README file for Criteo to run the 2-3 tier approach on their end

**Step 1:**1\_Pipeline\_to\_preprocess\_catalog\_data\_for\_embedding\_model.ipynb

Summary: this notebook performs the necessary preprocessing on the catalog data so it will be ready as input for the embedding model.

* Open the notebook 1\_Pipeline\_to\_preprocess\_catalog\_data\_for\_embedding\_model\_(need\_to\_run\_one\_time).ipynb
* In the second chunk, there are a bunch of variables you **have to change**



| Variable name | Explanation |
| --- | --- |
| catalog\_metadata\_file\_path | This is the path to the products catalog csv/xslx file. This file has, for each product the category, subcategory, description, etc. |
| catalog\_images\_location\_folder | This is the path to the directory where all the images are stored as jpg/jpeg files. |
| path\_to\_save\_the\_preprocessed\_df | This is the path where we want to store the pre-processed catalog data (we change column names, and remove products without any image file path). We will read this preprocessed file in notebook 2 |
| loction\_to\_save\_embeddings | This is the path where we want to store the embedding of the descriptions of the products. |
| location\_to\_save\_encoder | The model cannot process non numerical data. We map each category, sub-category and brand into a unique integer. In order to be able to reverse this mapping, we are sacing the encoder in this path so we can later re-load it and use it to decode the numerical integer into the original text. |
| category\_column\_name  subcategory\_column\_name  sub\_subcategory\_column\_name  brand\_column\_name  product\_description\_column\_name  image\_path\_column\_name | **Column names we need to change!**  Please change the given names to the names of the corresponding columns in your catalog data set (which we load in `catalog\_metadata\_file\_path`) |
| remove\_products\_without\_description = True | This is a cleaning data option: if there are products without product description in the catalog we need to choose what to do with them:  If you pass `True` here then we remove all the products without description fromthe preprocessed df  If you pass `False` here then we replace the NA with empty string. |

* After changing the variable names in the second chunk you can run the whole notebook

Some assumptions:

* We assume you have a column with “**image\_path**” which leads to the location in your machine / cloud that stores the image as png/jpeg. It should NOT be an exterior link to the image (which need to be download). If this is the case, we might need to change the code in notebook2/3 to be able to access that external-source image.
* Your data must have the following columns: Category, sub-category, brand, image path, product description

**Step 2:** 2\_Embedding\_model\_pipeline.ipynb

Summary: In this notebook we define the first tier - the embedding model and train it!

We load the pre-processed catalog data and descriptions embeddings.

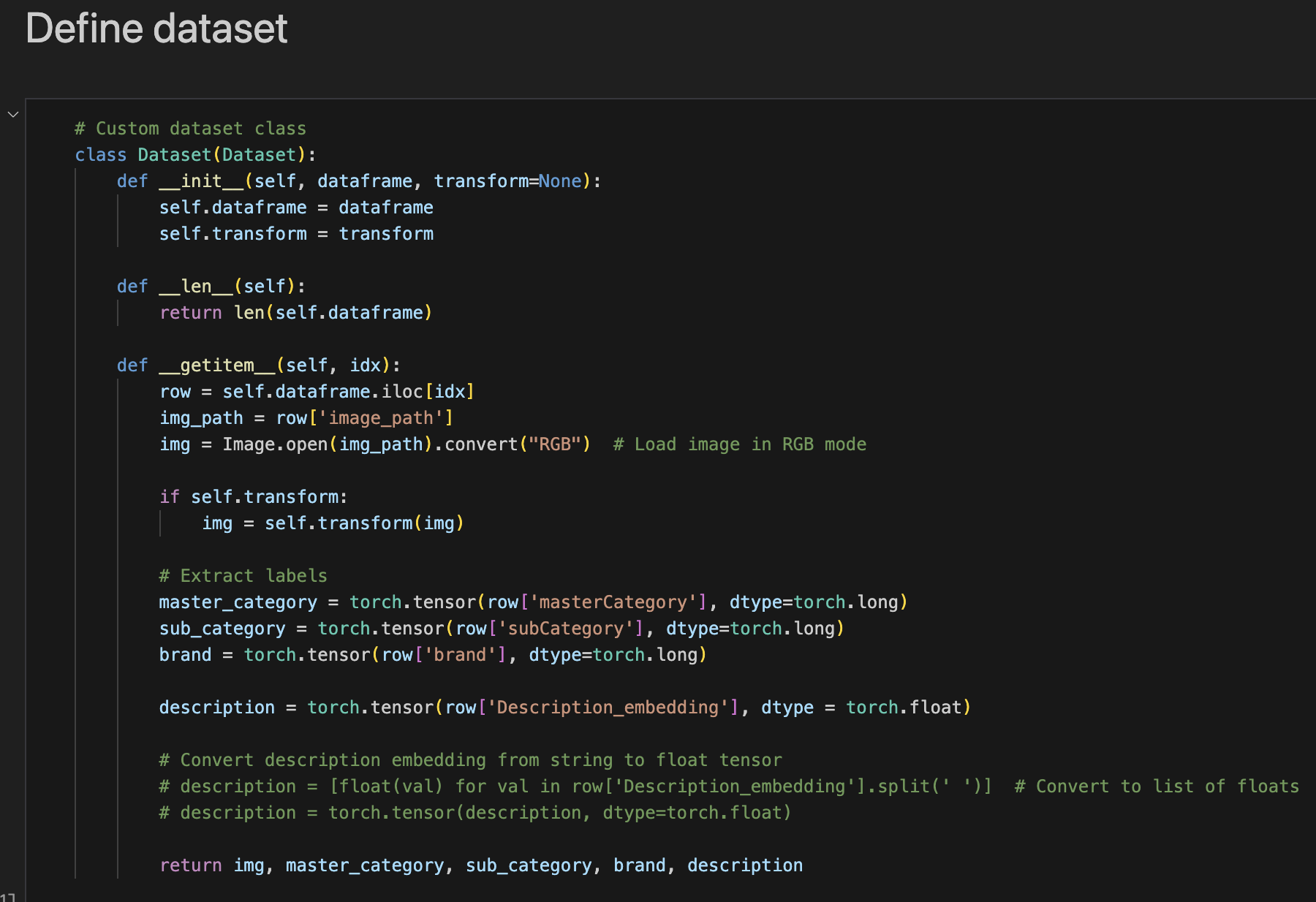
* **Open** the notebook 2\_Embedding\_model\_pipeline.ipynb
* In the second chunk, there are a bunch of variables you **have to change.**

| Variable name | Explanation |
| --- | --- |
| catalog\_images\_location\_folder | This is the path to the directory where all the images are stored as jpg/jpeg files. |
| path\_to\_load\_the\_preprocessed\_df | This is the path where we stored the pre-processed catalog data (we change column names, and remove products without any image file path). This is one of the outputs of notebook 1. |
| loction\_to\_load\_desc\_embeddings | This is the path where we stored the embedding of the descriptions of the products (in notebook 1). |
| location\_to\_save\_encoder | The model cannot process non numerical data. We map each category, sub-category and brand into a unique integer. In order to be able to reverse this mapping, we are sacing the encoder in this path so we can later re-load it and use it to decode the numerical integer into the original text. |
| log\_file | This is a log file we create and update while running the training chunk of the embedding model. Please provide the path where you want to save it. |
| folder\_to\_save\_embed\_model | This is the path to save the embedding model. |
| location\_to\_save\_embedding\_model\_performance\_in\_bucket | This is the path to save the performance metrics of the embedding model. |
| folder\_to\_save\_products\_embeddings | This is the path to the folder where you want to save the embeddings that the embedding model will create on the train and the test set.  This is important and should be the input path to the NN |

‘

* Defining the dataset:

In the following part we define the dataset, this is were we access the image from the image path. You shouldn’t need to change anything, but if needed this is where you should do it:



* Moreover, if we would like to incorporate other variables to the embedding model, like sub-sub category, price, etc., we will need to alter the dataset code, so we will be able to extract these features from the dataset.
* Note: we remove from the dataset brands that appear only once, i.e. if there is a brand with a single image we remove it.
* Look up in the notebook for `Model Definition for Embedding Generation`

In this part, we define the embedding model. Some configurations you can try changing:

* **Required\_dim\_embed\_description**: this is the description embedding size
* **dim\_embed\_image**: this is the image embedding size
* **self.backbone = models.resnet18(pretrained=True):** this is inside the models’ definition. Because of computing and storage limitations, we couldn’t use a bigger backbone.

You can try using:

* self.backbone = models.resnet34(pretrained=True)
* self.backbone = models.resnet50(pretrained=True)

self.backbone = models.resnet101(pretrained=True)

self.backbone = models.resnet152(pretrained=True)

(the output of the backbone will be now of size 2048)

self.image\_embedding = nn.Linear(2048, dim\_embed\_image)

* Look up in the notebook for `Define Loss Function and Optimizer`

In this part, we define the loss function and the optimizer.

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=1e-4)

You can try using a different optimizer, or a different learning rate.

* Look up in the notebook for `Training Loop for the embedding model`

In this part, we define the training loop for the embedding model. And right after that we run the training loop. Some configurations you can try changing:

* Epocs = 1: we ran for 1 epoch because of computing and time limitations, you can try running for more epochs
* save\_logs\_after\_k\_batches = 200: we are saving in the log file an update regarding the training loss every k=200 batches. You can choose a lower/higher number as you wish.

When running the training loop, we update the log file, so you can keep track on the learning, make sure the loss is decreasing and not increasing. If it increases, try to play with the learning rate.

* Important!! We need to save the embeddings of the train and test set!

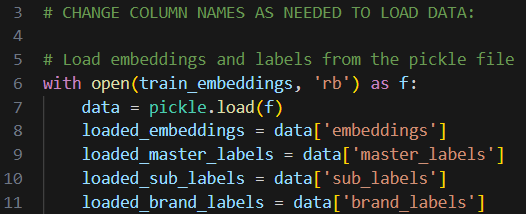
The whole point of the embedding model is to create the embedding for the products. We need to run the `**save\_embeddings**` function on the train loader and on the test loader separately.

**\***

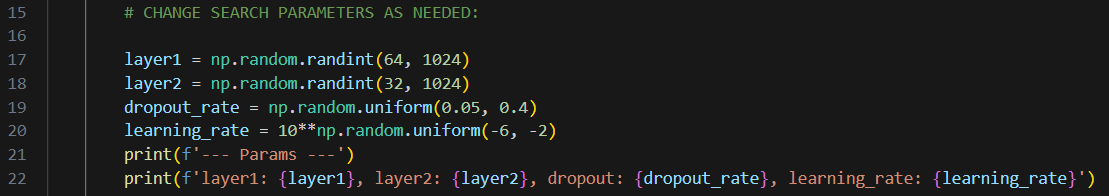
**Step 3:** 3\_Embedding\_classification\_pipeline\_2\_tier OR 3\_Embedding\_classification\_pipeline\_2-3\_tier

In 3\_Embedding\_classification\_pipeline\_2\_tier we define the second tier - the NN model and train it!

First, **load** the embedding (output of the 1st tier embedding model). **This may require changing the field names** depending on the column names



Then we run a **fine-tuning** pipeline to find optimal parameters with random search. This will print out the best parameters, model, and accuracy. Please **feel free to change the search parameters** in the code:



Then we can initialize and **train** the best model! The neural network has **2 hidden layers** (1 input, 1 output, 2 hidden layers for 4 total). Feel free to change the number of hidden layers! Here are some optional variables:

| Variable name | Explanation |
| --- | --- |
| k (optional) | The frequency the model prints out loss and accuracy. Defaults to 10.  Ex: if k=2, it will print out loss and accuracy every 2 epochs. |
| save\_directory (optional) | The directory for saving model weights. Will create directory if it doesn’t exist. Will also save training logs if log\_file variable is specified.  If not specified, weights and log file will not be saved. |
| log\_file (optional) | The name of the log file for saving training logs.  If not specified, log file will not be saved. |

Finally we can **evaluate** the model by getting its 95-99% thresholds. Here are more variables:

| Variable name | Explanation |
| --- | --- |
| metrics\_title (required) | The header title that is shown when printing metrics. |
| file\_path (optional) | The path and file name to save model evaluation metrics. Must end with .xlsx  If not specified, evaluation will not be saved. |
| prints\_metrics (optional) | Flag specifying whether or not to print metrics to the console. Defaults to False. |

3\_Embedding\_classification\_pipeline\_2-3\_tier is mostly the same, except we have an intermediate neural network for classifying category.