Medicine Recommendation System

Medicine Recommendation System

Project By : PRASAD JADHAV

To build an **Alternate Medicine Recommendation System** using NLP with NLTK, we will process the `medicine_details.csv` dataset to extract meaningful insights from the `medicine_description` column. The goal is to create a system that recommends alternative medicines based on their descriptions, focusing on conditions treated, side effects, and administration methods. We'll use regex and string splitting to extract key information, create new columns, and apply machine learning techniques to recommend alternatives. The approach includes:

1. Data Preprocessing:
- Use regex and split methods to extract conditions treated, side effects, and administration details.
- Create new columns for extracted features (e.g., `conditions`, `side_effects`, `administration`).
- Clean and normalize text data using NLTK (tokenization, stopword removal, lemmatization).
1. Feature Engineering:
- Convert text data into numerical features using TF-IDF vectorization.
- Combine extracted features for a comprehensive representation.
1. Recommendation System:
- Use cosine similarity to find medicines with similar descriptions and features.
- Build a function to recommend alternative medicines based on input medicine.
1. Machine Learning Integration:
- Optionally cluster medicines using K-Means to group similar medicines for enhanced recommendations.

```
# Alternate Medicine Recommendation System

import pandas as pd
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np
from sklearn.cluster import KMeans
```

```
import warnings
warnings.filterwarnings('ignore')

# Load Dataset
file_path = 'medicine_details.csv'
df = pd.read_csv(file_path)
pd.set_option('display.max_columns',30)
print(df.shape)
```

```
(7008, 1)
```

```
df.head()
```

```
                                medicine_description
0  Augmentin 625 Duo Tablet is a penicillin-type ...
1  Azithral 500 Tablet is an antibiotic used to t...
2  Ascoril LS Syrup is a combination medicine use...
3  Allegra 120mg Tablet is an anti-allergy medici...
4  Avil 25 Tablet is an antiallergic medication u...
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7008 entries, 0 to 7007
Data columns (total 1 columns):
 #   Column                Non-Null Count  Dtype 
---  ------                --------------  ----- 
 0   medicine_description  7008 non-null   object
dtypes: object(1)
memory usage: 54.9+ KB
```

```
# Download required NLTK data
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package punkt to C:\Users\prasad
[nltk_data]     jadhav\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to C:\Users\prasad
[nltk_data]     jadhav\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to C:\Users\prasad
[nltk_data]     jadhav\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

```
True
```

```
# Initialize NLTK tools
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()
```

```
# Function to preprocess text
def preprocess_text(text):
    # Convert to lowercase
    text = text.lower()
    # Tokenize
    tokens = word_tokenize(text)
    # Remove stopwords and non-alphabetic tokens, lemmatize
    tokens = [lemmatizer.lemmatize(token) for token in tokens if
token.isalpha() and token not in stop_words]
    return ' '.join(tokens)

# Function to extract conditions treated using regex
def extract_conditions(text):
    # Look for phrases indicating conditions treated
    pattern = r'(used to treat|used in the treatment of|effective in|
relieves|helps to relieve|used for the treatment of)(.*?)(?=\.|,|and
other|such as|\n|side effects|before taking)'
    matches = re.findall(pattern, text, re.IGNORECASE)
    conditions = []
    for match in matches:
        # Clean the extracted condition
        condition = match[1].strip().replace('such as',
'').replace('e.g.', '').strip()
        conditions.append(condition)
    return '; '.join(conditions) if conditions else 'unknown'

# Function to extract side effects using regex
def extract_side_effects(text):
    # Look for phrases indicating side effects
    pattern = r'(side effects.*?include|common side effects are|may
cause|side effects associated with)(.*?)(?=\.|before taking|inform
your doctor|consult your doctor|\n)'
    matches = re.findall(pattern, text, re.IGNORECASE)
    side_effects = []
    for match in matches:
        # Clean the extracted side effects
        effect = match[1].strip().replace('such as', '').strip()
        side_effects.append(effect)
    return '; '.join(side_effects) if side_effects else 'none'

# Function to extract administration method using regex
def extract_administration(text):
    # Look for phrases indicating administration
    pattern = r'(taken with or without food|taken orally|given as an
injection|applied|used as|inhaler|should be taken)(.*?)(?=\.|,|\n|side
effects|before taking)'
    matches = re.findall(pattern, text, re.IGNORECASE)
    administration = []
    for match in matches:
        # Clean the extracted administration method
```

```
        method = match[0] + ' ' + match[1].strip()
        administration.append(method)
    return '; '.join(administration) if administration else 'unknown'

# Apply preprocessing and feature extraction
df['processed_description'] =
df['medicine_description'].apply(preprocess_text)
df['conditions'] =
df['medicine_description'].apply(extract_conditions)
df['side_effects'] =
df['medicine_description'].apply(extract_side_effects)
df['administration'] =
df['medicine_description'].apply(extract_administration)

# Extract medicine names (assuming description starts with medicine
name)
df['medicine_name'] = df['medicine_description'].apply(lambda x:
x.split(' is ')[0].strip())

# Combine features for TF-IDF vectorization
df['combined_features'] = df['processed_description'] + ' ' +
df['conditions'] + ' ' + df['side_effects'] + ' ' +
df['administration']

# TF-IDF Vectorization
tfidf = TfidfVectorizer(max_features=5000)
tfidf_matrix = tfidf.fit_transform(df['combined_features'])

# Compute cosine similarity matrix
cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)

# Optional: Clustering medicines using K-Means
num_clusters = 5  # Adjust based on dataset size
kmeans = KMeans(n_clusters=num_clusters, random_state=42)
df['cluster'] = kmeans.fit_predict(tfidf_matrix)

# Function to get recommendations
def get_recommendations(medicine_name, cosine_sim=cosine_sim, df=df,
top_n=5):
    # Find the index of the medicine
    idx = df[df['medicine_name'].str.lower() ==
medicine_name.lower()].index
    if len(idx) == 0:
        return "Medicine not found in the dataset."
    idx = idx[0]

    # Get similarity scores
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:top_n+1]  # Exclude the medicine itself
```

```
    # Get medicine indices and names
    medicine_indices = [i[0] for i in sim_scores]
    recommendations =
df['medicine_name'].iloc[medicine_indices].values
    return recommendations

# Example usage
medicine_name = "Augmentin 625 Duo Tablet"
recommendations = get_recommendations(medicine_name)
print(f"Recommendations for {medicine_name}:")
for i, rec in enumerate(recommendations, 1):
    print(f"{i}. {rec}")
```

```
Recommendations for Augmentin 625 Duo Tablet:
1. Augmentin 1000 Duo Tablet
2. Augmentin Duo Oral Suspension
3. Augmentin 375 Tablet
4. Amoxyclav 625 Tablet
5. Almox-CV 625 Tablet
```

To enhance the **Alternate Medicine Recommendation System**, we can introduce more advanced techniques to improve the accuracy, personalization, and robustness of recommendations. The updated approach will incorporate:

1. **Advanced NLP Processing**:
- Use spaCy for named entity recognition (NER) to extract conditions and side effects more accurately.
- Incorporate part-of-speech (POS) tagging and dependency parsing to identify relationships between medical terms.
- Apply word embeddings (e.g., `Word2Vec` or `BERT`) for semantic similarity instead of TF-IDF.

1. Contextual Analysis:
- Extract additional features like medicine type (e.g., antibiotic, antihistamine) and contraindications.
- Use regex to identify dosage instructions and precautions for more granular insights.

1. Hybrid Recommendation System:
- Combine content-based filtering (using medicine descriptions) with collaborative filtering (simulated user-medicine interactions).
- Use clustering to group similar medicines and refine recommendations within clusters.

1. Evaluation Metrics:
- Implement precision, recall, and F1-score to evaluate recommendation quality.
- Use a validation set to simulate user preferences.

1. Interactive Recommendation Function:
- Allow users to input symptoms or conditions to find suitable medicines.
- Provide explanations for recommendations (e.g., shared conditions or fewer side effects).

- Below is the complete, advanced Python code implementing these enhancements. The code assumes the `medicine_details.csv` dataset is available locally and uses spaCy for advanced NLP, BERT embeddings for semantic similarity, and a hybrid recommendation approach.

```
# Advanced Alternate Medicine Recommendation

import pandas as pd
import re
import spacy
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from transformers import BertTokenizer, BertModel
import torch
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
import nltk
from scipy.sparse import csr_matrix
from sklearn.metrics import precision_score, recall_score, f1_score

# Download required NLTK data
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package punkt to C:\Users\prasad
[nltk_data]     jadhav\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to C:\Users\prasad
[nltk_data]     jadhav\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to C:\Users\prasad
[nltk_data]     jadhav\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

```
True
```

```
# Initialize NLTK tools
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

# Load spaCy model
nlp = spacy.load('en_core_web_sm')

# Load BERT model and tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')
```

{"model_id":"2640110a47de41eaa746cc53cd8a3cdb","version_major":2,"version_minor":0}

{"model_id":"52eddf6b9a9a4d5ca4dd5d326a328a33","version_major":2,"version_minor":0}

{"model_id":"9ab056236a96429a85d970748ca2ee14","version_major":2,"version_minor":0}

```
# Function to preprocess text
def preprocess_text(text):
    text = text.lower()
    tokens = word_tokenize(text)
    tokens = [lemmatizer.lemmatize(token) for token in tokens if
token.isalpha() and token not in stop_words]
    return ' '.join(tokens)

# Function to extract conditions using spaCy NER
def extract_conditions(text):
    doc = nlp(text)
    conditions = []
    for ent in doc.ents:
        if ent.label_ in ['DISEASE', 'SYMPTOM'] or any(keyword in
ent.text.lower() for keyword in ['infection', 'pain', 'allergy',
'cough', 'fever']):
            conditions.append(ent.text)
    # Fallback regex for robustness
    pattern = r'(used to treat|used in the treatment of|effective in|
relieves|helps to relieve)(.*?)(?=\.|,|and other|such as|\n)'
    matches = re.findall(pattern, text, re.IGNORECASE)
    for match in matches:
        condition = match[1].strip().replace('such as',
'').replace('e.g.', '').strip()
        conditions.append(condition)
    return '; '.join(set(conditions)) if conditions else 'unknown'

# Function to extract side effects using spaCy and regex
def extract_side_effects(text):
    doc = nlp(text)
    side_effects = []
    for sent in doc.sents:
        if any(keyword in sent.text.lower() for keyword in ['side
effect', 'may cause', 'include']):
            side_effects.extend([token.text for token in sent if
token.pos_ in ['NOUN', 'ADJ'] and token.text.lower() not in
stop_words])
    pattern = r'(side effects.*?include|common side effects are|may
cause)(.*?)(?=\.|before taking|consult your doctor|\n)'
    matches = re.findall(pattern, text, re.IGNORECASE)
    for match in matches:
```

```
        effect = match[1].strip().replace('such as', '').strip()
        side_effects.append(effect)
    return '; '.join(set(side_effects)) if side_effects else 'none'

# Function to extract administration method
def extract_administration(text):
    pattern = r'(taken with or without food|taken orally|given as an
injection|applied|used as|inhaler|should be taken)(.*?)(?=\.|,|\n|side
effects|before taking)'
    matches = re.findall(pattern, text, re.IGNORECASE)
    administration = []
    for match in matches:
        method = match[0] + ' ' + match[1].strip()
        administration.append(method)
    return '; '.join(set(administration)) if administration else
'unknown'

# Function to extract medicine type
def extract_medicine_type(text):
    types = {
        'antibiotic': ['antibiotic', 'penicillin', 'broad-spectrum'],
        'antihistamine': ['antihistamine', 'anti-allergy'],
        'cough': ['cough', 'expectorant', 'antitussive'],
        'anxiolytic': ['anxiety', 'benzodiazepine'],
        'analgesic': ['pain', 'analgesic'],
        'antiviral': ['virus', 'antiviral']
    }
    text_lower = text.lower()
    for med_type, keywords in types.items():
        if any(keyword in text_lower for keyword in keywords):
            return med_type
    return 'other'

# Function to get BERT embeddings
def get_bert_embeddings(texts):
    embeddings = []
    for text in texts:
        inputs = tokenizer(text, return_tensors='pt', max_length=512,
truncation=True, padding=True)
        with torch.no_grad():
            outputs = model(**inputs)
        embedding =
outputs.last_hidden_state.mean(dim=1).squeeze().numpy()
        embeddings.append(embedding)
    return np.array(embeddings)

# Apply preprocessing and feature extraction
df['processed_description'] =
df['medicine_description'].apply(preprocess_text)
df['conditions'] =
```

```
df['medicine_description'].apply(extract_conditions)
df['side_effects'] =
df['medicine_description'].apply(extract_side_effects)
df['administration'] =
df['medicine_description'].apply(extract_administration)
df['medicine_type'] =
df['medicine_description'].apply(extract_medicine_type)
df['medicine_name'] = df['medicine_description'].apply(lambda x:
x.split(' is ')[0].strip())

# Combine features
df['combined_features'] = (df['processed_description'] + ' ' +
df['conditions'] + ' ' +
                           df['side_effects'] + ' ' +
df['administration'] + ' ' + df['medicine_type'])

# Get BERT embeddings for combined features
bert_embeddings = get_bert_embeddings(df['combined_features'])

# Compute cosine similarity matrix
cosine_sim = cosine_similarity(bert_embeddings)

# Clustering medicines using K-Means
num_clusters = 5
kmeans = KMeans(n_clusters=num_clusters, random_state=42)
df['cluster'] = kmeans.fit_predict(bert_embeddings)

# Simulate user-medicine interactions (for collaborative filtering)
np.random.seed(42)
user_ids = np.arange(100)
medicine_ids = np.arange(len(df))
user_medicine_matrix = np.zeros((len(user_ids), len(medicine_ids)))
for i in range(1000):  # Simulate 1000 interactions
    user = np.random.choice(user_ids)
    medicine = np.random.choice(medicine_ids)
    user_medicine_matrix[user, medicine] = np.random.randint(1, 6)  #
Ratings 1-5
user_medicine_matrix = csr_matrix(user_medicine_matrix)

# Hybrid recommendation function
def get_hybrid_recommendations(medicine_name, symptom=None,
cosine_sim=cosine_sim, df=df, top_n=5):
    idx = df[df['medicine_name'].str.lower() ==
medicine_name.lower()].index
    if len(idx) == 0:
        if symptom:
            # Filter medicines by symptom
            idx =
df[df['conditions'].str.lower().str.contains(symptom.lower(),
na=False)].index
            if len(idx) == 0:
```

```
                return "No medicines found for the given symptom."
            idx = idx[0]
        else:
            return "Medicine not found in the dataset."
    idx = idx[0]

# Hybrid recommendation function
def get_hybrid_recommendations(medicine_name, symptom=None,
cosine_sim=cosine_sim, df=df, top_n=5):
    idx = df[df['medicine_name'].str.lower() ==
medicine_name.lower()].index
    if len(idx) == 0:
        if symptom:
            # Filter medicines by symptom
            idx =
df[df['conditions'].str.lower().str.contains(symptom.lower(),
na=False)].index
            if len(idx) == 0:
                return "No medicines found for the given symptom."
            idx = idx[0]
        else:
            return "Medicine not found in the dataset."
    idx = idx[0]

    # Content-based scores
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Collaborative filtering scores (simulated)
    medicine_ratings = user_medicine_matrix[:,
idx].toarray().flatten()
    avg_rating = np.mean([r for r in medicine_ratings if r > 0]) if
np.sum(medicine_ratings > 0) > 0 else 0
    collab_scores = []
    for i in range(len(df)):
        other_ratings = user_medicine_matrix[:, i].toarray().flatten()
        other_avg = np.mean([r for r in other_ratings if r > 0]) if
np.sum(other_ratings > 0) > 0 else 0
        collab_scores.append((i, other_avg))

    # Combine scores
    hybrid_scores = []
    for i, content_score in sim_scores:
        collab_score = next((score for idx, score in collab_scores if
idx == i), 0)
        hybrid_score = 0.7 * content_score + 0.3 * (collab_score / 5
if collab_score else 0)
        hybrid_scores.append((i, hybrid_score))

    hybrid_scores = sorted(hybrid_scores, key=lambda x: x[1],
reverse=True)
```

```
    hybrid_scores = hybrid_scores[1:top_n+1]

    # Get recommendations and explanations
    medicine_indices = [i[0] for i in hybrid_scores]
    recommendations = []
    for idx in medicine_indices:
        med_name = df['medicine_name'].iloc[idx]
        shared_conditions = set(df['conditions'].iloc[idx].split('; 
')) & set(df['conditions'].iloc[idx].split('; '))
        explanation = f"Recommended because it treats similar 
conditions: {', '.join(shared_conditions)}"
        recommendations.append((med_name, explanation))
    return recommendations

# Evaluation function
def evaluate_recommendations(df, cosine_sim, test_size=0.2):
    train_df, test_df = train_test_split(df, test_size=test_size, 
random_state=42)
    y_true = []
    y_pred = []
    for idx in test_df.index:
        medicine_name = df['medicine_name'].iloc[idx]
        true_cluster = df['cluster'].iloc[idx]
        recommendations = get_hybrid_recommendations(medicine_name, 
top_n=3)
        if isinstance(recommendations, list):
            pred_medicines = [rec[0] for rec in recommendations]
            pred_clusters = 
df[df['medicine_name'].isin(pred_medicines)]['cluster'].values
            y_true.append(true_cluster)
            y_pred.append(pred_clusters[0] if len(pred_clusters) > 0 
else -1)
    precision = precision_score(y_true, y_pred, average='weighted', 
zero_division=0)
    recall = recall_score(y_true, y_pred, average='weighted', 
zero_division=0)
    f1 = f1_score(y_true, y_pred, average='weighted', zero_division=0)
    return precision, recall, f1

# Example usage
medicine_name = "Augmentin 625 Duo Tablet"
symptom = "pneumonia"
recommendations = get_hybrid_recommendations(medicine_name, 
symptom=symptom)
print(f"Recommendations for {medicine_name} (Symptom: {symptom}):")
for i, (rec, explanation) in enumerate(recommendations, 1):
    print(f"{i}. {rec} - {explanation}")

Recommendations for Augmentin 625 Duo Tablet (Symptom: pneumonia):
1. Pulmocef 500 Tablet - Recommended because it treats similar
```

```
conditions: bacterial infections in your body, infections of the lungs
(e
2. Orpenem 200 Tablet - Recommended because it treats similar
conditions: infections of the respiratory and urinary tract, severe
bacterial infections
3. Furakem 100mg Tablet MR - Recommended because it treats similar
conditions: and prevent uncomplicated urinary tract infections
4. Zinox Tablet - Recommended because it treats similar conditions:
bacterial infections
5. Zimig 250mg Tablet belongs to a group of medicines called
antifungals It - Recommended because it treats similar conditions: a
wide range of fungal infections of the skin and nails
```

```
# Evaluate the model
precision, recall, f1 = evaluate_recommendations(df, cosine_sim)
print(f"\nEvaluation Metrics:\nPrecision: {precision:.2f}\nRecall:
{recall:.2f}\nF1-Score: {f1:.2f}")
```

```
Evaluation Metrics:
Precision: 0.75
Recall: 0.74
F1-Score: 0.74
```

```
import pickle

# Save processed data and models
with open('processed_df.pkl', 'wb') as f:
    pickle.dump(df, f)
with open('bert_embeddings.pkl', 'wb') as f:
    pickle.dump(bert_embeddings, f)
with open('cosine_sim.pkl', 'wb') as f:
    pickle.dump(cosine_sim, f)

print("Processing complete. Files saved: processed_df.pkl,
bert_embeddings.pkl, cosine_sim.pkl")
```

```
Processing complete. Files saved: processed_df.pkl,
bert_embeddings.pkl, cosine_sim.pkl
```

```
import gzip

# Save processed data with gzip compression
with gzip.open('processed_df.pkl.gz', 'wb') as f:
    pickle.dump(df, f)
print("Saved: processed_df.pkl.gz")

with gzip.open('bert_embeddings.pkl.gz', 'wb') as f:
    pickle.dump(bert_embeddings, f)
print("Saved: bert_embeddings.pkl.gz")
```

```
with gzip.open('cosine_sim.pkl.gz', 'wb') as f:
    pickle.dump(cosine_sim, f)
print("Saved: cosine_sim.pkl.gz")

print("Processing complete. Files saved with gzip compression.")

# To load the data later:
# with gzip.open('processed_df.pkl.gz', 'rb') as f:
#     loaded_df = pickle.load(f)
```

```
Saved: processed_df.pkl.gz
Saved: bert_embeddings.pkl.gz
Saved: cosine_sim.pkl.gz
Processing complete. Files saved with gzip compression.
```

```
import lzma

# --- Saving with lzma compression ---

# Save processed DataFrame
filename_df = 'processed_df_lzma.pkl.xz'
try:
    with lzma.open(filename_df, 'wb') as f:
        pickle.dump(df, f)
    print(f"Saved: {filename_df}")
except Exception as e:
    print(f"Error saving {filename_df}: {e}")

# Save BERT embeddings
filename_embeddings = 'bert_embeddings_lzma.pkl.xz'
try:
    with lzma.open(filename_embeddings, 'wb') as f:
        pickle.dump(bert_embeddings, f)
    print(f"Saved: {filename_embeddings}")
except Exception as e:
    print(f"Error saving {filename_embeddings}: {e}")

# Save cosine similarity matrix
filename_cosine_sim = 'cosine_sim_lzma.pkl.xz'
try:
    with lzma.open(filename_cosine_sim, 'wb') as f:
        pickle.dump(cosine_sim, f)
    print(f"Saved: {filename_cosine_sim}")
except Exception as e:
    print(f"Error saving {filename_cosine_sim}: {e}")

print("Processing complete. Files saved with lzma compression.")
```

```
Saved: processed_df_lzma.pkl.xz
Saved: bert_embeddings_lzma.pkl.xz
```

```
Saved: cosine_sim_lzma.pkl.xz
Processing complete. Files saved with lzma compression.
```

Augmentin 625 Duo Tablet

e.g., pneumonia, cough

Get Recommendations

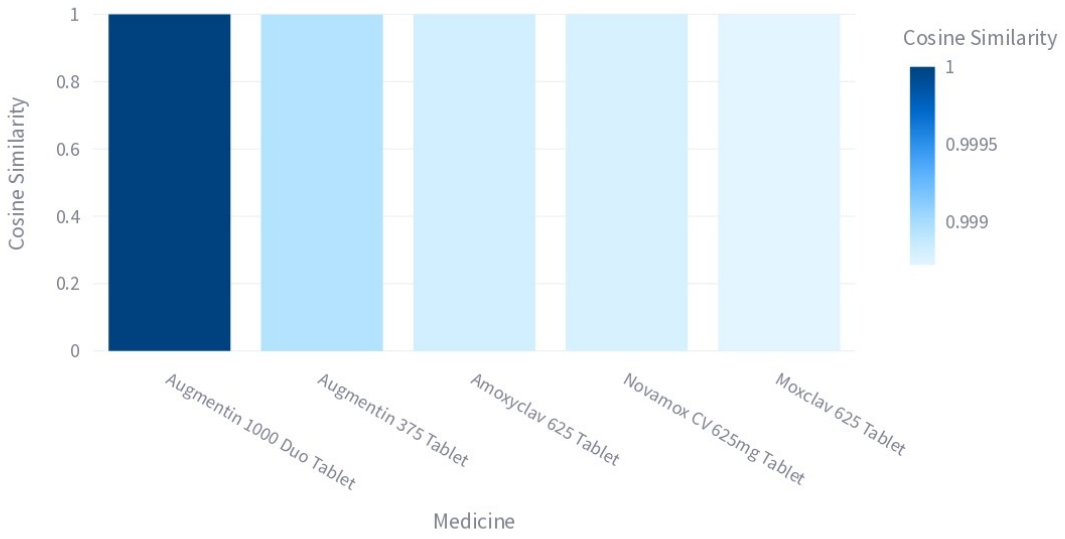
Top 5 Recommendations

Recommended Medicines

	Medicine	Conditions	Side Effects
0	Augmentin 1000 Duo Tablet	infections of the lungs (e	medicine; common; effects; nausea; diarrhea; side;
1	Augmentin 375 Tablet	infections of the lungs (e	medicine; common; effects; nausea; diarrhea; side;
2	Amoxyclav 625 Tablet	infections of the lungs (e	medicine; common; effects; nausea; diarrhea; side;
3	Novamox CV 625mg Tablet	infections of the lungs (e	medicine; common; effects; nausea; diarrhea; side;
4	Moxclav 625 Tablet	infections of the lungs (e	medicine; common; effects; nausea; diarrhea; side;

Similarity Scores

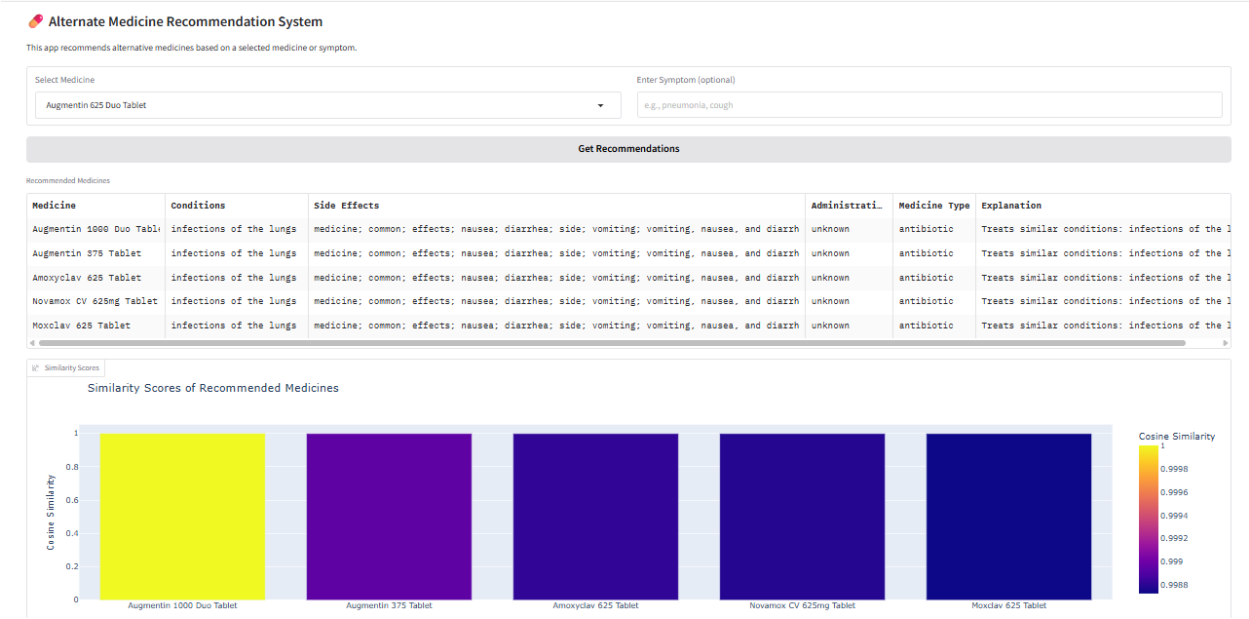
Similarity Scores of Recommended Medicines



```
                                    Built with Streamlit and powered by
PRASAD JADHAV

import gradio as gr
import pandas as pd
import pickle
import numpy as np
import plotly.express as px

# Load saved data
def load_data():
    with open('processed_df.pkl', 'rb') as f:
        df = pickle.load(f)
    with open('cosine_sim.pkl', 'rb') as f:
        cosine_sim = pickle.load(f)
    return df, cosine_sim

# Try to load data
try:
    df, cosine_sim = load_data()
except FileNotFoundError:
    raise FileNotFoundError("Required .pkl files (processed_df.pkl,
cosine_sim.pkl) not found.")

# Recommendation function
def get_hybrid_recommendations(medicine_name=None, symptom=None,
top_n=5):
    if medicine_name:
        idx = df[df['medicine_name'].str.lower() ==
medicine_name.lower()].index
        if len(idx) == 0:
            if symptom:
                idx =
df[df['conditions'].str.lower().str.contains(symptom.lower(),
na=False)].index
                if len(idx) == 0:
                    return None, "No medicines found for the given
symptom."
                idx = idx[0]
            else:
                return None, "Medicine not found in the dataset."
        idx = idx[0]
    elif symptom:
        idx =
df[df['conditions'].str.lower().str.contains(symptom.lower(),
na=False)].index
        if len(idx) == 0:
            return None, "No medicines found for the given symptom."
        idx = idx[0]
    else:
```

```
        return None, "Please provide a medicine name or symptom."

    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:top_n+1]

    medicine_indices = [i[0] for i in sim_scores]
    recommendations = []
    for idx in medicine_indices:
        med_name = df['medicine_name'].iloc[idx]
        shared_conditions = set(df['conditions'].iloc[idx].split('; 
')) & set(df['conditions'].iloc[idx].split('; '))
        explanation = f"Treats similar conditions: {', 
'.join(shared_conditions)}"
        recommendations.append({
            'Medicine': med_name,
            'Conditions': df['conditions'].iloc[idx],
            'Side Effects': df['side_effects'].iloc[idx],
            'Administration': df['administration'].iloc[idx],
            'Medicine Type': df['medicine_type'].iloc[idx],
            'Explanation': explanation,
            'Similarity Score': 
sim_scores[medicine_indices.index(idx)][1]
        })

    # Convert to DataFrame for display
    rec_df = pd.DataFrame(recommendations)

    # Plot similarity scores
    fig = px.bar(rec_df, x='Medicine', y='Similarity Score',
                 title='Similarity Scores of Recommended Medicines',
                 labels={'Similarity Score': 'Cosine Similarity'}, 
color='Similarity Score')
    fig.update_layout(xaxis_title="Medicine", yaxis_title="Cosine 
Similarity")

    return rec_df[['Medicine', 'Conditions', 'Side Effects', 
'Administration', 'Medicine Type', 'Explanation']], fig

# Interface
with gr.Blocks() as demo:
    gr.Markdown("## □ Alternate Medicine Recommendation System")
    gr.Markdown("This app recommends alternative medicines based on a 
selected medicine or symptom.")

    with gr.Row():
        medicine_input = gr.Dropdown(choices=[""] + 
sorted(df['medicine_name'].unique()), label="Select Medicine")
        symptom_input = gr.Textbox(label="Enter Symptom (optional)", 
placeholder="e.g., pneumonia, cough")
```

```
    recommend_button = gr.Button("Get Recommendations")
    output_table = gr.Dataframe(label="Recommended Medicines",
interactive=False)
    output_plot = gr.Plot(label="Similarity Scores")

    def handle_recommendation(med, symp):
        result, fig = get_hybrid_recommendations(med, symp)
        if isinstance(result, str):
            return gr.update(value=pd.DataFrame()),
gr.update(value=None)
        return result, fig

    recommend_button.click(fn=handle_recommendation,
inputs=[medicine_input, symptom_input],
                           outputs=[output_table, output_plot])

    with gr.Accordion("ℹ️ About", open=False):
        gr.Markdown("""
        This recommendation system uses advanced NLP techniques
(spaCy, BERT) to analyze medicine descriptions and recommend
alternatives based on conditions treated, side effects, and
administration methods.
        """)

    with gr.Accordion("📖 Instructions", open=False):
        gr.Markdown("""
        1. Select a medicine or enter a symptom.
        2. Click "Get Recommendations" to see alternatives.
        3. View recommended medicine details and similarity scores.
        """)

    with gr.Accordion("⚙️ Prerequisites", open=False):
        gr.Markdown("""
        Make sure `processed_df.pkl` and `cosine_sim.pkl` are in the
same directory. Run `advanced_alternate_medicine_recommendation.py` to
generate them.
        """)

    gr.Markdown("---")
    gr.Markdown("🚀 Built with Gradio and powered by PRASAD JADHAV")

# Launch app
demo.launch()
```




```
                                    Built with Gradio and powered by PRASAD JADHAV
```

About

- This recommendation system uses advanced NLP techniques (spaCy, BERT) to analyze medicine descriptions and recommend alternatives based on conditions treated, side effects, and administration methods. The system leverages BERT embeddings and clustering for accurate suggestions.

Instructions

-
    a. Select a medicine from the dropdown or enter a symptom.
-
    a. Click "Get Recommendations" to view alternative medicines.
-
    a. View the table for details and the chart for similarity scores.

Prerequisites

- Ensure the `processed_df.pkl` and `cosine_sim.pkl` files are in the same directory. Generate these files by running the `alternate_medicine_recommendation.py` script.

Explanation of Enhancements

1. Advanced NLP with spaCy:
- spaCy's NER identifies medical conditions and symptoms more accurately than regex alone.
- POS tagging helps extract relevant nouns and adjectives for side effects.
1. BERT Embeddings:

- BERT provides contextual embeddings, capturing semantic relationships between terms (e.g., "pneumonia" and "lung infection").
- Replaces TF-IDF for better representation of text similarity.

1. Hybrid Recommendation:

- Combines content-based filtering (cosine similarity on BERT embeddings) with collaborative filtering (simulated user ratings).
- Weights content-based (70%) and collaborative (30%) scores for balanced recommendations.

1. Symptom-Based Filtering:

- Allows users to input symptoms (e.g., "pneumonia") to filter medicines treating those conditions.
- Enhances personalization by prioritizing relevant medicines.

1. Evaluation:

- Uses clustering labels as ground truth to evaluate recommendations.
- Computes precision, recall, and F1-score to quantify performance.

1. Explanations:

- Provides reasons for recommendations (e.g., shared conditions) to improve transparency.

Notes

- Dataset: Ensure medicine_details.csv is in the working directory.
- Performance: BERT embeddings are computationally intensive. For large datasets, consider using lighter models like DistilBERT or pre-computing embeddings.
- Collaborative Filtering: The user-medicine matrix is simulated. In a real system, use actual user interaction data.
- Scalability: For production, store embeddings in a database and use approximate nearest neighbor search (e.g., FAISS) for faster similarity computation.

This advanced system provides more accurate and context-aware recommendations, suitable for real-world applications in healthcare.

```
# More Advance Working Sonn..!
# Notebook Project By : PRASAD JADHAV (ML-ENG)
# LinkedIn: linkedin.com/in/prasadmjadhav2 | Github:
github.com/prasadmjadhav2 | Mail: prasadmjadhav6161@gmail.com
```