recipe_recommender_hybrid_model-Copy1

November 30, 2023

[51]:
```
import numpy as np
import pandas as pd
import pickle

# Load the datasets
ingr_map = pd.read_pickle("ingr_map.pkl")
raw_recipes = pd.read_csv("RAW_recipes.csv")
raw_interactions = pd.read_csv("RAW_interactions.csv")
pp_users = pd.read_csv("PP_users.csv")
pp_recipes = pd.read_csv("PP_recipes.csv")
interactions_validation = pd.read_csv("interactions_validation.csv")
interactions_train = pd.read_csv("interactions_train.csv")
interactions_test = pd.read_csv("interactions_test.csv")


# Show some basic information about each dataset
datasets = {
    'ingr_map': ingr_map,
    'raw_recipes': raw_recipes,
    'raw_interactions': raw_interactions,
    'pp_users': pp_users,
    'pp_recipes': pp_recipes,
    'interactions_validation': interactions_validation,
    'interactions_train': interactions_train,
    'interactions_test': interactions_test
}

info_dict = {}

for name, dataset in datasets.items():
    info_dict[name] = {
        'Number of Rows': dataset.shape[0],
        'Number of Columns': dataset.shape[1],
        'Columns': ', '.join(dataset.columns)
    }

info_df = pd.DataFrame(info_dict).T
```

```
info_df
```

[51]:

```
                        Number of Rows  Number of Columns  \
ingr_map                         11659                  7
raw_recipes                     231637                 12
raw_interactions               1132367                  5
pp_users                         25076                  6
pp_recipes                      178265                  8
interactions_validation           7023                  6
interactions_train              698901                  6
interactions_test                12455                  6

                                                                   Columns
ingr_map                 raw_ingr, raw_words, processed, len_proc, repl…
raw_recipes              name, id, minutes, contributor_id, submitted, …
raw_interactions                  user_id, recipe_id, date, rating, review
pp_users                 u, techniques, items, n_items, ratings, n_ratings
pp_recipes               id, i, name_tokens, ingredient_tokens, steps_t…
interactions_validation             user_id, recipe_id, date, rating, u, i
interactions_train                  user_id, recipe_id, date, rating, u, i
interactions_test                   user_id, recipe_id, date, rating, u, i
```

[22]:

```
raw_recipes_df = pd.read_csv('RAW_recipes.csv')
raw_interactions_df = pd.read_csv("RAW_interactions.csv")
```

1 SVD collaborative filtering

[23]:

```
from surprise import Dataset, Reader
from surprise import SVD
from surprise.model_selection import cross_validate
from surprise.model_selection import train_test_split
from surprise import accuracy
```

[24]:

```
data = raw_interactions_df[['user_id', 'recipe_id', 'rating']]
```

[5]:

```
reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(data, reader)
model = SVD()

cross_validation_results = cross_validate(model, data, measures=['RMSE',␣
↪'MAE'], cv=5, verbose=True)
print(cross_validation_results)
```

```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.2236  1.2252  1.2139  1.2206  1.2193  1.2205  0.0039
```

```
MAE (testset)     0.7417  0.7416  0.7362  0.7394  0.7397  0.7397  0.0020
Fit time          12.16   12.13   12.30   12.57   11.94   12.22   0.21
Test time         1.44    1.37    1.32    1.41    1.67    1.44    0.12
{'test_rmse': array([1.22363968, 1.22522626, 1.21388836, 1.22058748,
1.21926462]), 'test_mae': array([0.74166363, 0.7416333 , 0.73620976, 0.73935663,
0.73973383]), 'fit_time': (12.156907081604004, 12.127675294876099,
12.29780387878418, 12.570320844650269, 11.936063051223755), 'test_time':
(1.437319040298462, 1.3681721687316895, 1.316875696182251, 1.4091508388519287,
1.6688730716705322)}
```

[6]:
```
# Split the data into training and test set (e.g., 75% training, 25% testing)
trainset, testset = train_test_split(data, test_size=0.25)

# Train the model on the training set
model = SVD()
model.fit(trainset)

# Make predictions on the test set
predictions = model.test(testset)

# Compute and print the accuracy metrics
rmse = accuracy.rmse(predictions)
mae = accuracy.mae(predictions)
```

```
RMSE: 1.2214
MAE:  0.7404
```

2 Hyperparameter tuning for SVD collaborative filtering

[25]:
```
from surprise import SVD
from surprise.model_selection import GridSearchCV
from surprise import Dataset, Reader
```

[7]:
```
# Define the parameter grid
param_grid = {
    'n_factors': [50, 100, 150],
    'n_epochs': [20, 30, 40],
    'lr_all': [0.002, 0.005],
    'reg_all': [0.02, 0.1]
}

# Setup grid search
gs = GridSearchCV(SVD, param_grid, measures=['rmse'], cv=3)

# Load the dataset
reader = Reader(rating_scale=(1, 5))
```

```
data = Dataset.load_from_df(raw_interactions_df[['user_id', 'recipe_id',␣
  ↪'rating']], reader)

# Run grid search
gs.fit(data)

print("Best RMSE score: ", gs.best_score['rmse'])
print("Best parameters: ", gs.best_params['rmse'])
```

```
Best RMSE score:  1.2161208333623452
Best parameters:  {'n_factors': 50, 'n_epochs': 20, 'lr_all': 0.005, 'reg_all':
0.1}
```

3 Using the Best Parameters to Train the SVD collaborative filtering Model

[26]:
```
# Setup the SVD model with the best parameters
optimized_SVD = SVD(n_factors=50, n_epochs=20, lr_all=0.005, reg_all=0.1)

reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(raw_interactions_df[['user_id', 'recipe_id',␣
  ↪'rating']], reader)

# Split your dataset into train and test sets
trainset, testset = train_test_split(data, test_size=0.25)

# Train the model on the trainset
optimized_SVD.fit(trainset)

# Make predictions on the testset
predictions = optimized_SVD.test(testset)

# Calculate and print the RMSE on the test set
rmse = accuracy.rmse(predictions)
mae = accuracy.mae(predictions)
```

```
RMSE: 1.2155
MAE:  0.7407
```

[28]:
```
cross_validation_optimized_SVD = cross_validate(optimized_SVD, data,␣
  ↪measures=['RMSE', 'MAE'], cv=5, verbose=True)
print(cross_validation_optimized_SVD)
```

```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.2099  1.2150  1.2124  1.2163  1.2117  1.2130  0.0023
```

```
MAE (testset)     0.7381  0.7407  0.7399  0.7421  0.7398  0.7401  0.0013
Fit time          9.98    9.80    9.83    10.45   10.56   10.12   0.32
Test time         2.78    2.39    2.51    2.52    2.54    2.55    0.13
{'test_rmse': array([1.20986348, 1.21495517, 1.21239871, 1.21627694,
1.21166121]), 'test_mae': array([0.73814761, 0.74072785, 0.73988309, 0.74213283,
0.73976056]), 'fit_time': (9.978832960128784, 9.798351049423218,
9.83065915107727, 10.449662208557129, 10.555390119552612), 'test_time':
(2.7790751457214355, 2.3928768634796143, 2.5148427486419678, 2.5208659172058105,
2.5412588119506836)}
```

4 Co-clustering Collaborative Filtering Model

[6]:
```python
from surprise import CoClustering

reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(raw_interactions_df[['user_id', 'recipe_id',␣
↪'rating']], reader)

trainset, testset = train_test_split(data, test_size=0.25)
co_clustering_model = CoClustering()
co_clustering_model.fit(trainset)
```

[6]:
```
<surprise.prediction_algorithms.co_clustering.CoClustering at 0x7fac2163fa00>
```

[7]:
```python
predictions = co_clustering_model.test(testset)
rmse = accuracy.rmse(predictions)
mae = accuracy.mae(predictions)
```

```
RMSE: 1.3097
MAE:  0.7562
```

[8]:
```python
param_grid = {'n_cltr_u': [3, 5, 7], 'n_cltr_i': [3, 5, 7], 'n_epochs': [20,␣
↪30, 40]}
gs = GridSearchCV(CoClustering, param_grid, measures=['rmse', 'mae'], cv=3)
gs.fit(data)


print("Best RMSE score: ", gs.best_score['rmse'])
print("Best parameters: ", gs.best_params['rmse'])
```

```
Best RMSE score:  1.3103096749322833
Best parameters:  {'n_cltr_u': 3, 'n_cltr_i': 5, 'n_epochs': 20}
```

[9]:
```python
# Initialize the Co-clustering model with the best parameters
model = CoClustering(n_cltr_u=3, n_cltr_i=5, n_epochs=20)

# Train the model on the entire dataset
trainset, testset = train_test_split(data, test_size=0.25)
```

```python
model.fit(trainset)
```

[9]: <surprise.prediction_algorithms.co_clustering.CoClustering at 0x7fac2165e130>

[10]:
```python
predictions = model.test(testset)
rmse = accuracy.rmse(predictions)
mae = accuracy.mae(predictions)
```

```
RMSE: 1.3041
MAE:  0.7479
```

5 Content-based Recommendation Model

[13]:
```python
# Get Tags for Content-based features
import ast

recipe_data = pd.read_csv("RAW_recipes.csv")

# Parsing the tags from string representation of list to actual list
recipe_data['tags'] = recipe_data['tags'].apply(ast.literal_eval)

# Exploring the unique tags and their frequencies
all_tags = [tag for sublist in recipe_data['tags'] for tag in sublist]
unique_tags = set(all_tags)
tag_frequency = pd.Series(all_tags).value_counts()

num_unique_tags = len(unique_tags)
selected_indices = [4,5,8, 11] + list(range(13, 61))
selected_tags = tag_frequency.iloc[selected_indices]



num_unique_tags, selected_tags
```

[13]:
```
(552,
 dietary                165091
 easy                   126062
 low-in-something        85776
 60-minutes-or-less      69990
 meat                    56042
 30-minutes-or-less      55077
 vegetables              53814
 taste-mood              52143
 4-hours-or-less         49497
 north-american          48479
 3-steps-or-less         44933
 15-minutes-or-less      43934
 low-sodium              43349
```

```
desserts                   43203
low-carb                   42189
healthy                    40340
dinner-party               37561
low-cholesterol            36743
low-calorie                36429
vegetarian                 35651
beginner-cook              35561
5-ingredients-or-less      35466
holiday-event              34920
inexpensive                32619
low-protein                32522
low-saturated-fat          31378
fruit                      31324
oven                       31180
american                   31179
eggs-dairy                 30142
pasta-rice-and-grains      27084
kid-friendly               27074
side-dishes                26902
healthy-2                  26619
comfort-food               26136
european                   24912
presentation               24470
poultry                    24160
lunch                      23800
for-1-or-2                 23084
low-fat                    22170
stove-top                  22095
seasonal                   21933
weeknight                  20948
chicken                    20381
appetizers                 20379
brunch                     18927
to-go                      18524
for-large-groups           17391
beef                       17074
one-dish-meal              16807
cheese                     15147
Name: count, dtype: int64)
```

[14]:

```
# One-hot encoding tags to speed up computation
top_tags = selected_tags.index.tolist()

# Initializing columns for top tags with default value 0
for tag in top_tags:
    recipe_data[f'tag_{tag}'] = 0
```

```
# Setting the value to 1 if the recipe contains the tag
for index, row in recipe_data.iterrows():
    for tag in top_tags:
        if tag in row['tags']:
            recipe_data.at[index, f'tag_{tag}'] = 1
```

[15]:
```
# Merging the user ratings data (interactions_data) with the one-hot encoded␣
 ↪tags from recipe_data
# The merging key will be 'recipe_id'

interactions_data = pd.read_csv("RAW_interactions.csv")

# Selecting relevant columns from recipe_data (recipe_id and one-hot encoded␣
 ↪tags)
recipe_tags_data = recipe_data[['id'] + [col for col in recipe_data.columns if␣
 ↪col.startswith('tag_')]]

# Renaming 'id' column to 'recipe_id' for consistency
recipe_tags_data.rename(columns={'id': 'recipe_id'}, inplace=True)

# Merging the datasets
merged_data = interactions_data.merge(recipe_tags_data, how='left',␣
 ↪on='recipe_id')
```

```
/var/folders/bg/hdj9jw_j33g1vg7x_rmvr9lc0000gn/T/ipykernel_10539/1372912005.py:1
0: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  recipe_tags_data.rename(columns={'id': 'recipe_id'}, inplace=True)
```

[16]:
```
# Filling missing values in tag columns with zeros
merged_data.fillna({col: 0 for col in merged_data.columns if col.
 ↪startswith('tag_')}, inplace=True)

# Checking the first few rows of the updated merged dataset
merged_data.head()
```

[16]:

	user_id	recipe_id	date	rating	\
0	38094	40893	2003-02-17	4	
1	1293707	40893	2011-12-21	5	
2	8937	44394	2002-12-01	4	
3	126440	85009	2010-02-27	5	
4	57222	85009	2011-10-01	5	

```
                                              review  tag_dietary  tag_easy  \
0  Great with a salad. Cooked on top of stove for…            1         1
1  So simple, so delicious! Great for chilly fall…            1         1
2  This worked very well and is EASY.  I used not…            1         1
3  I made the Mexican topping and took it to bunk…            0         1
4  Made the cheddar bacon topping, adding a sprin…            0         1

   tag_low-in-something  tag_60-minutes-or-less  tag_meat  …  tag_seasonal  \
0                     0                       0         0  …             0
1                     0                       0         0  …             0
2                     0                       0         0  …             1
3                     0                       0         0  …             0
4                     0                       0         0  …             0

   tag_weeknight  tag_chicken  tag_appetizers  tag_brunch  tag_to-go  \
0              1            0               0           0          0
1              1            0               0           0          0
2              0            0               0           0          1
3              0            0               0           0          0
4              0            0               0           0          0

   tag_for-large-groups  tag_beef  tag_one-dish-meal  tag_cheese
0                     0         0                  0           0
1                     0         0                  0           0
2                     1         0                  0           0
3                     0         0                  0           0
4                     0         0                  0           0

[5 rows x 57 columns]
```

[35]:
```
merged_data2= merged_data
```

6 Hyperparameter Tuning for Content-based Model

[17]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor

# Prepare the dataset for content-based model
X = merged_data.drop(columns=['user_id', 'recipe_id', 'rating', 'review',␣
 ↪'date'])
y = merged_data['rating']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Train RandomForest model
```

```
content_model = RandomForestRegressor()
content_model.fit(X_train, y_train)

# Make predictions on the test set
predictions_content = content_model.predict(X_test)
```

[18]:
```
from sklearn.metrics import mean_squared_error, mean_absolute_error

mse = mean_squared_error(y_test, predictions_content)
print("Mean Squared Error (MSE):", mse)

rmse = mean_squared_error(y_test, predictions_content, squared=False)
print("Root Mean Squared Error (RMSE):", rmse)

mae = mean_absolute_error(y_test, predictions_content)
print("Mean Absolute Error (MAE):", mae)
```

```
Mean Squared Error (MSE): 1.6841104017151067
Root Mean Squared Error (RMSE): 1.2977327928796076
Mean Absolute Error (MAE): 0.8311654118787669
```

[ ]:
```
#Optimizing hyperparameters
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor

# Define the parameter grid
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, 30],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

# Create a base model
rf = RandomForestRegressor()

# Instantiate the grid search model
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=3,␣
 ↪n_jobs=-1, verbose=2)

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Best parameters
best_params = grid_search.best_params_
print("Best parameters:", best_params)
```

Best parameters: {'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 300}

[40]:
```python
# Create a new model with the best parameters
optimized_rf = RandomForestRegressor(max_depth=10, min_samples_leaf=2,␣
 ↪min_samples_split=5, n_estimators=300)

# Retrain the model on the entire training set
optimized_rf.fit(X_train, y_train)

# Predict on the test set
y_pred = optimized_rf.predict(X_test)

# Calculate evaluation metrics
mse = mean_squared_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
mae = mean_absolute_error(y_test, y_pred)

print("MSE:", mse)
print("RMSE:", rmse)
print("MAE:", mae)
```

```
MSE: 1.6006633320865211
RMSE: 1.2651732419263857
MAE: 0.8456339830113258
```

[29]:
```python
# Running cross-validation

from sklearn.model_selection import cross_val_score

optimized_rf = RandomForestRegressor(max_depth=10, min_samples_leaf=2,␣
 ↪min_samples_split=5, n_estimators=300)
num_folds = 5

mse_scores = cross_val_score(optimized_rf, X, y,␣
 ↪scoring='neg_mean_squared_error', cv=num_folds)
mse_scores = -mse_scores
print("Mean MSE:", mse_scores.mean())

rmse_scores = cross_val_score(optimized_rf, X, y,␣
 ↪scoring='neg_root_mean_squared_error', cv=num_folds)
rmse_scores = -rmse_scores
print("Mean RMSE:", rmse_scores.mean())

mae_scores = cross_val_score(optimized_rf, X, y,␣
 ↪scoring='neg_mean_absolute_error', cv=num_folds)
mae_scores = -mae_scores
print("Mean MAE:", mae_scores.mean())
```

```
Mean MSE: 1.5944072085619527
Mean RMSE: 1.2627100066704786
Mean MAE: 0.8473926819010259
```

7 Creating a Hybrid Model with collaborative filtering and content-based model

[31]:
```
# Train the models individually
# RandomForest
optimized_rf = RandomForestRegressor(max_depth=10, min_samples_leaf=2,␣
 ↪min_samples_split=5, n_estimators=300)

X = merged_data.drop(columns=['user_id', 'recipe_id', 'rating', 'review',␣
 ↪'date'])
y = merged_data['rating']
#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

optimized_rf.fit(X_train, y_train)

# SVD
optimized_SVD = SVD(n_factors=50, n_epochs=20, lr_all=0.005, reg_all=0.1)

reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(raw_interactions_df[['user_id', 'recipe_id',␣
 ↪'rating']], reader)
trainset = data.build_full_trainset()
optimized_SVD.fit(trainset)
```

[31]:
```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x7fab35d8f280>
```

[54]:
```
# Optimizing model weights
from sklearn.model_selection import train_test_split

feature_columns = [col for col in merged_data2.columns if col.
 ↪startswith('tag_')]
X = merged_data2[feature_columns]
y = merged_data2['rating']

user_ids = merged_data2['user_id']
item_ids = merged_data2['recipe_id']

X_train, X_test, y_train, y_test, user_ids_train, user_ids_test,␣
 ↪item_ids_train, item_ids_test = train_test_split(
    X, y, user_ids, item_ids, test_size=0.2, random_state=42)

reader = Reader(rating_scale=(1, 5))
```

```python
# Create the training dataset for SVD
train_data = pd.DataFrame({
    'user_id': user_ids_train,
    'item_id': item_ids_train,
    'rating': y_train
})
train_data = Dataset.load_from_df(train_data, reader)
trainset = train_data.build_full_trainset()

def weighted_prediction(user_id, item_id, features, weight_rf, weight_svd):
    rf_prediction = optimized_rf.predict([features])[0]
    svd_prediction = optimized_SVD.predict(user_id, item_id).est
    return weight_rf * rf_prediction + weight_svd * svd_prediction

def compute_error(weights, user_ids, item_ids, features, actual_ratings):
    weight_rf, weight_svd = weights
    predictions = [
        weighted_prediction(user_id, item_id, feature, weight_rf, weight_svd)
        for user_id, item_id, feature in zip(user_ids, item_ids, features)
    ]
    return mean_squared_error(actual_ratings, predictions)
```

[ ]:

```python
```

[ ]:

```python
from scipy.optimize import minimize

subset_size = 10000
subset_indices = np.random.choice(X_train.index, subset_size, replace=False)

X_subset = X_train.loc[subset_indices].values
y_subset = y_train.loc[subset_indices].values
user_ids_subset = user_ids_train.loc[subset_indices].values
item_ids_subset = item_ids_train.loc[subset_indices].values

# Initial guesses for weights
initial_weights = [0.5, 0.5]

# The bounds ensure that weights are between 0 and 1
bounds = [(0, 1), (0, 1)]

# Perform the optimization with a tolerance value
result = minimize(
    compute_error,
    initial_weights,
    args=(user_ids_subset, item_ids_subset, X_subset, y_subset),
    bounds=bounds,
```

```
    method='SLSQP',  # Sequential Least Squares Programming
    tol=1e-3  # Adjust the tolerance for faster convergence
)

optimized_weights = result.x
print("Optimized weights:", optimized_weights)
```

[57]:

```
print("Optimized weights:", optimized_weights)
```

```
Optimized weights: [0. 1.]
```

[ ]:

```
# Testing the model with optimized weights

X_test_rf = merged_data2.drop(columns=['user_id', 'recipe_id', 'date',␣
 ↪'rating', 'review']) # Prepare the data for the RandomForest model


test_data_svd = merged_data2[['user_id', 'recipe_id']] # Prepare the data for␣
 ↪the SVD model


y_true = merged_data2['rating'] # Actual ratings for evaluation

weight_rf = 0
weight_svd = 1


hybrid_predictions = []
for index, row in merged_data2.iterrows():
    # Get features for RandomForest
    features_rf = row.drop(['user_id', 'recipe_id', 'date', 'rating',␣
 ↪'review']).values.reshape(1, -1)

    # Make predictions using both models
    rf_pred = optimized_rf.predict(features_rf)[0]
    svd_pred = optimized_SVD.predict(str(row['user_id']),␣
 ↪str(row['recipe_id'])).est

    # Combine predictions using a weighted average
    hybrid_pred = (rf_pred * weight_rf) + (svd_pred * weight_svd)
    hybrid_predictions.append(hybrid_pred)

# Evaluate the hybrid model
from sklearn.metrics import mean_squared_error, mean_absolute_error

mse = mean_squared_error(y_true, hybrid_predictions)
rmse = mean_squared_error(y_true, hybrid_predictions, squared=False)
```

```python
mae = mean_absolute_error(y_true, hybrid_predictions)

print(f'MSE: {mse}, RMSE: {rmse}, MAE: {mae}')
```

[59]:
```python
print(f'MSE: {mse}, RMSE: {rmse}, MAE: {mae}')
```

```
MSE: 1.5995958057022754, RMSE: 1.2647512821508724, MAE: 0.8492393396162298
```

[ ]:
```python

```