

EXPERIMENT NO.: 8

AIM: To implement a recommendation system on your dataset using the following machine learning techniques: Regression, Classification, Clustering, Decision tree, Anomaly detection, Dimensionality Reduction, Ensemble Methods.

Theory:**Types of Recommendation Systems**

A Recommendation System suggests relevant items to users based on their preferences, behavior, or other factors. There are several types of recommendation techniques:

1. Content-Based Filtering

Idea: Recommends items similar to those the user has liked before.

- Works on: Item features (attributes such as brand, price, category).

Example:

- If a user buys a Samsung phone, they might be recommended another Samsung device based on brand preference.
- Uses techniques like TF-IDF (for text data), Cosine Similarity, Decision Trees, etc.

2. Collaborative Filtering (CF)

Idea: Recommends items based on similar users' preferences.

- Works on: User interactions rather than item features.

Example: • If User A and User B have similar purchase histories, items bought by User

A but not yet by User B will be recommended to User B.

- Uses methods like User-Based CF and Item-Based CF.

3. Hybrid Recommendation System

Idea: Combines Content-Based Filtering and Collaborative Filtering for better accuracy.

Example:

- Netflix uses a hybrid approach, considering both user preferences and what similar users watch.

4. Knowledge-Based Recommendation

Idea: Recommends items based on explicit domain knowledge rather than past user behavior.

Example:

- A car recommendation system suggests vehicles based on engine type, price, and fuel efficiency, regardless of past purchases.

Recommendation System Evaluation Measures

Accuracy Measures:

These metrics evaluate how well the recommended items match the actual preferences or ratings of users.

- **Mean Absolute Error (MAE):**

- Measures the average of the absolute differences between predicted ratings and actual ratings.

- **Formula:** $MAE = \frac{1}{n} \sum_{i=1}^n |r_i - \hat{r}_i|$

- r_i = Actual rating

- \hat{r}_i = Predicted rating

- **Lower is better.**

- **Root Mean Squared Error (RMSE):**

- Similar to MAE but gives higher weight to large errors due to squaring the differences.

- **Formula:** $RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (r_i - \hat{r}_i)^2}$

- **Lower is better.**

- **Precision:**

- Measures the fraction of recommended items that are actually relevant to the user.

$$\text{Formula: Precision} = \frac{\text{Number of relevant recommended items}}{\text{Total number of recommended items}}$$

- Higher is better.

- Recall:

- Measures the fraction of relevant items that were actually recommended to the user.

$$\text{Formula: Recall} = \frac{\text{Number of relevant recommended items}}{\text{Total number of relevant items}}$$

- Higher is better.

- F1-Score:

- The harmonic mean of Precision and Recall, balancing both.

$$\text{Formula: } F1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

- Higher is better.

- Hit Rate:

- Measures the fraction of users for whom at least one relevant item is recommended.

$$\text{Formula: HitRate} = \frac{\text{Users with at least one relevant recommendation}}{\text{Total users}}$$

- Higher is better.

- Coverage:

- Measures the proportion of items from the total available set that are recommended to users.

$$\text{Formula: Coverage} = \frac{\text{Number of unique recommended items}}{\text{Total number of items available}}$$

- Higher is better.

Diversity and Novelty Measures:

These metrics focus on the **variety** of recommended items and how **unexpected** they are.

- **Diversity:**

- Measures how different the recommended items are from each other. A diverse set prevents the system from recommending very similar items.
- **Formula:**
 - Calculate the pairwise similarity between recommended items (e.g., cosine similarity) and compute the average diversity across all users.
 - Higher diversity is better.

- **Novelty:**

- Measures how **unexpected** or **unknown** the recommended items are to the user.
- For example, recommending items that the user hasn't interacted with before (e.g., exploring genres they haven't tried).
- Higher novelty is better.

- **Serendipity:**

- Similar to novelty but with a focus on the surprise element that still fits the user's interests.

- The idea is to recommend items that are surprising but still relevant.

Implementation

The Diet recommendation is built using Nearest Neighbors algorithm which is an unsupervised learner for implementing neighbor searches. For our case, we used the brute-force algorithm using cosine similarity due to its fast computation for small datasets.


1. Importing dataset


```
import kagglehub
from kagglehub import KaggleDatasetAdapter

# Set the path to the file you'd like to load
file_path = "recipes.csv"

# Load the latest version
df = kagglehub.load_dataset(
    KaggleDatasetAdapter.PANDAS,
    "irkaal/foodcom-recipes-and-reviews",
    file_path,
)

print("First 5 records:", df.head())
```

 data = df
data.head()



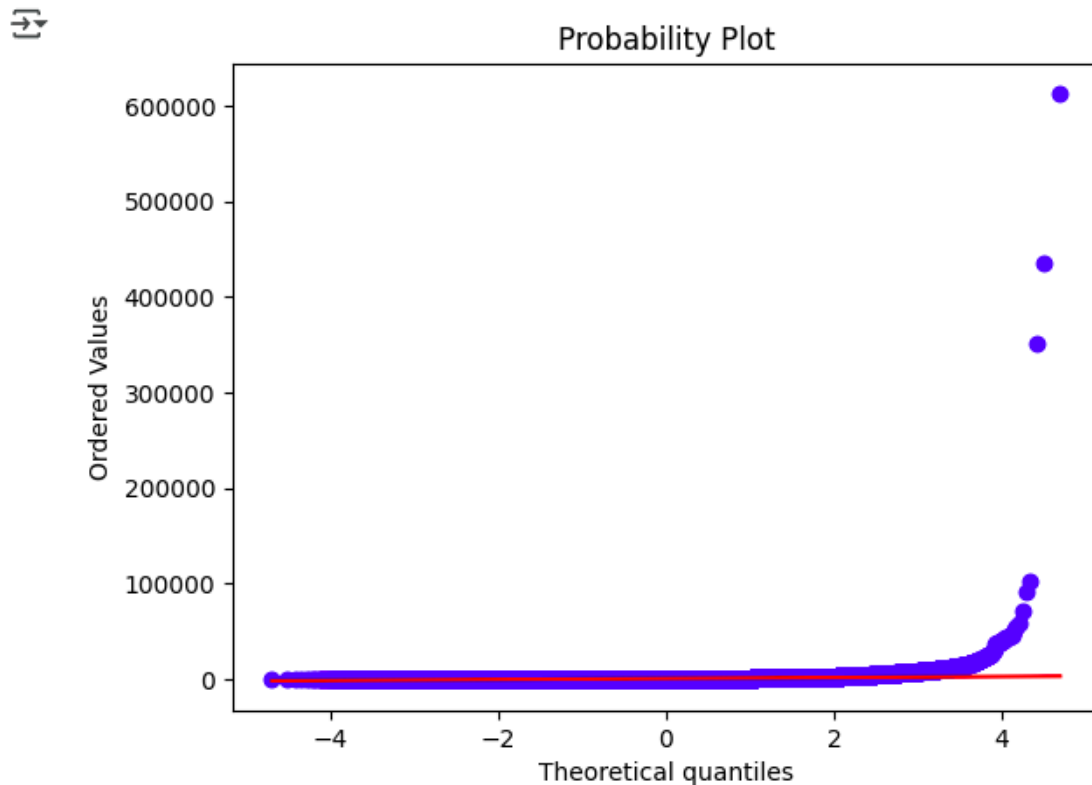
	RecipeId	Name	AuthorId	AuthorName	CookTime	PrepTime	TotalTime	DatePublished	Description
0	38	Low-Fat Berry Blue Frozen Dessert	1533	Dancer	PT24H	PT45M	PT24H45M	1999-08-09T21:46:00Z	Make and share this Low-Fat Berry Blue Frozen ...
1	39	Biryani	1567	elly9812	PT25M	PT4H	PT4H25M	1999-08-29T13:12:00Z	Make and share this Biryani recipe from Food.com.
2	40	Best Lemonade	1566	Stephen Little	PT5M	PT30M	PT35M	1999-09-05T19:52:00Z	This is from one of my first Good House Keepi...
3	41	Carina's Tofu-Vegetable Kebabs	1586	Cyclopz	PT20M	PT24H	PT24H20M	1999-09-03T14:54:00Z	This dish is best prepared a day in advance to...
4	42	Cabbage Soup	1538	Duckie067	PT30M	PT20M	PT50M	1999-09-19T06:19:00Z	Make and share this Cabbage Soup recipe from F...

5 rows × 28 columns

The recipes dataset contains 522,517 recipes from 312 different categories. This dataset provides information about each recipe like cooking times, servings, ingredients, nutrition, instructions, and more.

2. Detecting Outlier

```
import pylab
import scipy.stats as stats
stats.probplot(data.Calories.to_numpy(), dist="norm", plot=pylab)
pylab.show()
```



The data exhibits a significant right-skewness, indicating that the majority of values are relatively small, while a few extreme values with very high calorie counts are pulling the distribution toward the right. Additionally, the presence of outliers, particularly the very large values, seems to be influencing the overall distribution.

3. Setting the maximum nutritional values for each category for healthier recommendations

```

max_Calories=2000
max_daily_fat=100
max_daily_Saturatedfat=13
max_daily_Cholesterol=300
max_daily_Sodium=2300
max_daily_Carbohydrate=325
max_daily_Fiber=40
max_daily_Sugar=40
max_daily_Protein=200
max_list=[max_Calories,max_daily_f

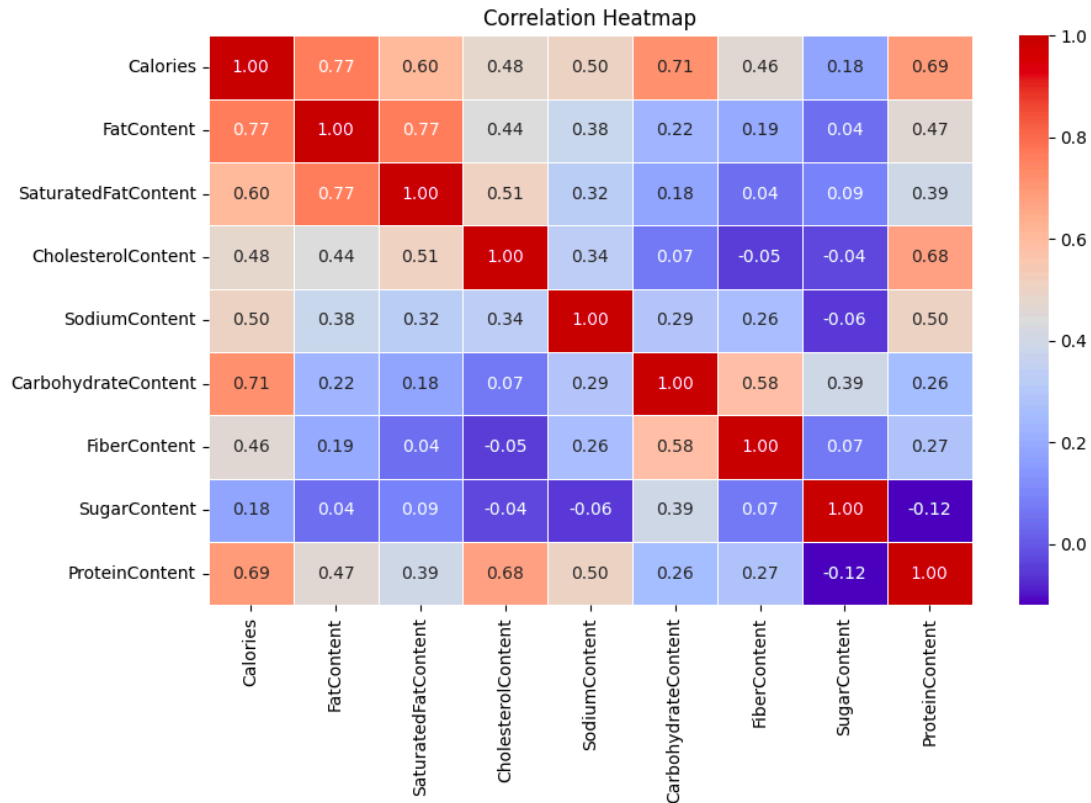
```

4. Correlation among nutritional values

```
extracted_data.iloc[:,6:15].corr()
```

	Calories	FatContent	SaturatedFatContent	CholesterolContent	SodiumContent	CarbohydrateContent	FiberContent	SugarContent	ProteinContent
Calories	1.000000	0.767356	0.603317	0.478934	0.501082	0.711640	0.458711	0.180895	0.689447
FatContent	0.767356	1.000000	0.767357	0.440515	0.381944	0.223549	0.192142	0.042603	0.468088
SaturatedFatContent	0.603317	0.767357	1.000000	0.512186	0.319671	0.176623	0.044003	0.090721	0.388618
CholesterolContent	0.478934	0.440515	0.512186	1.000000	0.335843	0.066104	-0.047346	-0.036112	0.675302
SodiumContent	0.501082	0.381944	0.319671	0.335843	1.000000	0.294636	0.260479	-0.055518	0.500457
CarbohydrateContent	0.711640	0.223549	0.176623	0.066104	0.294636	1.000000	0.580535	0.390120	0.255447
FiberContent	0.458711	0.192142	0.044003	-0.047346	0.260479	0.580535	1.000000	0.068758	0.273488
SugarContent	0.180895	0.042603	0.090721	-0.036112	-0.055518	0.390120	0.068758	1.000000	-0.120441
ProteinContent	0.689447	0.468088	0.388618	0.675302	0.500457	0.255447	0.273488	-0.120441	1.000000

- **Fat and calories** are strongly correlated, suggesting that food with high fat content tends to have higher calorie content.
- **Fiber and carbohydrates** are related, with fiber often being part of the carbohydrate content in food.
- **Cholesterol and protein** content are moderately related, possibly due to the fact that animal-based foods, which are rich in cholesterol, are also rich in protein.
- **Sodium** is moderately correlated with both calories and protein content, suggesting that foods higher in calories or protein often have more sodium (which is typical of processed foods).



5. Normalising data using z-score normalisation

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
prep_data=scaler.fit_transform(extracted_data.iloc[:,6:15].to_numpy())
```

```
array([[ -0.55093359, -0.91281917, -0.77924852, ...,  0.15672078,
         2.35502102, -0.68338127],
       [ 1.47428542,  1.13139595, -0.0647135 , ...,  3.91055068,
         2.56324444,  1.25158691],
       [-0.92414618, -1.11248669, -1.12222533, ...,  0.4855234 ,
         0.98513013, -0.60183088],
       ...,
       [ 0.49162165,  0.73206091,  1.85024037, ..., -0.61048534,
         1.76322815, -0.56476253],
       [ 0.25704672,  0.03797856,  1.02137974, ..., -0.61048534,
         1.54404561, -0.63148557],
       [-1.40937801, -1.09347074, -1.12222533, ..., -0.82968708,
        -0.94367625, -0.74269064]])
```

6. Training the model using K Nearest Neighbours (KNN)

```
from sklearn.neighbors import NearestNeighbors  
neigh = NearestNeighbors(metric='cosine',algorithm='brute')  
neigh.fit(prepare_data)
```

Here, the metric is set to **'cosine'**, meaning the model will measure the cosine similarity between data points. The **'brute'** algorithm is being used, which computes all pairwise distances between data points without optimization for speed.

7. Applying KNN

```
from sklearn.pipeline import Pipeline  
from sklearn.preprocessing import FunctionTransformer  
transformer =  
FunctionTransformer(neigh.kneighbors,kw_args={'return_distance':False})  
pipeline=Pipeline([('std_scaler',scaler),('NN',transformer)])  
params={'n_neighbors':10,'return_distance':False}  
pipeline.get_params()  
pipeline.set_params(NN__kw_args=params)
```

8. Testing the model

```
extracted_data[extracted_data['RecipeIngredientParts'].str.contains("egg",regex=False)]
```

	RecipeId	Name	CookTime	PrepTime	TotalTime	RecipeIngredientParts	Calories	FatContent	SaturatedFatContent
3	41	Carina's Tofu-Vegetable Kebabs	PT20M	PT24H	PT24H20M	c("extra firm tofu", "eggplant", "zucchini", "...	536.1	24.0	3.8
7	45	Buttermilk Pie With Gingersnap Crumb Crust	PT50M	PT30M	PT1H20M	c("sugar", "margarine", "egg", "flour", "salt"...	228.0	7.1	1.7
12	50	Biscotti Di Prato	PT50M	PT20M	PT1H10M	c("flour", "sugar", "baking powder", "salt", "...	89.4	2.6	0.3
18	56	Buttermilk Pie	PT1H	PT20M	PT1H20M	c("butter", "margarine", "sugar", "flour", "eg...	395.9	19.1	9.8
22	60	Blueberry Dessert	NaN	PT35M	PT35M	c("Bisquick baking mix", "sugar", "butter", "m...	381.1	17.3	8.8
...
522484	541351	Spinach & Mushroom Quiche with Boursin	PT1H	PT20M	PT1H20M	c("butter", "onion", "sweet pepper", "carrots"...	197.6	11.0	4.0
522490	541357	Chocolate Rum Snowballs	PT8M	PT15M	PT23M	c("rolled oats", "sweetened flaked coconut", "...	127.8	6.2	4.1
522500	541367	Thick Peanut Pancakes	PT10M	PT45M	PT55M	c("plain flour", "baking powder", "baking soda...	712.9	25.4	8.6
522510	541377	Slow-Cooker Classic Coffee Cake	PT3H	PT20M	PT3H20M	c("all-purpose flour", "brown sugar", "butter"...	358.9	19.8	10.5

In the above example, we instructed the model to recommend recipes that contain "eggs," and it successfully retrieved those recipes.

9. Creating functions for all

def scaling(dataframe):

scaler=StandardScaler()

prep_data=scaler.fit_transform(dataframe.iloc[:,6:15].to_numpy())

return prep_data,scaler

def nn_predictor(prepare_data):

neigh = NearestNeighbors(metric='cosine',algorithm='brute')

neigh.fit(prepare_data)

return neigh

def build_pipeline(neigh,scaler,params):

print("Building pipeline with params (type):", type(params)) # Debug: should print
<class 'dict'>

transformer = FunctionTransformer(neigh.kneighbors,kw_args=params)

pipeline=Pipeline([('std_scaler',scaler),('NN',transformer)])

```
    return pipeline

#Extracts the numeric columns with numeric values less than
def extract_data(dataframe, ingredient_filter, max_nutritional_values,
allergic_filter=None):
    extracted_data = dataframe.copy()

    # Filter based on nutritional limits
    for column, maximum in zip(extracted_data.columns[6:15], max_nutritional_values):
        extracted_data = extracted_data[extracted_data[column] < maximum]

    # Include recipes that contain specified ingredients (if provided)
    if ingredient_filter is not None:
        for ingredient in ingredient_filter:
            extracted_data = extracted_data[
                extracted_data['RecipeIngredientParts'].str.contains(ingredient, regex=False)
            ]

    # Exclude recipes that contain any allergenic ingredients
    if allergic_filter is not None:
        for allergen in allergic_filter:
            extracted_data = extracted_data[
                ~extracted_data['RecipeIngredientParts'].str.contains(allergen, regex=False)
            ]

    return extracted_data

def apply_pipeline(pipeline, _input, extracted_data):
    return extracted_data.iloc[pipeline.transform(_input)[0]]

def recommend(dataframe, _input, max_nutritional_values, ingredient_filter=None,
allergic_filter=None, params={'return_distance': False, 'n_neighbors': 10}):
    extracted_data = extract_data(dataframe, ingredient_filter, max_nutritional_values,
allergic_filter)
    prep_data, scaler = scaling(extracted_data)
    neigh = nn_predictor(prep_data)
    pipeline = build_pipeline(neigh, scaler, params)
    return apply_pipeline(pipeline, _input, extracted_data)
```

10. Testing Recommendation with custom nutritional values

Column Names: ['Calories', 'FatContent', 'SaturatedFatContent', 'CholesterolContent', 'SodiumContent', 'CarbohydrateContent', 'FiberContent', 'SugarContent', 'ProteinContent']

Test Input Values: [[1800, 30, 12, 250, 1500, 300, 30, 20, 100]]

We will input the following test values: [[1800, 30, 12, 250, 1500, 300, 30, 20, 100]] into the model, and it will recommend recipes that have approximate values matching these inputs.

```
[1800, 30, 12, 250, 1500, 300, 30, 20, 100]
```

```
ingredient_filter = ['egg', 'milk']
```

```
allergic_filter = ['flour']
```

```
params = {'return_distance': False, 'n_neighbors': 5}
```

```
test_input = manual_input_df.to_numpy()
```

```
# print(recommendations)
```

```
recommend(dataset, test_input, max_list, ingredient_filter, allergic_filter,
params=params)
```

Building pipeline with params (type): <class 'dict'>

	RecipeId	Name	CookTime	PrepTime	TotalTime	RecipeIngredientParts	Calories	FatContent	SaturatedFatContent	CholesterolContent
192471	201006	Spaghetti With Turkey Meatballs	PT10M	PT50M	PT1H	c("olive oil", "garlic cloves", "crushed tomat...	918.1	28.2	5.9	145.4
5142	8067	Meatball Stew with Dumplings	NaN	PT0S	PT20M	c("sour cream", "green beans", "carrots", "pot...	1331.2	32.0	10.1	164.5
129249	135785	Perfect Pastitsio (Vegetarian)	PT40M	PT35M	PT1H15M	c("olive oil", "onion", "vegetarian ground bee...	575.4	15.8	5.2	100.7
188211	196628	Western Tamale Pie	PT25M	PT15M	PT40M	c("onion", "green bell pepper", "chili powder"...	565.2	21.3	6.6	88.2
226699	236213	Slow-Cooked Meatloaf and Veggies	PT8H	PT15M	PT8H15M	c("egg", "milk", "onion", "green peppers", "sa...	668.2	20.0	7.9	159.2

In this case, we also specified that the recommended recipes should include certain ingredients like eggs and milk, while excluding allergens such as flour. The model then provided recipes that contained the specified ingredients but excluded those with allergens.

Conclusion:

In conclusion, the experiment successfully demonstrated the implementation of a content based recommendation system that leverages data normalization, K-Nearest Neighbors with cosine similarity, and a tailored pipeline to provide personalized recipe recommendations. The system effectively incorporates nutritional constraints, ingredient preferences, and allergen filters, ensuring that the output meets health-focused criteria while aligning with user-specific ingredient requirements. This modular approach not only highlights the strengths of supervised learning in recommendation contexts but also lays the foundation for future improvements through user feedback and more advanced evaluation metrics.