AI Usage

Throughout all tasks, I used ChatGPT mainly to help me understand errors, clean up my code, and write try/except blocks to catch errors.

Task 1

Data & Prep

For this task, I collected data using a Python script that pulled product names from the UK Open Food Facts website. I chose the UK section so I wouldn't get items in multiple languages. After cleaning the text with pandas and regex, I mapped all products into a custom set of categories that made more sense for grocery shopping.

I then added extra data from a grocery list dataset on Hugging Face (https://huggingface.co/datasets/AmirMohseni/GroceryList). To fill in missing categories (like Household or Pet Supplies), I tried scraping multiple retail websites. Only Pets at Home worked well; Boots had too much promotional content, and it was difficult to pick out only item names (even with CSS selectors) and Eataly wasn't reliable.

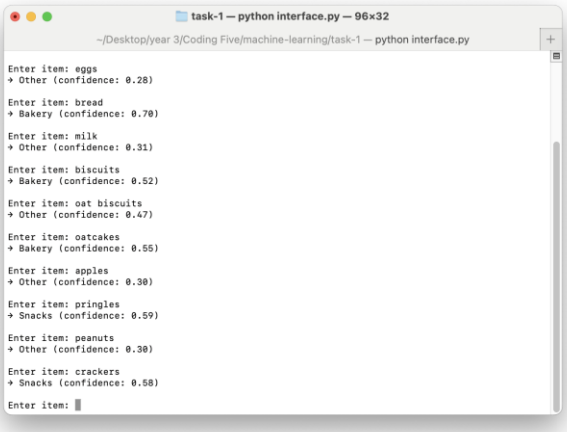
Examples of junk items scraped from Boots:

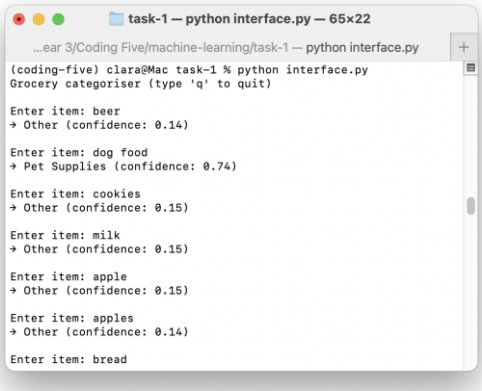
```
please stand by,Personal Care
maximum basket size reached,Personal Care
out of stock,Personal Care
select your shipping destination,Personal Care
la roche-posay,Personal Care
no7,Personal Care
olay,Personal Care
boots,Personal Care
fenty beauty,Personal Care
laneige,Personal Care
skincareunder 5,Personal Care
skincareunder 10,Personal Care
skincareunder 15,Personal Care
eucerin,Personal Care
l'oréal paris,Personal Care
liz earle,Personal Care
benefit,Personal Care
beauty of joseon,Personal Care
elemis,Personal Care
k-beauty,Personal Care
mediheal,Personal Care
vitamin c,Personal Care
peptides,Personal Care
spf,Personal Care
retinol,Personal Care
boots skincare tool,Personal Care
```

The scraping helped grow the dataset but added a lot of messy text, so most of my time ended up being spent cleaning what I collected.

I also added an extra manual dataset that I wrote with some items I felt are more essential.

Before and after scraping:





Model

For Task 1, I used the Keras Functional API because my model needed to handle text input in several steps. My data started as strings and moved through a lot of different layers. The

Functional API made it easier to control the flow and see exactly how the text moved through the model. It also gave me more flexibility to add or remove layers without breaking anything, which was good for experimenting with different combinations of layers.

At first, the model performed well for categories like Bakery and Snacks. But after adding scraped data, only Pet Supplies really improved, because it became a large, consistent category.

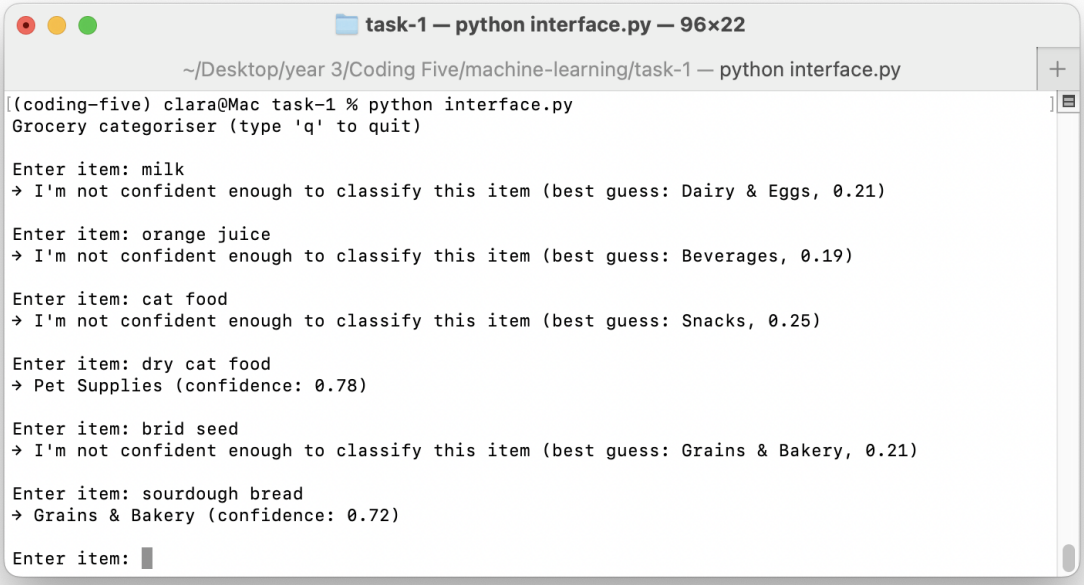
Explanation of model choices (most of these choices were either tips from lectures or trial and error):

- I used sparse categorical cross-entropy as the loss function since all labels were stored as integers, not one-hot vectors
- The model's accuracy increased significantly, from around 0.2 to approximately 0.7 on the training set, after adding an embedding layer, a GlobalAveragePooling1D layer, and a Dropout layer. I experimented with removing or changing these components, but this configuration consistently gave the best results. I also tested different activation functions, optimisers, and loss functions, as well as reducing the Dense layer size from 64 to 32, but none of these changes improved performance.

Steps taken to improve accuracy:

- I tried increasing sequence length from 20 to 30 and capping categories at different sizes (100–130 items each), but neither improved the accuracy much.
- I added bigrams (ngrams = 2) to the text vectoriser since many grocery items consist of two-word phrases. This resulted in an accuracy increase of around 10%.
- I merged smaller and overlapping categories (e.g. Meat & Seafood + Deli -> Meat and Deli), which improved accuracy because it made each category bigger.
- In interface.py, I lowered the confidence threshold, which allowed the model to make more predictions. Even though confidence scores were low, the predicted categories were correct most of the time.

In the final version, I found that the model performed much better when given longer product names. For example, 'cat food' was misclassified as Snacks, while 'dry cat food' was correctly classified as Pet Supplies. This likely happened because longer phrases more closely resembled the training data and reduced ambiguity.



Final Thoughts & Future Improvements

Most of the time spent on this task went into collecting and cleaning the data. I think that the main reason the model does not achieve very high accuracy is the nature of the dataset itself. The categories are uneven in size, and many of them don't have enough examples, which makes it difficult for the model to learn patterns. In the future, I would prioritise finding additional data sources (especially simpler data) or cleaning the existing dataset even more.

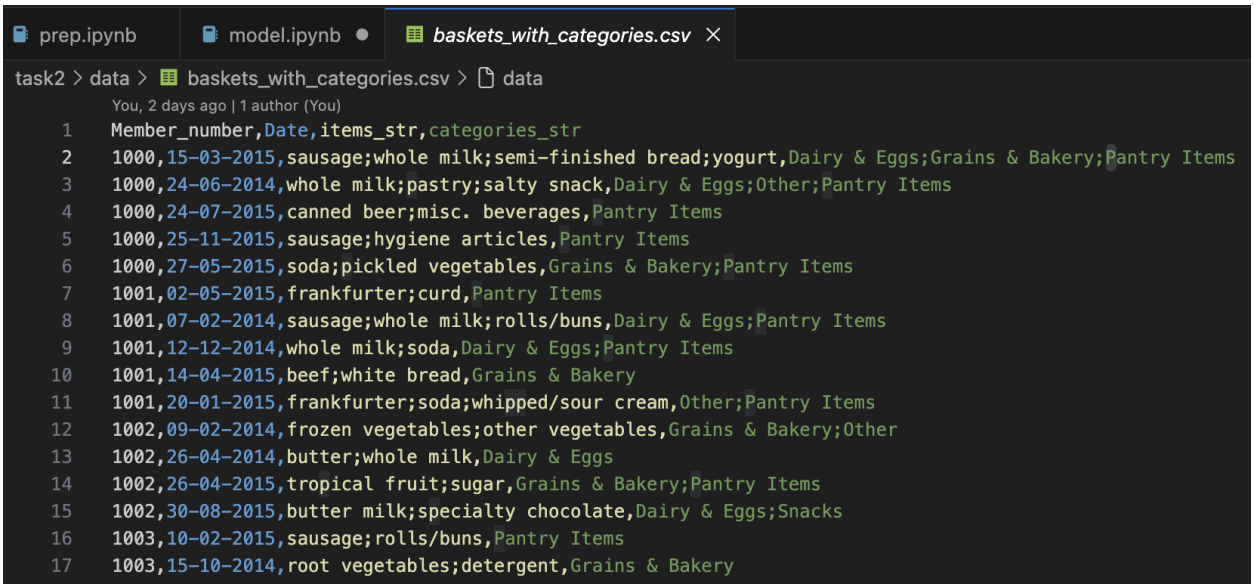
Task 2

Data & Prep

The first step was sorting the grocery data so it could be used for recommendations. The original Kaggle dataset (https://www.kaggle.com/datasets/heeraldedhia/groceries-dataset) listed one item per row, so I grouped the data by customer and date to turn it into shopping baskets. Each basket (row) then represented a single shopping trip containing multiple items.

I then used the text classification model from Task 1 to predict a category for each item in a basket. I added an extra column to the dataset with the list of categories for each basket.

This was the final dataset that I only saved to check the data was correct:



One challenge during this stage was the performance. The Task 1 model had to run on every item in every basket, which made the preparation step very slow (took around 16 minutes to run the whole dataset).

Another challenge at this stage was that I first started Task 2 by working across two separate notebooks: one for preparing the data and one for building the model. At the beginning this made sense, but I realised that both notebooks needed to run the exact same categorising code. Having to repeat code in two different files became confusing and made the workflow harder to follow.
I also realised that I didn't actually need to save a final dataset anywhere, because the only thing that mattered was having the categories predicted before training the recommendation model. For that reason, I decided it was much simpler to merge everything into one notebook. Instead of doing the category prediction in a separate prep file, I moved that step directly into the model file, right before using the Task 2 model. Because there was not much prep needed anyways, the code became much easier to understand.

I also learned that I couldn't import functions from a Jupyter notebook into interface.py, so I moved the final recommendation functions directly into the interface file.

The Model

For Task 2, I chose the Sequential API because the model was much simpler. The inputs were already numbers, and the model only needed a few dense layers stacked one after

another. Since everything flowed in a clean, single direction, Sequential felt more natural and easier to read.

Instead of predicting specific items, this model predicts what category is most likely to be missing from the basket. This approach ensures the suggestion makes sense (e.g., it won't recommend "toilet cleaner" with "fresh fruit" unless the basket actually fits that pattern).

The way I trained the model was that I turned each basket into multiple training examples by hiding one category at a time and teaching the model to predict it from the remaining categories.

A big challenge was realising I needed to use multi-hot vectors, since each basket can contain multiple categories; this representation allowed all categories to be encoded simultaneously in a fixed-size numeric format that the model can process.

Final Thoughts & Future Improvements

Even though it was challenging to manage two models at the same time, this task was more enjoyable, and I liked learning how to use task 1 model to feed into this model.

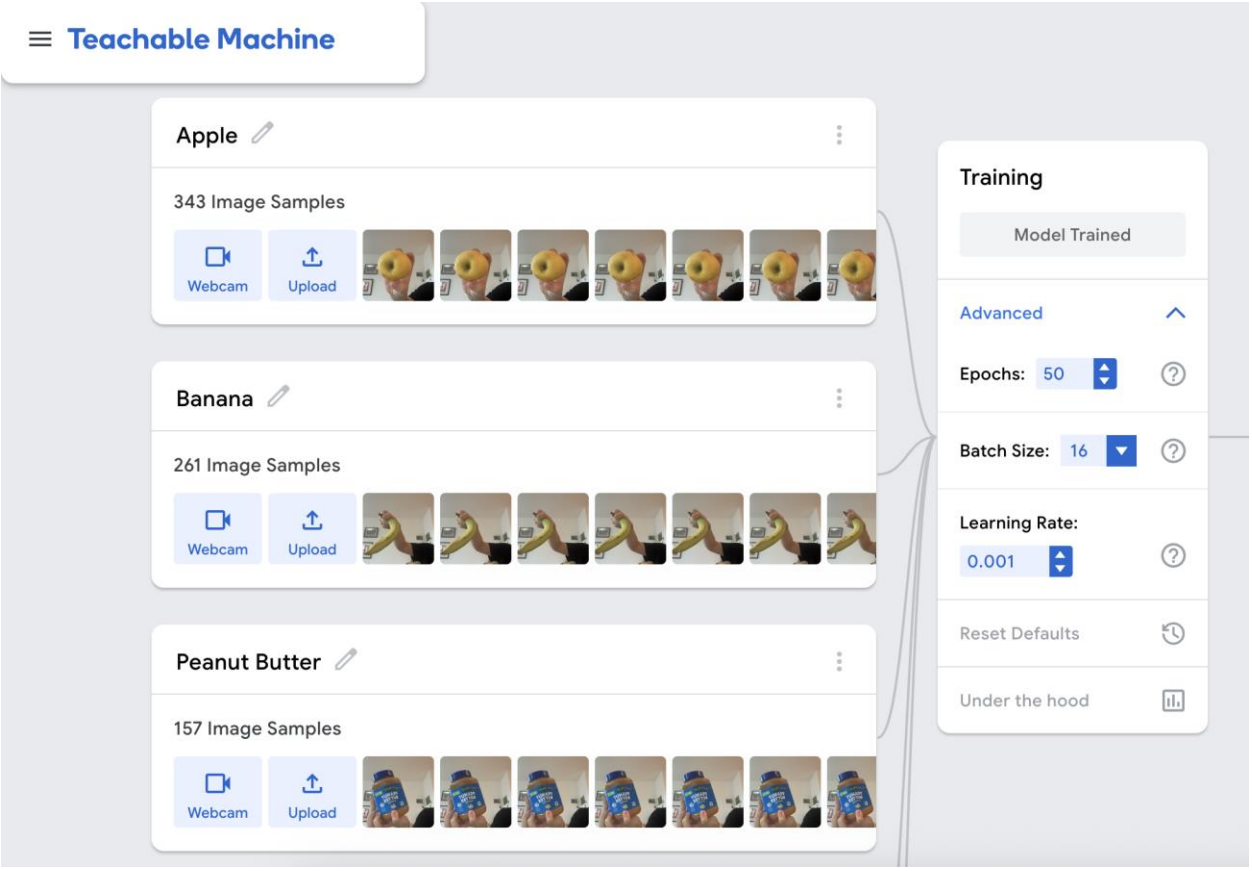
In the future, I would like to improve the final recommendation step by making it less random, for example by suggesting the most commonly bought item within a category or filtering out items that already appear in the basket. I also like the idea of an item-based approach, where recommendations are made based on which products are most often bought together. This approach feels more intuitive and could lead to more specific suggestions, although it does not reuse the task 1 model directly.

One major limitation of this model is that the original item classification model was trained on 12 grocery categories. However, when working with the Kaggle Groceries basket dataset, I found that only 8 of these categories actually appeared in the shopping baskets. As a result, the recommendation model operates on this smaller set of categories. Categories that appeared very rarely or not at all in the basket data, such as Household and Pet Supplies, were effectively excluded from the recommender. If I continue working on this, I would definitely prioritise expanding both datasets (task 1 and task 2).

Task 3

Data & Prep

For this task, I used Teachable Machine model. I had 5 items (apple, banana, peanut butter, salt and sauce) in 2 categories (fruits & vegetables, pantry items). I liked this because it was very easy to get sample images.



I installed some extra libraries such as Pillow for image loading. While testing, I noticed how some image formats don't work, such as heic from iPhone photos taken with the camera. Later, I decided to also integrate webcam input, and used OpenCV for camera access.

During this stage, I encountered several technical issues related to model compatibility and conda environment. Initially, I attempted to directly load both the Teachable Machine image model and the task 1 text classification model within the same Python environment. However, this caused repeated errors due to TensorFlow and Keras version mismatches. I tried resolving this by creating a new environment and adjusting the TensorFlow version,

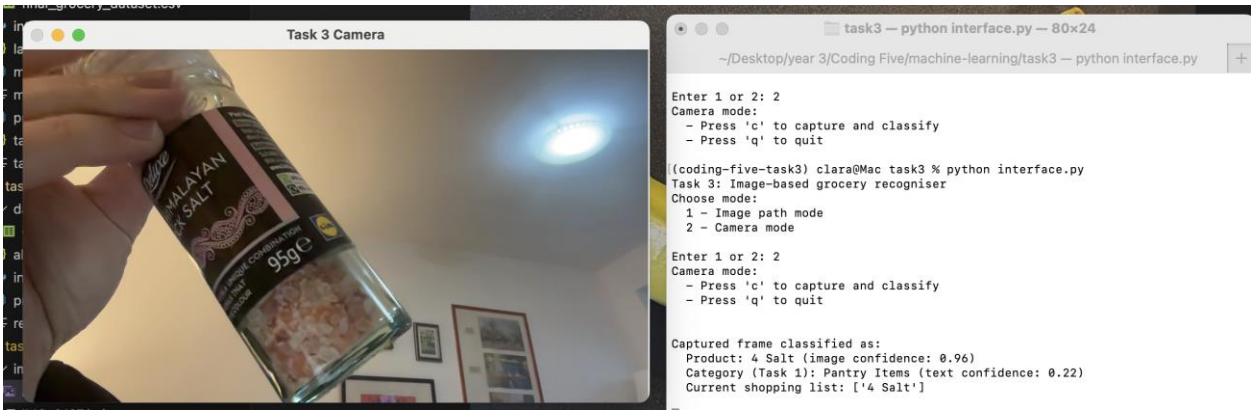
but this introduced further issues where one model would load successfully while the other would fail. To solve this, I retrained the Task 1 text classification model within the same environment used for Task 3, ensuring full compatibility between the image model and the text model. This approach allowed both models to load and run correctly together.

Installing OpenCV automatically upgraded NumPy to an incompatible version, causing conflicts with TensorFlow. This was fixed by downgrading NumPy to a compatible version and installing a specific OpenCV release that works with TensorFlow.

Model

The model itself was very fun to experiment with. I liked that I could change settings easily and get new training images fast.

The only problem was that I used a golden apple for the apple samples and it sometimes confused it with bananas. Similarly, the peanut butter and salt would have different packaging depending on the brand, so my model would not be very accurate.
With the photos I took of the products that I trained on, the model worked well.



Final thoughts & Future Improvement

This model lacks accuracy because of the way task 1 model was trained. Since part of this relies on the accuracy of model 1, there wasn't much I could do here to improve the accuracy other than getting more sample pictures. For example, the teachable machine model would recognise the banana, but model 1 would classify it as beverages.

```
(coding-five-task3) clara@Mac task3 % python interface.py
Task 3: Image-based grocery recogniser
Choose mode:
  1 - Image path mode
  2 - Camera mode

Enter 1 or 2: 1

Image path mode (type 'q' to quit).

Enter image path: /Users/clara/Desktop/year 3/Coding Five/machine-learn
ing/task3/images/banana.jpg
Product: 1 Banana (1.00)
Category: Beverages (0.15)


Enter image path:
```

The next step for this task would be to go to a grocery store and collect more images of different varieties of each item so I can improve accuracy.