Final Project

A. Project Overview

Goal: My project aims to develop a clothing recommendation system that can recommend clothes based on an individual piece of clothing. I want the user to be able to change specific parameters to make the recommendation system more suited to their taste.

Dataset: Fashion Products

B. Data Processing

I loaded the CSV file directly into Rust by uploading it to the folder of my VSCode project. Because the file is not particularly big, there was nothing to tweak in the dataset. However, when loading the data, I never used the User ID field as it did nothing for my recommendation system.

C. Code Structure

Modules:

preprocessing.rs: This file contains the majority of the code for my project. It houses functions for building feature matrices and adjacency lists from the data in the dataset, as well as a struct that would allow me to smoothly print the attributes of each clothing item in my output.

graph.rs: This module was used for the Graph struct, which allows me to access the neighbors of specific nodes easily. Primarily useful for the recommendation function in my main file.

main.rs: This file puts everything from the other modules together, and is what prints the final output. Also contains the recommendation function.

Key Functions and Types:

fn load_and_preprocess: reads a CSV, encodes numeric and categorical columns, normalizes numerics, and returns a feature matrix of shape (n_samples, n_features). Takes a file path and categorical/numerical column labels as its inputs. One key component of this function is that it does two passes over the data. The first pass is for simply inserting the data into specific columns, and the second pass scales the numerics while turning the categorical labels into one-hot categories. These are stored in one matrix, which is then returned right after.

struct Item + fn load_metadata: Item is a struct that simply notes data down for easy access later, and the load_metadata function takes the data from each line of the CSV to store as items. load_metadata takes a file path as its only input. The function is quite self-explanatory, as it simply takes the appropriate data from each element and stores the data in a new Item instance.

fn build_graph_from_features: turns the feature matrix from the load_and_preprocess function and builds a graph containing all the nodes and the neighbors for each node. This function takes a feature matrix and a value k for its inputs. First takes the norms of all the nodes, then calculates the cosine similarity of each node with the rest of the nodes. Once this is done, the top k nodes with the highest similarity scores are added to the graph as a vector. Each node will have its unique list of nodes it is closest to.

type AdjList: a simple data type for the Graph struct. Same data type as the adjacency list to be used for creating graphs later.

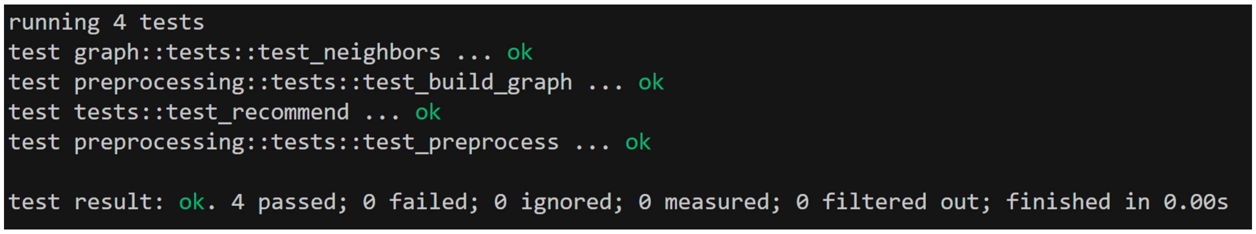
struct Graph: takes the adjacency list from the previous function as input to create a new Graph. The implementation only includes the new function and the neighbors function, which gets all the neighbors of a specified node.

fn recommend: uses the neighbors function of Graph to get the top k neighbors of a given node. Takes as input a graph, a node number, and a value k. Implementation is simple enough, as it clones the top k neighbors of the specified node and collects them into one vector.

Main Workflow:

The algorithm for my project is as follows. First, the data is sorted through the load_and_preprocess and build_graph_from_features functions. Moreover, the data is stored in items through the load_metadata function. From there on, the adjacency list built from build_graph_from_features is used to create a Graph object, which is then used to get the top k recommendations for a given node (the node is specified by the user in the terminal). These node values are then used to get information from the items list to print out the recommendations in the terminal.

<u>D. Tests</u>



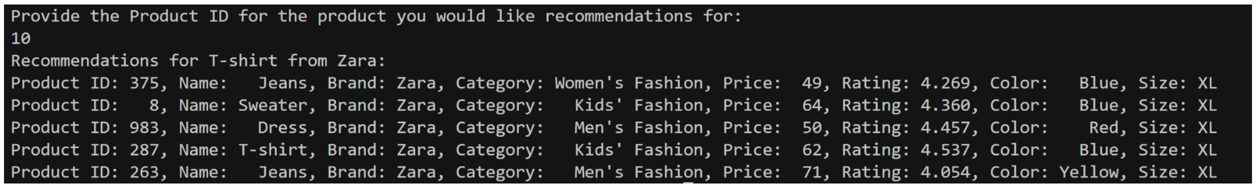
```
running 4 tests
test graph::tests::test_neighbors ... ok
test preprocessing::tests::test_build_graph ... ok
test tests::test_recommend ... ok
test preprocessing::tests::test_preprocess ... ok

test result: ok. 4 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
```

test_neighbors: This test makes sure that the Graph struct and neighbors function within the Graph struct work properly. This test matters because the neighbors function is essential to the recommendation system working.

test_build_graph: This test checks if the build_graph_from_features function works. It is important as the adjacency list created from this function is used to get recommendations later, and any flaws in the function will create inaccurate recommendations.
fn test_recommend: makes sure the recommend function gets the right values and the right amount of them. Important for the final output to return the right results.

test_preprocess: tests the load_and_preprocess function, ensuring that the resulting matrix has the right dimensions. Extremely important for building an accurate adjacency list with the build_graph_from_features function.

E. Results



```
Provide the Product ID for the product you would like recommendations for:
10
Recommendations for T-shirt from Zara:
Product ID: 375, Name:   Jeans, Brand: Zara, Category: Women's Fashion, Price:  49, Rating: 4.269, Color:   Blue, Size: XL
Product ID:   8, Name: Sweater, Brand: Zara, Category:   Kids' Fashion, Price:  64, Rating: 4.360, Color:   Blue, Size: XL
Product ID: 983, Name:   Dress, Brand: Zara, Category:   Men's Fashion, Price:  50, Rating: 4.457, Color:    Red, Size: XL
Product ID: 287, Name: T-shirt, Brand: Zara, Category:   Kids' Fashion, Price:  62, Rating: 4.537, Color:   Blue, Size: XL
Product ID: 263, Name:   Jeans, Brand: Zara, Category:   Men's Fashion, Price:  71, Rating: 4.054, Color: Yellow, Size: XL
```

While not a particularly impressive output, this achieves the goal of my entire project. I have asked my recommendation system for recommendations on Clothing Piece 10, and specified that it should find products with a similar price and rating, and should also come from the same brand in the same size. I would say that this output definitely satisfies my objectives.

F. Usage Instructions

Running my code is very simple. Depending on your preferences, you can change the numeric/categorical column labels in the main function. Then, you can use cargo run to run the project. At this stage, you will be asked to provide a Product ID in the terminal for which you would like recommendations. Any value from 1-1000 is a valid Product ID. Once inputted, the program will automatically output recommendations based on your specified item and parameters. If you would like, you can also increase or decrease the k value at the top of the main function to get more recommendations.