**Intelligence Beyond Fashion**

**Group Name:** Group 1

**Group Members:**

|  |  |  |
| --- | --- | --- |
| **First name** | **Last Name** | **Student number** |
| Nikeshh | Vijayabaskaran | C0849544 |
| Jebin | George | C0850509 |
| Deeksha | Naikap | C0835440 |
| Shravan Kumar Reddy | Gunthala | C0833124 |
| Raghavendra Reddy | Putta Hanumantha Reddy | C0851724 |

**Submission date: 22nd April 2023**

*Intelligence Beyond Fashion*

*Instructions to Run the Project*

1. Structure:

The project delivered contains the following folders.

|  |  |  |
| --- | --- | --- |
| **ID** | **Folder** | **Description** |
| **1** | **API** | Contain the postman api collection which can be imported in postman. We have also saved the examples that we tried.  There are three main api endpoints:   1. default or health – GET API – to check the health of the application 2. findrelativeimages – POST API – takes text and image input and provides the predicted list of text and images 3. lda – GET API – return current trends html visualization |
|  |  |  |
| **2** | **Chrome Extension** | Chrome extension to scrape the image and text data on amazon website and returns the similar products |
| 2.1 | ibf | The chrome extension folder to be loaded |
|  |  |  |
| **3** | **Flask App** | Flask web application to perform necessary model operations |
| 3.1 | \_\_pycache\_\_ | Cache files – can be ignored |
| 3.2 | app.py | Main flask file with all the api endpoints |
| 3.3 | compose.yaml | File descriptor for the docker application |
| 3.4 | Dockerfile | File used for docker build |
| 3.5 | embeddings | All the text and image embeddings are saved here |
| 3.6 | google\_cloud\_key.json | Google cloud key to access the google cloud bucket. Sentence transformer because of its huge size it is saved in google cloud bucket which will be pulled when the flask application is run. |
| 3.7 | input | CSV file which contains all the product details |
| 3.8 | models | All machine learning models are saved here. Sentence transformer because of its huge size it is saved in google cloud bucket which will be pulled when the flask application is run |
| 3.9 | README.md | Readme file for flask application. It can be referred for running the flask application |
| 3.10 | requirements.txt | Python packages requirements. These packages needs to be installed before running the flask application using the command  pip install -r requirements.txt |
| 3.11 | template | Folder which is used for saving the lda html file that gets generated |
| 3.12 | upload | Image files sent through are saved into this upload folder |
|  |  |  |
| **4** | **Intelligence Beyond Fashion – Core / Model** | Contains all the core files of the project including the model build files etc |
| **4.1** | **TrendAnalysis** | Find new trends based on the twitter tweets and topic modelling |
| 4.1.1 | lda\_vis.html | Sample visualization html created using lda |
| 4.1.2 | lda\_vis.pkl | Lda model pickle file |
| 4.1.3 | Topic Modelling\_LDA and NMF (sneakers).ipynb **(Refer to section 2 to know more about file structure and how the model is created)** | Notebook file / Code for the Top modelling |
| 4.1.4 | tweets\_data.csv | Tweets collected from twitter |
|  |  |  |
| **4.2** | **TrendMatch** | Matches the given image and text across a set of products |
| **4.2.1** | **Data, Models, Build and Test files** | Contains data, models, build files and test files |
| 4.2.1.1 | autoencoder\_decoder\_first\_image\_model.h5 | Autoencoder decoder first image saved model file |
| 4.2.1.2 | autoencoder\_decoder\_first\_image\_model.png | Autoencoder decoder first image model structure |
| 4.2.1.3 | autoencoder\_decoder\_second\_image\_model.h5 | Autoencoder decoder second image saved model file |
| 4.2.1.4 | autoencoder\_decoder\_second\_image\_model.png | Autoencoder decoder second image model structure |
| 4.2.1.5 | cnn\_image\_model.h5 | CNN Image saved model file |
| 4.2.1.6 | data | Input image data folder separated by folders / labels |
| 4.2.1.7 | data.zip | Input image data folder zip |
| 4.2.1.8 | filedata | Datafile generated using knn mapping |
| 4.2.1.9 | input.csv | Input CSV file which contains all the details of the products |
| 4.2.1.10 | kmeans\_model.pkl | KMeans model pickle file |
| 4.2.1.11 | knn\_model.pkl | KNN model pickle file |
| 4.2.1.12 | metadata | Input metadata containing the product details of various labels |
| 4.2.1.13 | metadata.zip | Input metadata folder zip |
| 4.2.1.14 | own\_image\_model\_finetune\_embedding.npy | Image model finetune embedding generated based on the choosen image model |
| 4.2.1.15 | pretrained\_best\_model\_efnb5.h5 | Pretrained Model – EfficienetB5 – Used for comparison |
| 4.2.1.16 | pretrained\_efficient\_net\_b5\_finetune\_embedding.npy | Pretrained Model Finetune Embeddings |
| 4.2.1.17 | pretrained\_reconstructive\_image\_model.h5 | Pretrained Reconstructive Image model generated based on the arcface generation with pretrained model |
| 4.2.1.18 | reconstructive\_own\_image\_model.h5 | Reconstructive Image model generated based on the arcface generation with any of the own image models |
| 4.2.1.19 | sentence\_transformer\_text\_embeddings.npy | Sentence Transformer Text embeddings file trained with the product titles |
| 4.2.1.20 | sentence\_transformers\_text\_model.sav | Sentence transformer text model |
| 4.2.1.21 | test | Test folder containing single test file |
| 4.2.1.22 | X\_encoded\_compressed.npy | KNN encoded data |
| **4.2.2** | **Python files** | Trend Match code |
| 4.2.2.1 | Intelligence\_Beyond\_Fashion.ipynb **(Refer to section 3 to know more about file structure and how the model is created)** | Notebook file / Code for the TrendMatch |
| 4.2.2.2 | intelligence\_beyond\_fashion.py **(Refer to section 3 to know more about file structure and how the model is created)** | Python file version of Intelligence\_Beyond\_Fashion.ipynb notebook – Created directly from google colab |
| **5** | **ReactJS website -> ibf-web** | Contains the code for reactjs website |
| **6** | **ReactNative app -> ibf-mobile** | Contains the code for react native app |

1. TrendAnalysis folder and code explanation:

*TrendAnalysis is about finding new trends based on Twitter tweets and topic modelling.*

Topic Modelling\_LDA and NMF (sneakers).ipynb will have the following structure:

1. **Install Dependencies:** Before starting the project, you will need to install the necessary dependencies such as Tweepy (for accessing the Twitter API), Pandas (for working with data), Matplotlib (for data visualization), NLTK (for natural language processing), and so on. You can install these libraries using pip or conda depending on your preference and environment.
2. **Import and Construct API:** Once the dependencies are installed, you need to import them into your Python code and construct an API object using your Twitter developer account credentials. This will allow you to access the Twitter API and collect data.
3. **Collect Tweets and Populate Dataframe:** With the API object in place, you can start collecting tweets based on specific search criteria (such as keywords, hashtags, user handles, etc.) and store them in a Pandas dataframe. You can use Tweepy's Cursor object to iterate through the search results and add them to the dataframe.
4. **Save Dataframe to CSV:** Once you have collected the tweets and populated the dataframe, you can save it to a CSV file using Pandas' to\_csv() method. This will allow you to store the data and use it later for analysis or visualization.
5. **Cleaning Dataframe:** Before you can perform any meaningful analysis on the collected tweets, you need to clean the dataframe by removing irrelevant columns, duplicates, and any other unwanted data. You can also perform data cleaning tasks such as removing special characters, punctuation, stop words, and so on.
6. **Tokenize the Sentences:** Once the dataframe is cleaned, you can tokenize the sentences (i.e., split them into individual words) using NLTK's word\_tokenize() method. This will allow you to perform further analysis on the text data.
7. **Visualization:** With the tokenized data in hand, you can start visualizing the text data using Matplotlib or other visualization libraries. You can create different types of plots such as bar charts, histograms, scatterplots, and so on to gain insights into the data.
8. **Generating Word Cloud:** Another popular way of visualizing text data is by generating word clouds. A word cloud is a visual representation of the most common words in the text data, where the size of each word corresponds to its frequency in the text. You can use Python libraries such as wordcloud or matplotlib to generate word clouds.
9. **LDA:** Latent Dirichlet allocation: LDA is a topic modeling technique used to extract topics from text data. It works by identifying common patterns in the text and grouping them together into topics. You can use Python libraries such as gensim or sklearn to perform LDA on the tokenized data.
10. **Non-negative Matrix-Factorization:** Non-negative matrix factorization (NMF) is another unsupervised learning technique used to extract topics from text data. It works by decomposing the text data into two matrices: one that represents the topics and another that represents the document-term frequency. You can use Python libraries such as sklearn to perform NMF on the tokenized data.
11. TrendMatch folder and code explanation:

*TrendMatch is about matching the given image and text across a set of products.*

Intelligence\_Beyond\_Fashion.ipynb will have the following structure:

1. **GPU Setup and Rapids API**

This sections contains the details gpu setup and rapids api setup.

Rapids API (<https://rapids.ai/>) is used for improving the processing of data using the Graphical Processing Unit.

This is not completely used throughout the project however learning how to use them was definitely helpful

1. **Preparation**

This section contains the necessary preparation code for our project

* 1. **Mount Google Drive and Install the dependencies**

Connect to the google drive to load the data and save files in the cloud

* 1. **Load dependencies**

This section loads the necessary packages or dependencies

* 1. **Importing data and exploration**

The image data and metadata is loaded from the cloud, exploration and preprocessing is carried out on the data

* 1. **Plotting**

The function in this section helps in plotting the image data based on the given parameters.

1. **Text Model and Prediction**

This section contains the code for the text model used for creating embedding for the product titles and used for product title similarity prediction.

The text model is more of a supplementary model to our project, and hence the pre-trained sentence transformer and TFIDF vectorizer were used. The image model discussed in section 4 is the main part of our project.

* 1. **Sentence Transformer**

This section contains the code for the sentence transformer text model and its embedding generation based on the product title. Prediction of the title is also carried out in this section.

* 1. **TFIDF Vectorizer**

This section contains the code for the TFIDF vectorizer text model and its embedding generation based on the product title. Prediction of the title is also carried out in this section.

1. **Image Model and Prediction**

This section contains the code for the image model used for creating the embedding of the image data and used for image similarity prediction.

* 1. **Own models**

This section contains the code for our own models used for finding similar images and the generation of the image embeddings.

* + 1. **Data Preparation**

In case of autoencoder decoder model, low resolution images are required to process the data and hence the image data is converted into low resolution images.

* + 1. **Model 1 – Autoencoder decoder model**

This section contains the autoencoder decoder model – type 1 which contains less number of layers and processing.

This section also contains the code for finding the similar images using this model.

* + 1. **Model 2 – Autoencoder decoder model**

This section contains the autoencoder decoder model – type 2 which contains more number of layers and processing when compared to type 1.

This section also contains the code for finding the similar images using this model.

* + 1. **Model 3 – CNN model**

This section contains the cnn model. This section also contains the code for finding the similar image using the model.

* + 1. **Choosing the model**

Based on the processing and efficiency of the above three models, one model is choosen among them.

* + 1. **KNN Similarity Matching**

**This section contains the code for similarity matching using clustering data using K-Means and using K-NN to find the nearest neighbours**

* + - 1. **Extraction of features**

Features are extracted from the encoded image data

* + - 1. **K-Means to cluster image data**

The extracted features and encoded data are clustered using K-Means. Labels and centroids are also found.

* + - 1. **K-NN to find the nearest neighbours**

K-NN is used to find the nearest neighbours

* + - 1. **Predictions finding**

The input image is then processed to find the nearest neighbours using the K-NN data.

***It should be noted that this method didn’t give the proper results we expected.***

* + 1. **Cosine Similarity Matching**

The cosine values of the input data are found and stored as NumPy arrays in the latent space.

The input image is then processed to find similar images using the NumPy values.

* 1. **Pretrained model – EfficientB5**

This section contains the code for an efficient b5 pre-trained model to find similar images. This model or code was only used as a comparison with our own models. It should be noted that for similarity predictions, only cosine similarity prediction or matching method is used, but the KNN similarity matching method is not implemented.

1. **Combined Prediction**
   1. **Preparation**

This section contains the code for converting the test product title and test product image into embeddings to be given to the models for prediction. Also both the text and image embeddings are combined.

* 1. **Prediction**

This section contains the code for prediction by comparing the cosine values of the model combined embeddings data and inputs embeddings data.

1. **References**

This section contains the necessary references or citations used for achieving text and image similarity matching.

1. Important Note

We trained all our code and model using Google Colab Pro+ with High RAM – 85GB ram. Since the dataset contains a huge set of images, training this huge dataset requires huge processing power.

Otherwise, we were getting session crashes and out-of-memory errors.

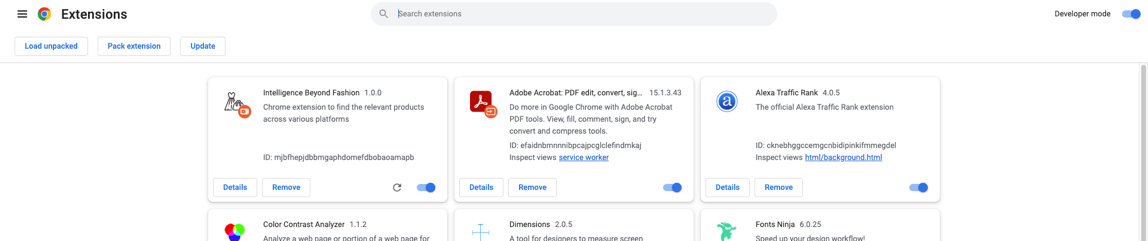
1. Run the Flask application:

Make sure that the python 3.8 version is used to be compatible with the package versions used in our project

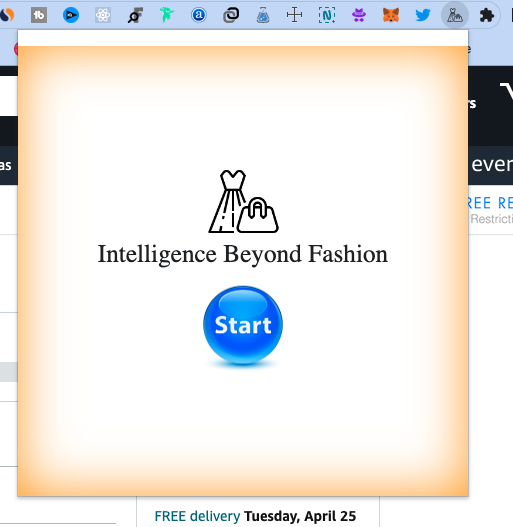
It is preferable to use a conda environment. If possible in the conda environment, create a new environment and start running the application

Import Flask App folder in VSCode, in terminal at the root of the application, in the terminal run

1. pip install -r requirements.txt
2. “export FLASK\_APP=app.py” to set the entry point of the application
3. “flask run” to run the application
4. Run the chrome extension:
5. Go to Google Chrome -> Extensions -> Manage Extensions
6. Make sure Developer Mode is enabled
7. Click on Load unpacked and load the complete folder “ibf” present inside Chrome Extension folder



1. Now, go to any listing on Amazon.
2. Click on the “Intelligence Beyond Fashion” extension in the extension bar and click on the start button



1. The extension will scan the product image and title and get the relevant products.



1. Run the ReactJS website:

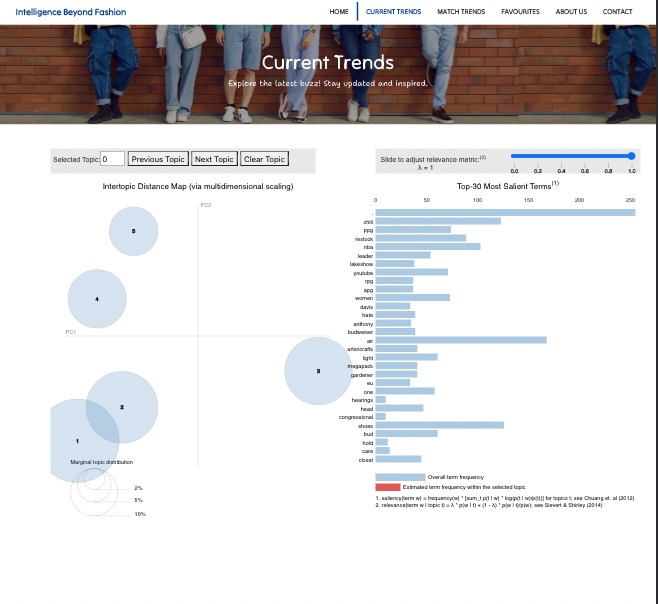
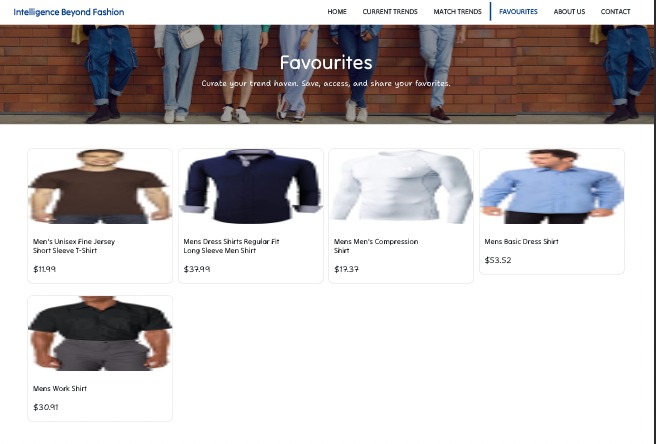
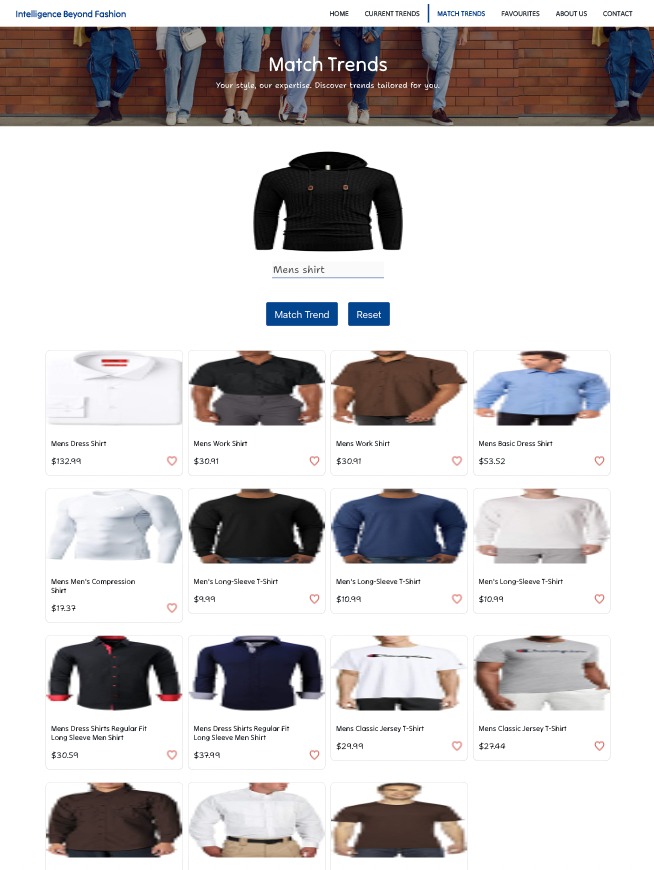
Before running the ReactJS website, make sure the latest version of

* + - 1. nodejs – v16.15.1
      2. npm – v8.11.0

are installed.

Import ReactJS Website folder and go to ibf-web folder in VSCode, in terminal at the root of the application, in the terminal run

1. “npm install” to install the dependencies
2. “npm start” to start the project



1. Run the react native app:

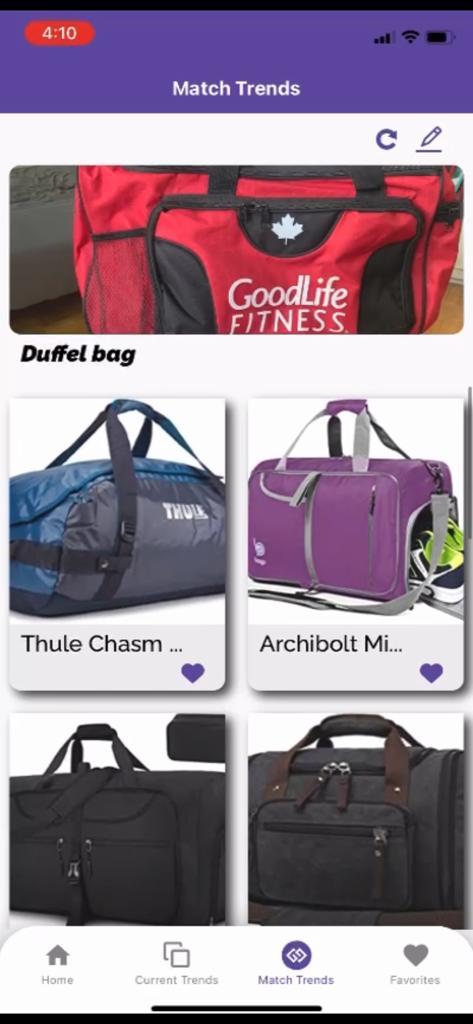
Before running the React Native app, make sure the latest version of

1. nodejs – v16.15.1
2. npm – v8.11.0

are installed.

Import the ReactNative App folder and go to ibf-mobile in VSCode, in terminal at the root of the application, in the terminal, run

1. “expo install” to install the dependencies.
2. “expo start” to start the app





*Thanks for the wonderful opportunity. Contact us (Intelligence Beyond Fashion Team) for any questions.*

*By*

Nikeshh Vijayabaskaran,

Raghavendra Reddy,

Shravan Kumar Reddy,

Jebin George,

Deeksha Naikap