Hyper-Personalization

Team Name: CodeBlooded

Date Mar 26, 2025

Owner : Team CodeBlooded

Team Members : Deshpande Riya; Singh Aditya K ; Motwani Ajay; Jada Srinivasarao; Santra Susmita

Problem Statement Overview

Modern day customers expect a better highly personalized experience matching their preference in product , services or content recommendation provided to them. Currently there are multiple platforms to showcase the social media , Online Shopping recommendations based on users habit. There are not many financial recommendation platforms for a user or an organization.

A report by Deloitte found that customers now demand the same level of sophistication, immediacy and personalisation in their interactions with banks as they do in other industries. They are even willing to share their data if assured of receiving personalized services.

Despite the increasing demand from customers for hyper-personalised services, implementation isn't always simple. A report titled 'Maximizing Digital Engagement', co-produced by Infosys Finacle and Qorus, revealed that although 71% of banks are running personalised campaigns and communications, less than 50% are leveraging enterprise customer data management, providing proactive advice and recommendations, data-driven micro-segmentation, and incorporating human-augmented sales, leveraging AI and ML recommendations.

Hyper-personalization in banking involves using data and advanced technologies like AI and ML to:

- Gain a deep understanding of each customer's needs, preferences, and behaviors.
- Offer highly targeted and personalized services that enhance customer satisfaction and engagement.
- Deliver tailored banking experiences to each customer by understanding their unique needs and using data to offer the right products and services.
- Craft personalized experiences that resonate deeply with each customer, using advanced data analytics, machine learning, and artificial intelligence.
- Recommend specific products or services based on individual preferences.

Proposed Solution

- We have developed a **Personalized Banking system** for a financial Institution to recommend financial products based on the users profile.
- We have leveraged data insights along with technology such as **Artificial Intelligence (AI), and Natural Language Processing (NLP).**
- The proposed **"Banking Personalization System"** provides a Generative AI-driven solution that enhances hyper-personalisation in the banking system by analyzing customer financial profiles, demographic details, interest domain.
- Personalized set of recommendations is provided each with a relevant score . User feedback is obtained for each of the recommendations. Based on this feedback the model is self trained and the recommendations are updated for each user.

Future Scope of the Developed Model

Hyper-personalized banking is not just a trend; it is a fundamental shift in how financial institutions engage with and serve their customers. The benefits, from enhanced customer satisfaction to increased loyalty and competitive advantage, make it a strategic imperative for banks in the digital age.

- The developed Banking Personalized System keeps a track of the recommendations explored by the user and self-trains itself to keep a track of the number of user interactions in each of the recommendations.
- Financial institutions can leverage this model and use these insights to provide customers with highly customized experiences.

Technical Stack

- Python
- Streamlit
- Data Store (Internal CSV file)
- AI Models
    - Hugging Face Model
    - Sentence Transformer

Functional flow details

- ## Data Processing

  We are using pandas to process raw data into data-frames and process it further for our model training.

  The code snippet for the same has been shared below:

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sentence_transformers import SentenceTransformer
from transformers import pipeline
import torch
import joblib
import os

class DataProcessor:
    def __init__(self):
        self.scaler = StandardScaler()
        self.sentence_model = SentenceTransformer('all-MiniLM-L6-v2')
        self.sentiment_analyzer = pipeline("sentiment-analysis", model="distilbert-base-uncased-finetuned-sst-2-english")
        self.product_classifier = pipeline("zero-shot-classification", model="facebook/bart-large-mnli")

    def load_financial_products(self, file_path):
        """Load and preprocess financial products data"""
        df = pd.read_csv(file_path)
        # Convert string lists to actual lists
        df['features'] = df['features'].apply(lambda x: x.split(','))
        df['tags'] = df['tags'].apply(lambda x: x.split(','))
        return df

    def create_product_embeddings(self, df):
        """Create embeddings for product descriptions"""
        product_texts = []
        for _, row in df.iterrows():
            text = f"{row['product_name']} {row['description']} {' '.join(row['features'])} {' '.join(row['tags'])}"
            product_texts.append(text)

        embeddings = self.sentence_model.encode(product_texts)
        return embeddings
```

- # Model Training

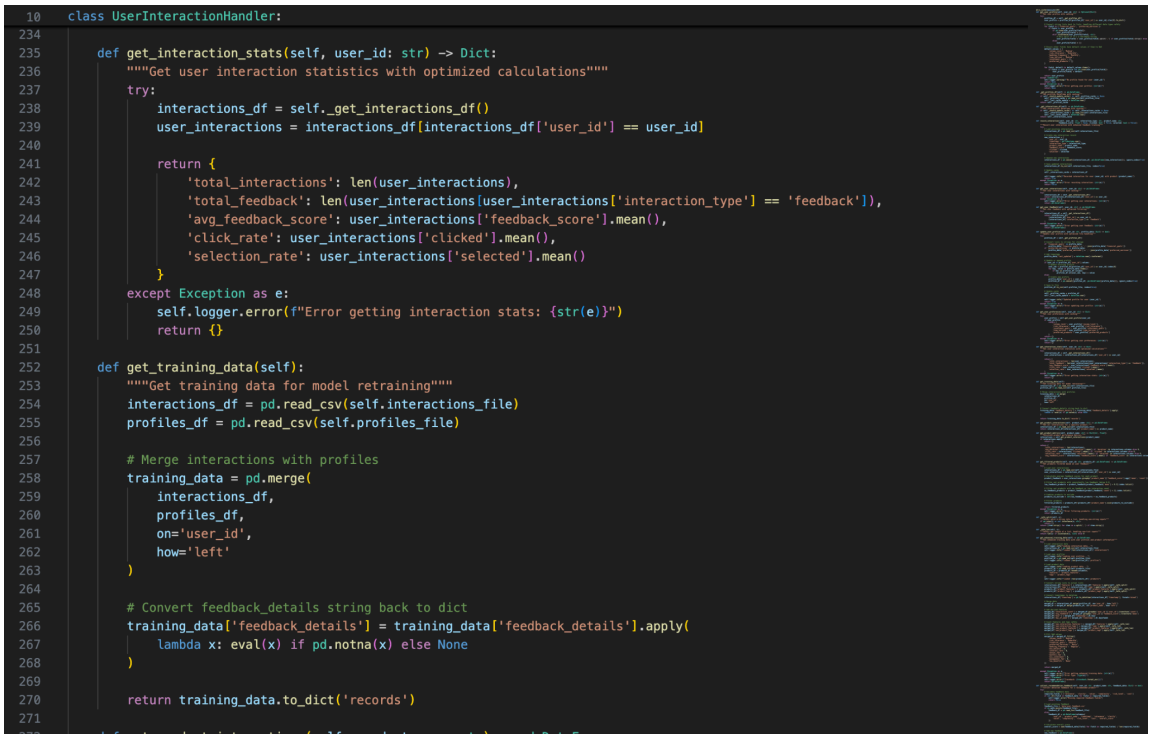
Once the data has been processed we are moving ahead with the model training. For the LLM model we have used Hugging Face's Sentence Transformer and Facebook's Bart for zero-shot classification.

The code snippet for the same can be found below:

```python
import pandas as pd
from utils.model_utils import HuggingFaceModels
from utils.user_interaction_handler import UserInteractionHandler
import logging

# Set up logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

def train_model():
    """Train the recommendation model with new data"""
    try:
        # Initialize models and handlers
        hf_models = HuggingFaceModels()
        interaction_handler = UserInteractionHandler()

        # Get enhanced training data
        logger.info("Loading training data...")
        training_data = interaction_handler.get_enhanced_training_data()

        if training_data.empty:
            logger.warning("No training data available. Please add some user interactions first.")
            return

        # Train the model
        logger.info("Training recommendation model...")
        mse, r2 = hf_models.retrain_model(training_data)

        logger.info(f"Model training completed!")
        logger.info(f"Mean Squared Error: {mse:.4f}")
        logger.info(f"R² Score: {r2:.4f}")

    except Exception as e:
        logger.error(f"Error training model: {str(e)}")

if __name__ == "__main__":
    train_model()
```

- # User Interaction Handling

  Based on the user interactions which includes the views to the recommended products and feedback provided by the user we are retraining the model.
  The code snippet to handle the user interaction is given below:



```python
class UserInteractionHandler:

    def get_interaction_stats(self, user_id: str) -> Dict:
        """Get user interaction statistics with optimized calculations"""
        try:
            interactions_df = self._get_interactions_df()
            user_interactions = interactions_df[interactions_df['user_id'] == user_id]

            return {
                'total_interactions': len(user_interactions),
                'total_feedback': len(user_interactions[user_interactions['interaction_type'] == 'feedback']),
                'avg_feedback_score': user_interactions['feedback_score'].mean(),
                'click_rate': user_interactions['clicked'].mean(),
                'selection_rate': user_interactions['selected'].mean()
            }
        except Exception as e:
            self.logger.error(f"Error getting interaction stats: {str(e)}")
            return {}

    def get_training_data(self):
        """Get training data for model retraining"""
        interactions_df = pd.read_csv(self.interactions_file)
        profiles_df = pd.read_csv(self.profiles_file)

        # Merge interactions with profiles
        training_data = pd.merge(
            interactions_df,
            profiles_df,
            on='user_id',
            how='left'
        )

        # Convert feedback_details string back to dict
        training_data['feedback_details'] = training_data['feedback_details'].apply(
            lambda x: eval(x) if pd.notna(x) else None
        )

        return training_data.to_dict('records')
```