.

Step 2. Item Bias Construction: The mean rating (Item Bias) for each item (airline/flight) is calculated. This value serves as one of the input variables for the Rating Calibration Regression model in Step 4.

```
def build_item_bias(self, df: pd.DataFrame) -> None:
    """아이템 평균 평점 (회귀 입력 feature로 사용)."""
    self.item_bias = df.groupby("item_id")[TARGET_COL_R].mean().to_dict()
```

Step 3. Calibration Data Pair Generation: Pairs of items rated by the same user are extracted in the format (Seed Item, Target Item) to generate a dataset for regression training.

```
for seed in items:
    x_s = self.item_profiles.get(int(seed))
    if x_s is None:
        continue
    seed_rating = r_by_item[int(seed)]
    for target in items:
        if target == seed:
            continue
        x_i = self.item_profiles.get(int(target))
        if x_i is None:
            continue
        sim = self._cos(x_s, x_i)
        ib  = float(self.item_bias.get(int(target), np.nan))
        if np.isnan(ib):
            continue
        y   = r_by_item[int(target)]
        X_list.append([sim, seed_rating, ib])
        y_list.append(y)
```

- Each pair is formatted as $X = [Sim(Seed, Target), Rating(Seed), Bias(Target)]$ with the target being $y = \text{Rating(Target)}$
- Sim(Seed, Target) is the cosine similarity between the two item profiles.

Step 4. Rating Calibration Regression Training: A small Linear Regression (Ridge) model is trained using the data generated in Step 3.

```
# 5) 작은 회귀 학습: r_hat = a*sim + b*seed + c*item_bias + d
reg = LinearRegression()
reg.fit(X, y)
model.calib_coef = reg.coef_.astype(float)          # [a,b,c]
model.calib_intercept = float(reg.intercept_)       # d
```

- This regression model uses similarity (Sim), Seed rating, and Item Bias to calibrate the predicted rating ($\hat{r}$) of the Target item.
- The prediction follows the form: $\hat{r} = a * Sim + b * Rating(Seed) + c * Bias(Target) + d$ with the output clipped to the [1, 10] range.

Step5. Recommendation Threshold Learning (Global τ): The model learns a global

threshold(τ) that maximizes the F1 Score to determine the final recommendation outcome (Recommended=1 or 0)

```python
# 내부 헬퍼: 주어진 score/label에 대해 best τ 찾기
def _search_best_tau(scores_np: np.ndarray,
                     labels_np: np.ndarray,
                     grid_vals: np.ndarray):
    best = (-1.0, 0.0, 0.0, 0.0, None)  # (f1, prec, rec, n, tau)
    labels_list = labels_np.tolist()
    for tau in grid_vals:
        y_pred = (scores_np >= tau).astype(int).tolist()
        f1, prec, rec, n = self._f1_from_preds(labels_list, y_pred)
        if f1 > best[0]:
            best = (f1, prec, rec, n, float(tau))
    return best
```

- The model uses either the cosine similarity $(Sim)(sim - mode)$ or the calibrated rating($\hat{r}$) (Rating-mode) as the prediction score15.
- It analyzes whether recommended items are more similar than items that are not recommended to set τ.
- The model also learns separate pattern thresholds ($\tau_{pose}\ and\ T_{neg}$) based on the Recommended status of the Seed item (1 or 0).

Step6. Evaluation

```python
# compute metrics (micro)
tp = sum(1 for yt, yp in zip(all_y_true_cls, all_y_pred_cls) if yt==1 and yp==1)
fp = sum(1 for yt, yp in zip(all_y_true_cls, all_y_pred_cls) if yt==0 and yp==1)
fn = sum(1 for yt, yp in zip(all_y_true_cls, all_y_pred_cls) if yt==1 and yp==0)
tn = sum(1 for yt, yp in zip(all_y_true_cls, all_y_pred_cls) if yt==0 and yp==0)
prec = tp / max(tp+fp, 1)
rec  = tp / max(tp+fn, 1)
acc  = (tp+tn) / max(tp+tn+fp+fn, 1)
f1   = (2*prec*rec) / max(prec+rec, 1e-12)
```

- The model's predictions (Recommended/Not Recommended) are compared against the test users' actual recommendation status to measure performance.
- Metrics: Precision, Recall, Accuracy, and F1 Score were used.

[Result]

Model	Precision	Recall	Accuracy	F1	Samples
content_based.pkl	0.8865	0.8983	0.8081	0.8924	6332

4. Limitation Analysis

The Hybrid User-Based Collaborative Filtering and Content-Based Filtering methodologies employed in this project have the following key limitations:

1) Warm Start Only (Dependence on Seed Rating)
   - Both models are designed to operate exclusively in a Warm Start scenario, requiring the user to provide at least one evaluation record (Seed Rating) to estimate their preferences.

- This imposes a fundamental limitation: the system cannot provide recommendations to Cold Start users—new users with absolutely no prior rating history.

2) Global Decision Boundary
   - The recommendation decision relies on a Global Decision Threshold (τ or Logistic Regression boundary) applied uniformly across all users. This approach fails to account for individual differences in recommendation sensitivity, restricting the depth of personalization.
3) Inability to Directly Reuse Learned Similarities
   - The internal parameters (user preferences, item profiles) learned during training cannot be incrementally updated or reused when new data arrives. This forces the model to be recalibrated or fully retrained frequently, significantly increasing the operational cost and limiting real-time scalability in a dynamic environment.