Report on Project

Motivation and Background

In previous coursework, we focused extensively on text data, so in this course, I aim to challenge myself with multimodal data, integrating more unstructured types, such as images, sounds, and videos. This project specifically centers on feature extraction and effective alignment of multimodal features. Additionally, it serves as an initial step toward building a knowledge graph, a field that greatly interests me.

The project draws from an e-commerce background, with data sourced from *Shopee*, a leading online shopping platform. Product matching is crucial in e-commerce, as it addresses significant real-world business challenges. Effective product matching enables companies to offer competitive pricing on identical products sold by different retailers. However, visually similar images of different products may either represent the same item or entirely unrelated ones. Retailers seek to avoid the confusion and misrepresentation that may arise from incorrectly matching dissimilar products.

Currently, a combination of deep learning and traditional machine learning methods is employed to analyze image and text information for similarity comparison. However, notable variations in images, titles, and product descriptions often prevent these methods from achieving optimal accuracy.

Dataset Overview

We can download the dataset using Kaggle api: `kaggle competitions download -c shopee-product-matching`, and the dataset can be divided into 3 main parts:

- *[train/test].csv* - the training set metadata. Each row contains the data for a single posting. Multiple postings might have the exact same image ID, but with different titles or vice versa.
  `posting_id` - the ID code for the posting.
  `image` - the image id/md5sum.
  `image_phash` - a perceptual hash of the image.
  `title` - the product description for the posting.
  `label_group` - ID code for all postings that map to the same product. Not provided for the test set.
- *[train/test] images* - the images associated with the postings.

- *sample_submission.csv* - a sample submission file in the correct format.
  posting_id - the ID code for the posting.
  matches - Space delimited list of all posting IDs that match this posting. Here we focus on label matches.

Upon examining the training set, I found it contains 34,250 postings across 11,014 label groups, whereas the test set has only 3 postings available for evaluation. Since this project is part of a Kaggle competition, the complete test set—comprising roughly 70,000 hidden postings—can only be evaluated after submission, which also demands an effective evaluation method.

Additionally, I noticed that the title field contains certain characters like '\x', which are related to text encoding issues. To address this, I implemented an encoding format conversion during the text preprocessing step to ensure proper handling of such characters.

Approaches and Model selection

I tried to approach the issue using three methods:

- Generate text matches from text features and image matches from image features, then combine the text and image matches.
- Concatenate text and image features to produce combined matches.
- Create a union of combined matches, text matches, and image matches.

Approach 3 can be considered a state-of-the-art (SOTA) method, as it makes the most comprehensive use of the available information.

So, how to extract the features and how to find "matched" pairs?

Text (*text.py*)

For text features, I utilized *TF-IDF* method to extract key words first (coarse filtering), and then applied *BERT* to extract contextual features, which can capture more complex semantic information (fine filtering). By combining the similarity calculation results of these two methods, the robustness of the model can be further improved and the mismatch that may be caused by a single method can be avoided.

Currently the similarity thresholds are hard-coded (e.g. 0.3 for *TF-IDF* and 0.8 for *BERT*). These values can be tuned through methods such as cross-validation to find the best similarity threshold.

Images (*image.py*)

When it comes to images, I applied *EfficientNet* with *ArcFace* for feature extraction. The EfficientNetWithArcFace class combines the *EfficientNet* model (pre-trained on ImageNet) for feature extraction and the *ArcFace* loss function for improved representation learning. The final classification layer is removed to focus solely on feature extraction.

The model is trained using the Adam optimizer with a learning rate of 0.0001. For each epoch, images are passed through the model, and the *ArcFace*-enhanced features are used to compute the loss. After training, the model is switched to evaluation mode. Features are extracted for all images in the dataset using the trained model. The features are stored in image_features, while corresponding labels and posting IDs are also saved.

A K-Nearest Neighbors (*KNN*) model is used to find similar products based on the extracted features. The cosine similarity metric is employed to measure how similar two feature vectors are. For each image in the dataset, the KNN algorithm finds the most similar products (based on their feature vectors). The results are stored in a dictionary, similar_products, where the key is the posting_id of the image, and the value is a list of similar product posting ids. *Self-matches (i.e., a product being its own nearest neighbor) are excluded from the similarity results.

Combination (*combine.py*)

I implemented concatenation of BERT extracted text features and image features, and then utilized KNN clustering method to find combined matches.

At last, I tried to evaluate the union of combined matches, text matches, and image matches.

Evaluation

Because of scarcity of the test set mentioned before, I tend to use cross-validation score to evaluate the matching result, which can reduce overfitting, and ensure generalization to unseen data.

In implementation, I also tried *CLIP* model to align text and image features in the same space. However, the inherent defect of the pre-trained model is that it allocated too much computation memory, and the character limit is 77 while the longest title after pre-processing has 255 characters. So, to simplify the implementation, I subset 10% of the original train set. And in CLIP model, I truncated the title if its length is beyond 77.

Another embarrassing point is the trade-off between accuracy and recall. I initially implemented approach 3 and got a bad result. To find out the underlying causes, I calculated the real average number of similar products for each posting in the training subset and it is only **2.1**. That means that I cannot release the threshold in previous matching, if I release the tolerance, the recall can be improved while the accuracy would decrease, which determines a low F score.

So in my final submission, I gave up incorporating the combined matches and got the current best result (though still far from expectation): accuracy is 0.1401, recall is 0.2385 and F1 score is 0.1765.

Limitation and Future Work

The project gave me a glimpse into the challenges and intricacies involved in multimodal data processing. My implementation is more like an exploration rather than an optimal solution in a real Kaggle competition. It kind of implies what optimization problems look like in reality: the demanding design of solution, new problems always arise in implementation and the result tend to deviate from expectations.

Limited by time and resources, I didn't fully implement the designed solution. I managed to make the implementation end-to-end and thought about several possible future improvements:

- Enlarge training size

I just subset 10% of the training set this time, and I believe the result with more training data could be more robust.

- Optimize evaluation method

As I said in the evaluation part, the long implementation process brings out complexity and relatively high-tolerance matching results, which leaves limited adjusting space faced with a low size of real matched pairs. I am wondering if there exists a more suitable evaluation metric for this specific task.

- Adjusting parameters and fine-tuning the pre-trained model

With the implementation of multiple steps, many parameters need to be hard-coded, such as the thresholds for similarity and the size of neighbors that can be called "matched". I tried to adjust a lot based on current testing results, but it may be far from an optimal combination.

Additionally, after optimizing the evaluation method, the pre-trained model can be fine-tuned according to cross-evaluation results.