Report

Khalil Ouald Chaib

February 2025

1 Part 1

1.1 Data Analysis and Preprocessing

The first step I took was to examine the provided dataset. I observed that six rows had missing values in the style column. Additionally, I noticed a data imbalance, which, combined with the limited dataset, could negatively impact the results. I also identified and removed several duplicate rows. Furthermore, I eliminated unnecessary columns that were not required for Part 1 of the assignment. The remaining columns were: barcode, style, tops_fit, bottoms_fit, sleeve_type, pattern, more_attributes, image_url_1, and brand.

Next, I downloaded all the images and resized each one to 300×300 pixels, adding padding when necessary. This resizing was necessary because the model expects images in this specific input size. I then created a new column called image_path to store the file paths of the downloaded images. Afterward, I dropped the image_url_1 column, as it was no longer needed. The cleaned dataset was then saved as a CSV file in data/cleaned_data.csv.

The final step was encoding the text data in the CSV file. Since the output label columns could have multiple labels, I applied multi-hot encoding to encode them. Additionally, I used one-hot encoding for the brand column. However, I chose not to encode the barcode column, as I determined that this information would not contribute to the model's learning, barcodes are unique identifiers and do not carry meaningful patterns for prediction.

Before passing images to the model, they are preprocessed using a series of transformations. These transformations include random horizontal flipping, random rotation, conversion to tensor format, and normalization. These preprocessing steps help enhance model generalization by introducing slight variations in the dataset, preventing overfitting.

1.2 Model Architecture

The model expects two inputs, one is the image and the second input is the brand. In this section I will explain every part of my model architecture. As you may notice the barcode is not part of the input. The barcode serves as a unique identifier for each clothing item, meaning it does not contain inherent information that helps the model learn fashion attributes. Including it as a feature would not contribute to generalizable patterns.

1.2.1 Image Feature Extraction

The model utilizes EfficientNet-B1, a deep learning CNN model pre-trained on ImageNet, as the backbone for image feature extraction. This choice is motivated by the following considerations:

- Efficiency and Accuracy: EfficientNet models achieve superior performance compared to traditional CNN architectures like ResNet.
- The classifier head (fully connected layer) of EfficientNet-B1 is removed (self.efficientnet.classifier = nn.Identity()), allowing the model to output a 1280-dimensional feature vector for each input image.

1.2.2 Brand Feature Processing

Fashion brands often carry implicit information about style, quality, and target demographics. To incorporate this metadata, a separate fully connected (FC) network processes brand-related features. The brand input undergoes a transformation through a series of layers.

1.2.3 Fusion and Classification

The extracted image and brand features are concatenated into a single feature vector before passing through a final classification head:

- Concatenation (torch.cat((image_features, brand_features), dim=1)): Combines the 1280-dimensional image feature vector with the 64-dimensional brand feature vector, resulting in a 1344-dimensional fused feature representation.
- Final Output Layer (512 $\rightarrow$ num_classes): Produces predictions for multiple metadata attributes.

The final output layer is designed to support multi-label classification, allowing each clothing item to belong to multiple categories simultaneously.

1.3 Results, Challenges and improvements

- Test Accuracy(number of rows that are completely correct without any mistakes/total number of rows): 12.50%

- Hamming Loss: 0.0196
- Precision: 0.8576
- Recall: 0.5296
- F1 Score: 0.6325

To handle the class imbalance, I would try oversampling the underrepresented classes during the data loading process. This would ensure that the model sees more samples from the minority classes. Unfortunately, I did not have enough time to implement this.

Additionally, I would like to experiment with modifying the model architecture by adding or removing layers to observe the impact on performance. However, due to the long training times, I was unable to test this.

2 Part 2

The recommendation algorithm is rule-based, using predefined fashion guidelines for the three body types. Each body type has specific clothing attributes that enhance its proportions. The rules were created using an LLM because of my own lack of knowledge within the fashion world.

The algorithm assigns a score to each item in the dataset based on how well it aligns with the selected body type. The scoring follows these steps:

- Retrieve Rules: The algorithm fetches predefined fashion rules for the given body type.
- Define Conflicting Rules: It also identifies attributes associated with other body types to penalize mismatches.
- Compute Score: A match between the item's attributes and the target body type's rules increases the score. A match between the item's attributes and a conflicting rule decreases the score.
- Rank Items: The dataset is sorted based on score, and the top N items are returned as recommendations.

One potential improvement is incorporating user purchase history into the recommendation process. By analyzing past purchases, the system can infer user preferences for specific styles, colors, and fits. This would allow the algorithm to provide more personalized recommendations by combining predefined fashion rules with user-specific data.

3 Part 3

I chose to use the Flask library to build the API because it is the only library I have experience with for API development. To ensure scalability, I implemented a Kubernetes setup with horizontal scaling.

To start the Kubernetes service, run the build.sh script. If you need to delete the service and the Docker image, use the delete.sh script.

Once the Kubernetes service is running, execute the test_api.py script to send a few requests to the API. The responses will be printed to the console.