School of
Computing and
Information Systems

CS611: Machine Learning Engineering

Project Report: Retail Products Classification for E-Commerce

Group 2

Members:
Aloysius Teng (aloysiust.2021@mitb.smu.edu.sg)
Dang Thi Mai Vy (tmvdang.2022@mitb.smu.edu.sg)
Muhamad Ameer Noor (ameern.2022@mitb.smu.edu.sg)
Neel Ketan Modha (neelmodha.2022@mitb.smu.edu.sg)

Contents:

I. Business Problem

Background

Classifying products into different categories ís a compulsory step whenever sellers want to upload new products to e-commerce platforms. However, as e-commerce grows and product categories expand rapidly, product classification becomes tedious and prone to human error. Furthermore, the reliability of product categorization can have significant impacts on downstream systems. For example, if products are assigned incorrect or inconsistent categories, it can diminish the effectiveness of recommendation and search engine, ultimately affecting the buyer experience on e-commerce platform. Consequently, this leads to lower satisfaction levels for both sellers and buyers. As such, this project aims to provide a product classification model that aims to minimise errors and provide sellers with a better product uploading experience.

At the end of the project, we aimed to build a prototype that allow sellers to classify one product at a time by uploading the image and adding descriptions of the products that they sell. The classification model integrated in the system will then recommend what category should the seller tag their product as. For future development, features such as product bulk-upload are desirable to improve user experience of sellers that has a large variety of product in their catalogue.

Dev-ops solution

Our project focused on automating end-to-end pipeline of product classification model development. The solution on the pipeline encompassed the entire machine learning lifecycle starting with data collection and pre-processing, model training, evaluation, registration and deployment. The core component of our solution is a combination of models that leverages both the product's image and accompanying title or text description to generate category recommendations.

Resource-wise, the solution employed distributed system to support model training and load balancing. Employing a distributed system means utilizing multiple machines or nodes to distribute the workload and increase the efficiency of model training. This approach helps to reduce the training time by parallelizing the computations across multiple nodes. Load balancing techniques ensure that the training workload is evenly distributed among the available resources, optimizing resource utilization and improving overall system performance.

Considering seller as the user of the platform that needs to immediately list their product on the e-commerce platform, the deployment involves near-real time inference solution when sellers upload their product. For near real-time inference, the trained machine learning models need to be deployed in a production environment where they can process incoming data and provide predictions quickly. This involves setting up a scalable and robust deployment infrastructure, such as a containerization. By deploying the models in containers, the infrastructure can be easily scaled up or down based on the demand, ensuring sellers' product uploads are processed rapidly.

For further development, the solution needs to integrate model outputs with downstream applications such as Search and Recommendation. Integrating model outputs with downstream applications involves connecting the output of the machine learning models to other systems or services that depend on them. In the context of search and recommendation, this could mean incorporating the model's predictions into search algorithms or recommendation engines. This integration is achieved through APIs (Application Programming Interfaces) or message queues, allowing seamless communication and data flow between the machine learning models and downstream applications.

Future expansion will also involve monitoring model performance via seller feedbacks to trigger incremental training and redeployment (CICD). Continuous Integration and Continuous Deployment (CI/CD) practices are vital in a DevOps solution. Monitoring model performance involves collecting feedback from target users or users interacting with the downstream applications. Our solution partially simulates this on the user inference, where feedback is collected. The feedback includes whether the recommendation is correct or incorrect, and what is the category that the seller chose in the end.

By analysing this feedback, the e-commerce platform can identify areas where the model's performance can be improved. Incremental training refers to retraining the models with new data or fine-tuning

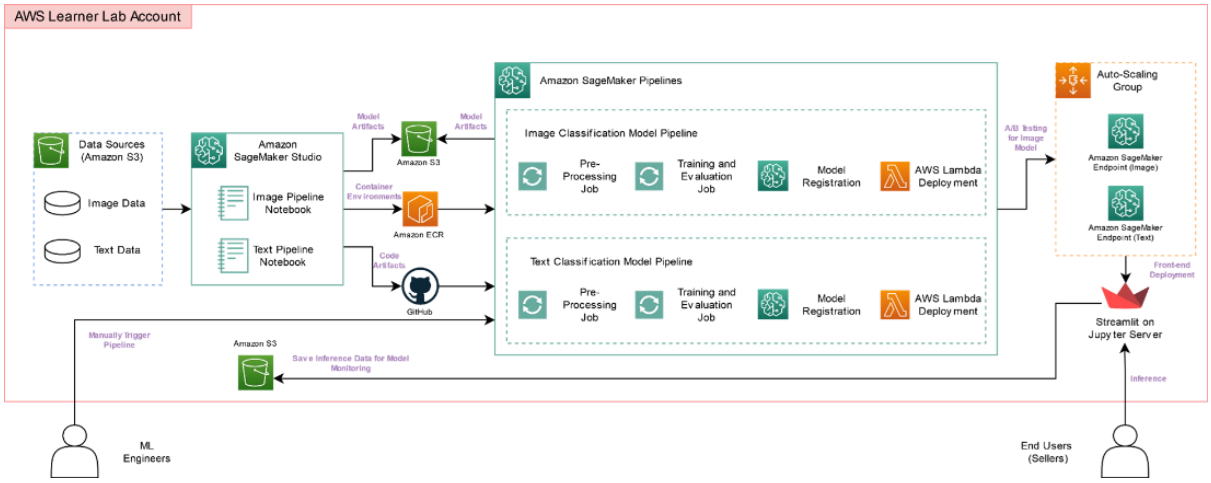
existing models based on the feedback. Once the model is retrained, the CI/CD pipeline is triggered to redeploy the updated model into the production environment.

Why Machine Learning?
Considering the goal to automate arduous task of labelling products, simpler option like rule-based system is a possible solution. However, product classification involves text and image information which is too complex to capture by rule-based system. On the other side, machine learning system keeps improving and able to learn such complex information by using on frameworks such as Neural Network. Therefore, Machine Learning is needed as the solution to automate labelling products.

The e-commerce business also trends towards increasing number of product varieties. Machine learning process can be incremental, thereby it is highly adaptable to include new products and new categories without manual work. If there is a change in the category tree, we can easily update the model with new category mapping to retrain and re-deploy.

Target Users
For the prototype deployment that we build, the main target is to simulate Seller experience on e-commerce where they upload new products and get support on recommendation what label should they tag the product with. On potential expansion, the system could be deployed in the E-Commerce company where the internal Data Scientist Team can monitor model performance to maintain feedback and continuous improvement of the model. Additionally, the team can also build expansion of use case such as fraud detection and product recommendation. These improvements will eventually enhance buyer experience too, when they're shopping in the platform.

II. Overall Architecture



The overall pipeline architecture that we build was illustrated above. At the start, both image and text data were stored on Amazon S3. Each data type was undergoing pre-processing and modelling on their own pipeline. The resulting endpoints are deployed in Streamlit where sellers experience in uploading product is simulated.

III. Data Collection and Local Pre-processing

This product uses Retail Classification Dataset sourced from Kaggle[1] with the following attributes:

- **ImgId**: Unique image name for the product
- **title**: Name of the product

[1] Retail Product Classification Dataset (Kaggle): https://www.kaggle.com/competitions/retail-products-classification/data

- **description**: A brief description of the product
- **category**: Name of the category the product belongs to – our target variable

Images in .jpg format are stored in a separate folder with each image name corresponding to an "ImgId". Here are some data samples:

	ImgId	title	description	categories
0	B000HYL1V6	TUNGSTEN SOLDER PICK WITH HANDLE	Solder Pick for picking up molten solder when ...	Arts, Crafts & Sewing
1	B00006HXWY	Write Right 98167 Screen Protector for Sony T615C	We all screen. And we all need to protect thos...	Cell Phones & Accessories
2	B000GAWSBS	Casio Mens DBC310-1 Databank 300 Digital Watch...	Bringing you precision at a glance, the Casio ...	Clothing, Shoes & Jewelry
3	B000040JOL	Factory-Reconditioned DEWALT DW260KR Heavy-Dut...	Factory-Reconditioned DEWALT DW260KR Heavy-Dut...	Tools & Home Improvement

img_name
B000HYL1V6.jpg
B00006HXWY.jpg
B000GAWSBS.jpg
B000040JOL.jpg

The Kaggle dataset has been pre-processed into "train" and "test" data. However, as "category" data – the target variable - is only provided for the train set, we only used "train" dataset for this project. The "train" dataset has 46,229 products evenly distributed over 21 categories with 2000 – 2225 products per category. There are 42,000 images available, which were pre-resized to 3x100x100. The collected data was then pre-processed:

(1) Preliminary exploration of the data shows that some categories are similar as "All Electronics" & "Electronics", "All Beauty" & "Beauty". As such, we merged these similar categories to create 18 new encoded categories:

Office Products	8
Cell Phones & Accessories	9
Pet Supplies	10
Industrial & Scientific	11
Tools & Home Improvement	12
Grocery & Gourmet Food	13
Patio, Lawn & Garden	14
Toys & Games	15
Musical Instruments	16
Sports & Outdoors	17

Electronics	0
All Electronics	0
Arts, Crafts & Sewing	1
Appliances	2
Health & Personal Care	3
Beauty	4
All Beauty	4
Automotive	5
Clothing, Shoes & Jewelry	6
Baby Products	7
Baby	7

(2) For image model, we filtered for only products with images, which are 42,000 out of 46,229 total products.

IV. Model Pipelines

1. Model descriptions

Text Classification Model Development

Specifically, for text classification model development, we utilize Sagemaker JumpStart BERT Text Classifier, which provides a foundation for our text classification tasks. BERT is a Pretrained Language Model (PLM) which is especially suited for text understanding and classification. We fine-tuned the model using Sagemaker Automatic Model Tuning to optimize its performance. The resulting accuracy of the text classifier is 0.83%.

Image Classification Model Development

For image classification model development, we employed Sagemaker MXNet Image Classifier, a pre-built image classification model in SageMaker, for this task. With one epoch-training within the pipeline, model accuracy was 29%. With further training outside the pipeline context, the resulting accuracy of the image classifier was improved to 51% with 30 epochs. Hence, model accuracy could be improved further with more training resources and time.

2. Pipeline Development

Due to resource constraint in Learner Lab, two separate pipelines were created for text and image models in parallel to decouple model development processes and provide an alternative back-up for later deployment in case one of the pipeline failed. Here are the pipeline structures of the two models:

Figure 1: Image Model pipeline

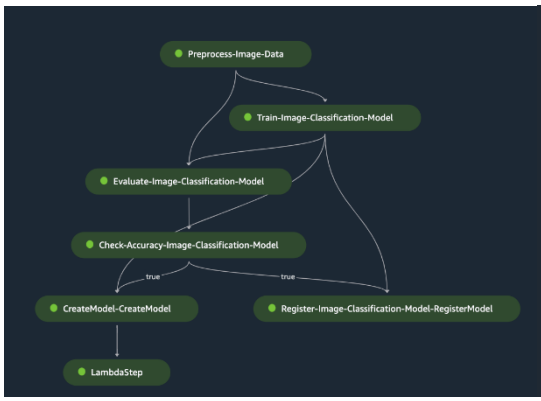
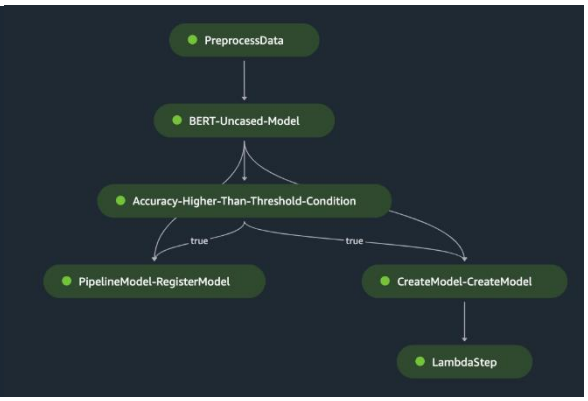


Figure 2: Text Model Pipeline



Detailed descriptions of the pipeline steps are as follow:

(1) Pre-processing Step:
- Text: Join "title" and "description" into a single feature; Remove headers and indices
- Image: Convert image data into .rec files (input format for image model)
- For both pipelines: split data into train/test/validation datasets

(2) Training Step:
- Fine-tuning the built-in SageMaker Text Classifier (BERT Pre-trained Language Model) and Image Classifier (MXNet) which were already pre-trained
- Training Instance Type: "ml.p2.xlarge" (GPU)
- Instance Count: 1
- Epochs: up to 3
- Mini batch size < 100
- Resulting Accuracy of 0.83 for Text Model and 0.29 for Image Model

(3) Evaluation Step:
- Models evaluated based on Classification Accuracy.
- For the text model, evaluation step was skipped as the model artifact could not be loaded in the 'evaluation.py' script. Instead, the condition for deployment was based on the training validation accuracy.

(4) Model Create, Register and Lambda Deployment:
- For Image Classification pipeline where classification accuracy was relatively low, model was only registered and deployed if model accuracy passed a specified threshold. Due to training resource constraint (*), the image model's accuracy threshold is set at 0.2 for demonstration purpose of later steps such as Model Register and Deployment. For the text model, the accuracy threshold was set at 0.75.
- Lambda deployment was used to automate the two model deployments to two real-time endpoints

(*) With Learner Lab account, training capability was limited to GPU instance type of "ml.p2.xlarge" and one instance per training session. As such, the number of epochs and mini batch size to fine tune the models were also greatly limited, else container memory would be exceeded. With this container configuration, it could take 1-2 hours for training 3 epochs, and 3-4 hours for the whole pipeline. With higher-tier account type, training instance type and the number of instances could be upgraded to allow higher training capacity and better model accuracy.

3. Prediction Generation

Input data including product description and image are submitted to both image and text model endpoints. The predicted probabilities from the text and image classifier's endpoint responses are combined using weights equal to the accuracy of each model. This is to ensure that the model with the

higher accuracy is given more priority. The ensembled probability results are used to decide the top 5 category predictions for the target product.

V. Model Deployment and User Experience

We deployed our image and text models using an **Online, Real-Time Inference Pipeline**. This is to ensure we can return predicted categories to our users with the lowest latency possible, as we expect this system to be used quite frequently with sellers uploading new products around the clock. The models are deployed as part of the ML pipeline, where the last step of the pipeline is a **Lambda Step for Automated Model Deployment** which deploys the image and text models if upon evaluation, each model crosses a certain threshold of accuracy. If a model fails to cross the threshold, the endpoint will continue to make inferences based on the previously trained model.

After the Lambda Step for deployment, a **TargetTracking Auto-Scaling** policy is attached to both endpoints, allowing them to scale out automatically in periods of higher traffic. This auto-scaling policy tracks two metrics: (i) *CPUUtilization* (target value 90%, after which it starts to scale out) and (ii) *SageMakerVariantInvocationsPerInstance* (target value 10 after which it starts to scale out), with a maximum capacity of two instances. This is to prevent the risk of downtime, especially during periods of high traffic such as e-Commerce campaigns, where sellers are uploading thousands of products at once. As part of further improvements, we would ideally like to add these auto-scaling policy assignments to the Lambda Step directly, to further automate the process. Additionally, for the image model, we simulate the deployment of two image models in an **A/B Testing Deployment Strategy** (simulation because both the A- and B-variants are the same model in our project). We simulate this showcase that it is feasible for ML engineers to test two different types of image (or text) models simultaneously, and compare their performance.

Once the two endpoints are deployed, we **deployed a lightweight Streamlit application hosted locally on a Jupyter Server**, to showcase the capabilities of our ML system as a prototype. The application allows users to upload a product image and its associated product title/description. Once both fields are entered, the user can push the button "Make Inference", which will generate the top 5 most likely categories and their associated probabilities for the user to assign to their uploaded product.

Our Streamlit application is able to call both image and text endpoints with the .invoke_endpoint() method which is part of the SageMaker runtime client. Once the user hits the "Make Inference" button, two sets of results are returned to the Streamlit application, and we combine both sets of inferences from the image and text models using the **accuracy of each model as a weight**, using the formula as shown below, for each product category:

This ensures that if in the future, the image or text model's accuracy starts to drop, predictions from that model will have a lower weightage to the final predictions shared with the seller. For now, we have hard-coded the accuracy values of the two models, as well as the endpoint names of the two models that the .invoke_endpoint() method calls. As part of further improvements, we would ideally like to reduce the reliance on hard-coded values and further automate this by referencing these values directly from the model package attributes saved in the model registry.

VI. Model Monitoring

To ensure the continuous performance and accuracy of our models, we monitor our models primarily by collecting data from inferences made by sellers. This allows us to review the collected data, assess the need for retraining, and make informed decisions to ensure the models remain accurate and effective. The continuous monitoring and retraining process enables the system to adapt to evolving user needs and improve the overall performance of the classification system over time. We collect three main types of data:

1. **Saving User Input Data:** We save the image and text input data provided by users to augment our original training dataset, as part of model re-training on this new data, and CI/CD in the future. By continuously collecting user input data, we can gather a diverse range of real-world examples, which helps to improve the model's ability to handle various scenarios and edge cases. [Attributes saved: (i) User Input Image (JPG), and (ii) User Input Title/Description]
2. **Saving User Feedback:** Once an inference is returned to the user, we gather feedback by asking users a simple question: "Did you find the predictions accurate". This is an explicit way to understand whether our users are finding our predictions accurate. In addition to this, if users select "Yes" to the previous question, we then prompt them to select which of the top 5 categories they selected in the end. If the selected "No" to the previous question, then we give them the ability to select the right category for their product, as a way to find out the ground-truth seamlessly. This information provides valuable insights into user preferences and helps evaluate the relevance and effectiveness of the suggested categories. If users consistently find our predictions inaccurate and choose categories with lower confidence levels, it indicates that our model may need adjustments to better prioritize relevant categories. While we did not build this into our existing ML system, as part of future improvements, this user feedback should be analysed in-depth by model monitoring jobs to detect model drift, and when re-training may be necessary, as part of CI/CD. [Attributes saved: (i) Yes/No on whether user found predictions accurate, and (ii) Final product category selected]
3. **Saving Model Inferences:** We also save what is the actual inference model our image and text models output to the user. By comparing which product category the user finally selected with the outputs of both our image and text models, we can determine whether one or both of our model outputs are aligned with the ground truth, and to trigger re-training and re-deployment if one or both of our image and text model starts to drift. [Attributes saved: (i) 18 categories and their probabilities based on Image model, and (ii) 18 categories and their probabilities based on Text model]

The above three sets of user feedback are all **saved to an S3 bucket** every time an inference is made (.JSON file for string/numerical variables, .JPG file for images). We identify each inference made by generating a **unique UUID per inference** that allows us to related user inputs, user feedback, and model inferences made, for model monitoring jobs. Based on these model monitoring jobs, ideally, the process of re-training and re-deployment would be automated, triggered by accuracy-based thresholds, or when significant drift starts to occur. This is an area which we would explore deeper in the future based as part of CI/CD.

VII. Limitations, Risk Mitigation, and Potential Expansion

While our system aims to provide accurate and efficient product classification, it is important to acknowledge potential challenges and plan appropriate mitigation strategies. The following are potential problems that may arise and corresponding measures to address them:

Limitations

1. Limited Classification Accuracy:
   a. Improve dataset quantity and quality: Continuous data collection efforts should focus on expanding the dataset with diverse and representative examples. Gathering additional labelled data can help improve the accuracy of the models.
   b. Improve prediction result with more training: The current solution was built with limited epochs on the training part due to limited resources to do the training. For better results, more training is recommendable if more resource is available
   c. Improve prediction result with better algorithms: Due to limited resource, the current solution was also using SageMaker built-in algorithms. Exploring more advanced algorithms or architectures may capture more complex patterns and enhance classification performance, should more resource available.
2. Slow training: Due to the same limited resource challenge, the training process is running slow. If more resource available, increasing number or upgrading training instances can solve the challenge.

3. The built solution was not yet a fully integrated pipeline due to limited account functions. With a higher-tier Amazon account, the solution can fully automate model development and deployment among pipelines by sharing the same S3 storage.
4. Limited variety of goods classification: To address this limitation, efforts should be made to expand the range of product categories that the models can accurately classify. This can be achieved by acquiring additional labeled data for underrepresented categories or employing transfer learning techniques to leverage pre-trained models from related domains.

Risk Mitigation

1. Adversarial attack: To mitigate the risk of adversarial attacks, which involve intentional manipulation of input data to deceive the classification system, implementing robust defence like input validation and anomaly detection methods can help identify and reject malicious inputs. Regular monitoring of the system's performance and anomaly detection techniques can help detect any suspicious activities or potential attacks.
2. Intentional system overloading: To prevent intentional system overloading, mechanisms such as rate limiting, request throttling, and load balancing can be implemented. These techniques ensure that the system can handle a reasonable amount of traffic and protect against malicious attempts to overload the infrastructure.
3. Scalability and Performance: As the system gains popularity and user traffic increases, ensuring scalability and maintaining optimal performance becomes crucial. Employing technologies like load balancing, horizontal scaling, and efficient resource allocation can help handle a larger user base and maintain fast response times.
4. Data Privacy and Security: Protecting user data and ensuring compliance with privacy regulations are very important. On the real-case deployment, implementing robust data encryption, access controls, and regularly auditing the system's security measures can help safeguard sensitive information and maintain user trust.

Potential Expansion

1. Fraud Detection: As a future development, integrating fraud detection mechanisms can help identify intentional misclassification of products by fraudulent sellers. For example, the same monitoring system that tracks low performance when user selecting least probable labels, can be utilized to also track the particular user if the user consistently doing so. By tracking the user and notifying the Data Scientist team, manual check on whether the incorrect labelling is intentional can be analysed.
2. Multilingual Support: Extending the system to handle multiple languages can cater to a broader user base. This may involve collecting labelled data for different languages, using language translation techniques, or employing cross-lingual transfer learning to adapt the models to different languages and cultures.
3. Product Similarity and Sub-Categorization Labelling: Expanding the system to handle finer-grained categorization or product similarity analysis can provide more detailed recommendations to users. This can involve exploring techniques like hierarchical classification, clustering, or embedding-based approaches to identify similarities between products and enable more precise categorization.

In general, to ensure the system's long-term effectiveness, continuous improvement is essential. This includes monitoring user feedback, tracking performance metrics, and collecting additional data to refine the models. Regular evaluations and feedback analysis help identify areas of improvement and guide future development efforts.

By addressing these potential challenges and implementing mitigation strategies, the system can evolve to handle a wider range of product classifications, enhance accuracy, and provide a robust and reliable solution which was mainly for online retailers, but can be expanded to other users such as buyers and internal data scientist team.