OLDEST DREAM

By

Rason Yudha Pati Nugraha

A Report Submitted
in Partial Fulfillment of the Requirements
for the Artificial Intelligence Class

Cikarang, Bekasi, Indonesia

TABLE OF CONTENTS

THEME

The main theme that will be used on this project is an operational efficiency AI system that can help the medicine discovery process by analyzing vast datasets to identify potential medicine candidates. It's an AI that works as a recommender system where it can predict the list of the closest medicine that has the same similarities with the medicine that is being input.

Recommender systems are information processing systems that actively gather various kinds of data in order to build their recommendations. Data is primarily about the items to be recommended and the users who will receive these recommendations. But, since the data and knowledge sources available for recommender systems can be very diverse, ultimately, whether they can be exploited or not depends on the recommendation.

Combining all the materials that have been taught in AI subjects, this project is mostly inspired by the movie recommendation system where treating the movie datasets as a vector then measuring similarities and differences by calculating the correlations between items.

PROBLEM & OBJECTIVES

1. Problems

A. Managing wide medicines type

With the development of the world of health, where much research has been done on various types of diseases, medicines have also been developed as a tool to help cure or be used as an antidote for a disease. There are many types of medicines with different indications circulating, which are used either to help cure or as an antidote to disease. With the increasing number of medicine brands circulating with the same indications and functions, doctors and pharmacists are chosen to provide the right medicine according to the disease suffered by the patient.

B. Limited medicine availability

At some point, pharmacy only has a limited type of medicine that may not be the most suitable solution for patient needs. Therefore pharmacists are demanded to be able to determine the replacement that serves the similar needs of the actual medicine. This could be a very challenging responsibility for pharmacists since they need to completely understand the purpose of each medicine and minimize the negative outcome as best as they can.

C. Inefficiency of manual search

There are not only small numbers of pharmacies that are still searching for the suitable medicine manually. This is inefficient in some scenarios where if there is a medicine prescribed by a doctor, it turns out that the medicine is not available. So, the pharmacist will open the information on the substitute medicine in the medicine book to get a look for the similarities of the medicine.

2. Objective

A. Increasing the efficiency of medicine searching

As the point that has already been mentioned above, having an efficient medicine search might be crucial for increasing the performance of pharmacists on finding the most suitable medicine for patients. With this project, it is hoped that this project can be developed further and used to reduce the time required to search for the medicines that patients need.

METHOD

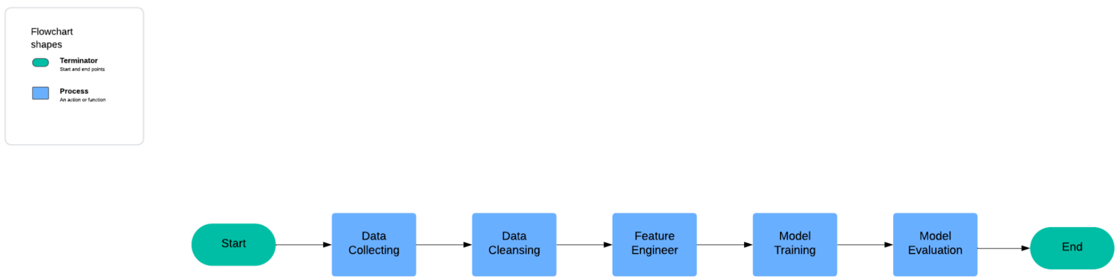
1. Background

In developing a recommender system there are several elements that oftentimes required in creating a considerably good recommendation system and one of them is called items. Items are the product or object that is being recommended throughout the system process. Then there's something called users. Users who will diversify the goals and characteristics by providing information or behavior that may be needed by the recommendation system to predict the list of items to be displayed. And the last element which will support in creating a good recommender system is transactions. Transactions are data that stores important information after the user interactions to determine the list of logs that the user has been accessed before to help the system on determining the better items.

To provide an accurate recommendation, the system must be able to compare each item's utility or characteristics and then decide the item based on the comparison. On performing the predictions of recommended items, there are several techniques that can be done in the system. Every technique that is being used is based on the data that the system will use to do a recommendation prediction. Since this project only takes datasets on performing the calculations therefore the most suitable approach is to use knowledge-based techniques. Knowledge-based techniques is a system recommendation that calculates based on how certain features meet input similarities and preferences.

2. Method

Since the main objective of this project is to develop a recommendation AI system that is able to generate the recommendations of the medicine based on the similarities of the medicines in the dataset there are several steps that need to be done in order to reach the objective. The development is done under the steps of importing the modules and dataset that are needed, performing data preprocessing to create more accurate predictions based on the datasets, do the model training, and then test the model.



A. Importing the required modules

Before getting into the code parts, some modules need to be prepared in order to run every step in this report. Here are the lists of modules that are being used to run this project smoothly.

```
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
import re

from gensim.models import Word2Vec
from tqdm import tqdm
```

Description :

- **pandas -** doing the data manipulations
- **numpy -** doing the mathematical calculations
- **cosine_similarity -** calculate the similarity between 2 vectors
- **re -** tool to matching the string pattern
- **Word2Vec -** the model that is used to interpret words into vectors
- **tqdm -** support module to create the progressing bar

B. Data preparation

Since the main objective is to create a recommendation AI system that could predicts the recommended medicine that has the similar traits as the input medicine, there are two datasets that is being used on this project which are “az medicine dataset of india” and “250k medicines usage side effects and substitutes”. Both datasets can be accessed from kaggle. Although in the dataset it’s explicitly mentioned the source is from India, but after getting into the detail of the dataset, it’s still valid to be used as a reference of the medicine’s compositions.

C. Data preprocessing

Before starting to get into any of the model training, it’s a good practice to check the data quality beforehand. Based on the 2 datasets, although overall it’s still considered as a ‘good to use’ dataset, there may be some parts that need to be modified in order to achieve more accurate results. Therefore, data preprocessing is viable and a good practice before directly starting the model training with the available dataset. There are two parts of the data preprocessing components that are being done which are data cleansing and feature engineering.

a. Data cleansing

The data cleansing step is being done due to the fact that there are some data duplications and irrelevant data in the dataset. Starts by checking for any data duplications that occur in the dataset, we should drop the column ‘id’ beforehand since it serves as the data identifier which consists of unique value among the other, therefore we should drop it first before checking the data duplication.

```
data1 = data1.drop(columns=['id'])
data2 = data2.drop(columns=['id'])
```

```
duplicates1 = data1.duplicated()
num_duplicates1 = duplicates1.sum()
print(f'Number of duplication in data 1: {num_duplicates1}')

duplicates2 = data2.duplicated()
num_duplicates2 = duplicates2.sum()
print(f'Number of duplication in data 2: {num_duplicates2}')
```

```
Number of duplication in data 1: 5
Number of duplication in data 2: 24204
```

```
data1 = data1.drop_duplicates()
data2 = data2.drop_duplicates()
```

The next step is to dive into a more specific column of the dataset that may contain false information or anything else. In both data 1 and data 2, we will check the duplication of the value inside the column 'name' to get the cause of the duplication.

```
data1[data1['name'] == "Ringer Lactate Infusion"]
```

	name	price(₹)	Is_discontinued	manufacturer_name	type	pack_size_label	short_composition1	short_composition2
186117	Ringer Lactate Infusion	22.50	False	Baxter India Pvt Ltd	allopathy	bottle of 500 ml Infusion	Sodium Chloride (0.600gm)	Sodium Lactate (0.320gm)
188582	Ringer Lactate Infusion	46.11	False	Albert David Ltd	allopathy	bottle of 500 ml Infusion	Sodium Chloride (0.600gm)	Sodium Lactate (0.320gm)
189321	Ringer Lactate Infusion	49.72	False	Otsuka Pharmaceutical India Pvt Ltd	allopathy	bottle of 500 ml Infusion	Sodium Chloride (0.6gm)	Sodium Lactate (0.32gm)
189803	Ringer Lactate Infusion	49.73	False	Parenteral Drugs India Ltd	allopathy	bottle of 500 ml Infusion	Sodium Chloride (600mg)	Sodium Lactate (320mg)
192464	Ringer Lactate Infusion	37.50	False	Otsuka Pharmaceutical India Pvt Ltd	allopathy	bottle of 1000 ml Infusion	Sodium Chloride (0.6gm)	Sodium Lactate (0.32gm)
194320	Ringer Lactate Infusion	23.33	False	Parenteral Drugs India Ltd	allopathy	bottle of 1000 ml Infusion	Sodium Chloride (0.6gm)	Sodium Lactate (0.32gm)
195433	Ringer Lactate Infusion	29.81	False	Nirlife Healthcare	allopathy	bottle of 500 ml Infusion	Sodium Chloride (0.600gm)	Sodium Lactate (0.320gm)

```
data2[data2['name'] == "ringer lactate infusion"]
```

	name	substitute0	substitute1	substitute2	substitute3	substitute4	sideEffect0	sideEffect1	sideEffect2	sideEffect3	...	sideEffect41
199320	ringer lactate infusion	Ringer Lactate Infusion	Ringer Lactate IP Poly Infusion	RL Infusion	Ringer Lactate Infusion	NaN	No common side effects seen	NaN	NaN	NaN	...	NaN
201811	ringer lactate infusion	Ringer Lactate Infusion	Ringer Lactate Infusion	Ringer Lactate IP Poly Infusion	RL Infusion	NaN	No common side effects seen	NaN	NaN	NaN	...	NaN
202568	ringer lactate infusion	Ringer Lactate Infusion	RINGER LACTATE INFUSION	RINGER LACTATE INFUSION	NaN	NaN	No common side effects seen	NaN	NaN	NaN	...	NaN
203065	ringer lactate infusion	NaN	NaN	NaN	NaN	NaN	No common side effects seen	NaN	NaN	NaN	...	NaN

From the data 1, we can conclude that the duplicated data happens due to the different value of the manufacturer name which is a valid reason to have a duplicated value on the 'name' column, therefore we don't drop the duplication. From the data2, we found that the data duplications are most likely to have the same value among the others only that some of it is written in upper case and some of it in lower case but the most important thing is that some of it contains a less detailed value in some columns. Therefore what we can do is to create a column of 'null_count' to store the value of the amount of null value on each row then sort it by the least null of the data and do the deletion for the data that has more empty value.

Continuing into a more detailed data cleansing, on the data 1, we do a checking on the 'pack_size_label' since there are some plural issues and could be fixed by just modifying it. Whereas on the data 2 there is a little error in the 'use' column where some of the values serve the same meaning. We just need to modify the value by eliminating the verbs that create the ambiguity.

```
def formatted_sentences(text):
    text = text.lower()

    # Eliminating the verbs that has the simillar meaning
    text = re.sub(r'\b(treatment|prevention)\s*(of|and prevention of)?\b', '', text)

    return text.strip()

data2['use0'] = [formatted_sentences(text) if isinstance(text, str) else text for text in data2['use0']]
data2['use1'] = [formatted_sentences(text) if isinstance(text, str) else text for text in data2['use1']]
data2['use2'] = [formatted_sentences(text) if isinstance(text, str) else text for text in data2['use2']]
data2['use3'] = [formatted_sentences(text) if isinstance(text, str) else text for text in data2['use3']]
data2['use4'] = [formatted_sentences(text) if isinstance(text, str) else text for text in data2['use4']]
```

b. Feature engineering

After achieving better data, we then move onto the feature engineering part. This part of data preprocessing is done in purpose of having an important/relevant column that will be used on the model training. We do feature engineering on both data 1 and data 2.

On the data 1 we add a column of type, primary_composition, all_composition, size_composition, all_size. Each column will have an important impact for the model later on which are :

- **type -** store the value of medicine type
- **primary_composition -** store the value of the main composition
- **all_composition -** store the value of combination of all existing compositions from all composition columns
- **size_composition -** store the amount of composition of the main composition
- **all_size -** store the amount of every composition

```
data1.head()
```

manufacturer_name	type	pack_size_label	short_composition1	short_composition2	primary_composition	all_composition	size_composition	all_size
Glaxo SmithKline Pharmaceuticals Ltd	tablet	strip of 10 tablets	Amoxycillin (500mg)	Clavulanic Acid (125mg)	amoxycillin	amoxycillin clavulanicacid	500mg	500mg 125mg
Alembic Pharmaceuticals Ltd	tablet	strip of 5 tablets	Azithromycin (500mg)	NaN	azithromycin	azithromycin	500mg	500mg
Glenmark Pharmaceuticals Ltd	syrup	bottle of 100 ml Syrup	Ambroxol (30mg/5ml)	Levosalbutamol (1mg/5ml)	ambroxol	ambroxol levosalbutamol	30mg/5ml	30mg/5ml 1mg/5ml
Sanofi India Ltd	tablet	strip of 10 tablets	Fexofenadine (120mg)	NaN	fexofenadine	fexofenadine	120mg	120mg
Sanofi India Ltd	tablet	strip of 15 tablets	Pheniramine (25mg)	NaN	pheniramine	pheniramine	25mg	25mg

On the data 2, we add a column of main_substitution, other_substitution, side_effect, and usage. Each column will have an important impact for the model later on which are :

- **main_substitution -** store the main substitution of the medicine
- **other_substitution -** store the list of all possible substitution of the medicine
- **side_effect -** store the list of side effect based on all the value in the side effect columns
- **usage -** store the list of usage based on all the value in the use columns

```
data2.head()
```

use4	Chemical Class	Habit Forming	Therapeutic Class	Action Class	main_substitution	other_substitution	side_effect	usage
NaN	Carboxylic acid derivative	No	UROLOGY	Uricosuric agent-gout	Cadicitron 1.4gm/5ml Syrup	Alkawin 1.4gm/5ml Syrup, Citodac Syrup, Alkaci...	Tiredness, Nausea, Vomiting, Stomach pain, Dia...	gout kidneystone
NaN	Dihydropyridinecarboxylic acids derivatives	No	CARDIAC	Calcium channel blockers-Dihydropyridines (DHP)	StayHappi Amlodipine 5mg Tablet	Amcard 5 Tablet, Amlip 5 Tablet, Amlong Tablet...	Fatigue, Nausea, Sleepiness, Headache, Abdomin...	hypertension(highbloodpressure) angina(heart-r...
NaN	P-Aminophenol Derivative	No	PAIN ANALGESICS	Analgesic & Antipyretic-PCM	Rakbi 250mg Suspension	Alcocin Suspension, Jetmol Suspension, Mormol ...	Stomach pain, Nausea, Indigestion, Vomiting	painrelief fever
NaN	NaN	No	PAIN ANALGESICS	NaN	Nimsa Plus 100mg/500mg Tablet	Nicet Tablet, Nesu P 100mg/500mg Tablet, Troma...	Increased liver enzymes, Nausea, Vomiting, Dia...	painrelief
NaN	Sesquiterpene lactones	No	ANTI MALARIALS	Antimalarial-Artemisinin and derivatives	Arteross 60mg Injection	Neosunate 60mg Injection, Falsitis AR Injectio...	Dizziness, Headache, Injection site reactions ...	malaria

After adding some relevant columns, we then merge both data into one using the left join. We use left join since data 1 is more complete than data 2 and we do the merging based on the column 'name'. Turns out there are over 11% of the data 1 that has no combination partner from the data 2. Since it's less than 50%, it can be considered as safe to use therefore we don't do any kind of data modifying.

```
# Checks rows of data 1 that has no partner data
unmatched_rows = df[df['_merge'] == 'left_only']
percentage_unmatched = (len(unmatched_rows) / len(df)) * 100
print(f'Percentage of unmacthed rows: {percentage_unmatched:.2f}%')
print(f'Total rows of data 1 that has no partner with data 2: {len(unmatched_rows)}')

Percentage of unmacthed rows: 11.06%
Total rows of data 1 that has no partner with data 2: 28092
```

The last step that we do before going into the model training is to drop any empty data and specify which column that will be processed for the model training.

```
# Rechecking missing value
missing_values = df_final.isnull().sum()
percentage_missing = (missing_values / len(df_final)) * 100

missing_data = pd.DataFrame({'Missing Values': missing_values, 'Percentage': percentage_missing})
print(missing_data)
```

```
                     Missing Values  Percentage
name                              0    0.000000
type                              0    0.000000
primary_composition               0    0.000000
all_composition                   0    0.000000
size_composition                  0    0.000000
all_size                          0    0.000000
usage                         27456   11.158118
```

```
# Drop the missing values
df_final = df_final.dropna(subset=['usage'])
```

```
# Specifies which columns will be processed for the model training.
text_columns = ['type','primary_composition', 'all_composition','size_composition','all_size','usage']
```

D. Model training

The model training is starting by turning each word value in the dataset into a vector by using the Word2Vec model. After converting a text word into a matrix, we then train the model with each data value from each column and then store the vector result into a new column that has '_embeddings' format.

```
# Function to train the model on the sentences input by finding patterns in data
def generate_word_embeddings(column):
    unique_values = df_fin[column].unique()
    sentences = [[value] for value in unique_values]
    model = Word2Vec(sentences, min_count=1, vector_size=100)
    return model
```

```
# A placeholder to hold the models
word_embedding_models = {}

for column in text_columns:
    model = generate_word_embeddings(column)
    word_embedding_models[column] = model
```

```
# Applying the trained Word2Vec models to the original columns t Loading... ew columns that contain the embeddings
for column, model in tqdm(word_embedding_models.items(), desc="embeddings"):
    df_fin[column + '_embeddings'] = df_fin[column].apply(lambda x: np.round(model.wv[x].reshape(1, -1), 4) if x in model.wv else [])
```

```
embeddings: 100%|██████████| 6/6 [00:01<00:00,  3.92it/s]
```

After that we do a hyper parameter tuning to define the weight of each column value which will be used on the weighted similarity calculation. We can do the calculation of text similarity by using the cosine similarity module that we've already imported before. It will calculate the similarities between 2 text (as a vector).

```python
# Determining the weights of the word from columns

weightage = {
    'type_embeddings': 0.1,                # Weight for the 'type' feature
    'primary_composition_embeddings': 0.1,       # Weight for the 'primary component' feature
    'all_composition_embeddings': 0.3,         # Weight for the 'entire component' feature
    'size_composition_embeddings': 0.1,              # Weight for the 'value' feature
    'all_size_embeddings': 0.1,        # Weight for the 'entire value' feature
    'usage_embeddings': 0.3                    # Weight for the 'use' feature
}
```

```python
# Calculate the similarities between 2 columns value
def calculate_text_similarity(value1, value2):
    similarity = cosine_similarity(value1, value2)
    return similarity[0][0]

# Calculate the weighted similarity between 2 medicines
def calculate_weighted_similarity(medicine1, medicine2):
    similarity_scores = []
    for column, weight in weightage.items():
        similarity = calculate_text_similarity(medicine1[column].values[0], medicine2[column])
        similarity_scores.append(similarity * weight)
    weighted_similarity = sum(similarity_scores)
    return weighted_similarity
```

3. Result

After doing all the steps, we can do the model testing and do the evaluation based on what the AI system will produce. By far, we use 3 medicines and evaluate what are the medicines that the system recommends to us.

```python
similar_medicines = get_similar_medicines("newtel-h 40 tablet")
print('Top 10 Similar Medicine')
print(similar_medicines)
```

```
Top 10 Similar Medicine
                                name  similarity_score
207941  supratel h 40mg/12.5mg tablet              1.0
20393   angiotel h 40mg/12.5mg tablet              1.0
221612     telsed h 40mg/12.5mg tablet             1.0
238817                 viritel-h tablet            1.0
250320    zunatel h 40mg/12.5mg tablet             1.0
29221      bigtel h 40mg/12.5mg tablet             1.0
161854                  ozotel-h tablet            1.0
220361     telpin h 40mg/12.5mg tablet             1.0
221613      telvo h 40mg/12.5mg tablet             1.0
219938   temmy h 40 mg/12.5 mg tablet              1.0
```

```
[41] similar_medicines = get_similar_medicines("laurunam 1000mg injection")
     print('Top 10 Similar Medicine')
     print(similar_medicines)

Top 10 Similar Medicine
                              name  similarity_score
148290     meropect 1000 injection               0.8
148695   merosler 1000mg injection               0.8
142276   my penum 1000mg injection               0.8
149475    mepwell 1000mg injection               0.8
10342     aspenem 1000mg injection               0.8
182804     peronum 1000mg injection              0.8
141544          meroclay injection               0.8
28829      biopect 1000mg injection              0.8
149625   merogoog 1000mg injection               0.8
146275  meroclass 1000mg injection               0.8
```

```
similar_medicines = get_similar_medicines("prestoheal ds tablet")
print('Top 10 Similar Medicine')
print(similar_medicines)

Top 10 Similar Medicine
                                    name  similarity_score
140360        maczo 90mg/100mg/48mg tablet          0.708741
74598      edeoflam 90mg/100mg/48mg tablet          0.708741
64138    dru 100mg/325mg/50000iu tablet xl          0.474300
101038                    gettifen-p tablet          0.474300
27164                      bromenz-d tablet          0.469255
41836              chymotech br plus tablet          0.469255
71585           defidrot p 80mg/325mg tablet          0.468724
173839               proxidom 500mg tablet          0.466324
107941            ibumax 50 mg/500 mg tablet          0.463604
28692              bemol p 50mg/500mg tablet          0.463604
```

Based on the testing result, we use prestoheal ds tablet, laurunam 1000mg injection, and newtel-h 40 tablet. We got each of them to display the 10 recommended medicines. This means that out of 10 test runs all of them produce accurate 10 results of the closest similarities of the medicine.

EXPECTED RESULTS

By developing this project, hopefully it can perfectly search for the most suitable medicine to help solve the patient's needs by looking at the similarities between each medicine. Not to mention that this project will also reduce the amount of time that it may take previously by pharmacist or any medicine specialist to search the similarities of medicines. If this project is being developed further, by creating a platform that helps to display the data for the pharmacist, this model will help in improving the search performance that will be done throughout the platform.

REFERENCE

Alkaff, M., Khatimi, H., & Eriadi, A. (2020). Sistem Rekomendasi Buku Menggunakan Weighted Tree Similarity dan Content Based Filtering. MATRIK J. Manajemen, Tek. Inform. dan Rekayasa Komput, 20(1), 193-202.

Aryanto, A. D., Santoso, J., & Purwanto, D. D. (2021). Sistem rekomendasi obat pengganti menggunakan metode cnn. Jurnal Sistem Cerdas dan Rekayasa (JSCR), E-ISSN: 2656-7504, 3(1), 25-36

F. Ricci, L. Rokach, and B. Shapira, Recommender System Handbook, Second Edi. New York, 2015.

Y. Koren, R. Bell, and C. Volinsky, “Matrix Factorization Techniques for Recommender Systems,” Computer, vol. 42, no. 8, Ieee, pp. 30–37, Aug. 2009.

LOG HOURS

Please mention the work done by the team members and the hour spent.

No	Student Name	Task Done and Log Hour Spent
1.	Rason Yudha Pati Nugraha	- Researching and comprehend any existing research to support the problem solving (3 hours) - Analyzing the suitable approach to create the AI model (3 hours) - Data Collections (30 mins) - Data Preprocessing (2 hours) - Model Training (1.5 hours) - Model Test (15 minutes) - Report writing (3 hours)
Total		13.25 hours