Training a Model:

Requires data.

Here are some available resources where to get Datasets:

* Your own proprietary data
* Kaggle
* HuggingFace datasets
* Synthetic data (fictitious data generated by a model)\*\* recommended in some cases
* Specific companies do it -> scale.com

In the course we saw this approach when dealing with data (part of the LLM engineer):

Data stages

1. Investigate: understanding how well populated, quality
2. Parse: parsing in a format to be easy to work
3. Visualize: ranges how they are distributed, using graph or visual tools,
4. Assess Data Quality
5. Curate how to craft the dataset, filtering out, finding imbalance
6. Save

Here is a use case: Train a model able to predict the price of an item based in their description.

We use dataset from HuggingFace related to amazon products reviews:

<https://huggingface.co/datasets/McAuley-Lab/Amazon-Reviews-2023>

the folder with all the product datasets is here:  
<https://huggingface.co/datasets/McAuley-Lab/Amazon-Reviews-2023/tree/main/raw/meta_categories>

1. **Investigate:**

We started by retrieving the data set:

```python

from huggingface\_hub import login

from datasets import load\_dataset, Dataset, DatasetDict

# Log in to HuggingFace

hf\_token = os.environ['HF\_TOKEN']

login(hf\_token, add\_to\_git\_credential=True)

# Load in our dataset

dataset = load\_dataset("McAuley-Lab/Amazon-Reviews-2023", f"raw\_meta\_Appliances", split="full", trust\_remote\_code=True)

#checking how many items there are

print(f"Number of Appliances: {len(dataset):,}")

``

Output: Number of Appliances: 94,327

1. **Parsing the data: use the data structure to work with**

Investigate a particular datapoint

```python

datapoint = dataset[0]

print(datapoint["title"])

print(datapoint["description"])

print(datapoint["features"])

print(datapoint["details"])

print(datapoint["price"])

```

We noticed these product does not have \*Price\* value. Then that is an issue for our solution as we are going to use the price for training our model. We will need to dig into the data and filter out the items without price.

How many have prices?

```python

prices = 0

for datapoint in dataset:

try:

price = float(datapoint["price"])

if price > 0:

prices += 1

except ValueError as e:

pass

#note you can use {prices**:,**} to have the “,” separator in 1,000

print(f"There are {prices:,} with prices which is {prices/len(dataset)\*100:,.1f}%")

```

Output: There are 46,726 with prices which is 49.5%

What is the length of each item description?

```python

# For those with prices, gather the price and the item content length

prices = []

lengths = []

for datapoint in dataset:

try:

price = float(datapoint["price"])

if price > 0:

prices.append(price)

contents = datapoint["title"] + str(datapoint["description"]) + str(datapoint["features"]) + str(datapoint["details"])

lengths.append(len(contents))

except ValueError as e:

pass

```

1. **Visualize the data:**

We are going to use Matplotlib for visualization

```python

import matplotlib.pyplot as plt

%matplotlib inline

# Plot the distribution of items content lengths

plt.figure(figsize=(15, 6))

plt.title(f"Lengths: Avg {sum(lengths)/len(lengths):,.0f} and highest {max(lengths):,}\n")

plt.xlabel('Length (chars)')

plt.ylabel('Count')

plt.hist(lengths, rwidth=0.7, color="lightblue", bins=range(0, 6000, 100))

plt.show()

```

Check for those with price what the length of the description

Some has large description

A graph of a number of people

AI-generated content may be incorrect.

Now let’s check the price ranges

```python

# Plot the distribution of prices

plt.figure(figsize=(15, 6))

plt.title(f"Prices: Avg {sum(prices)/len(prices):,.2f} and highest {max(prices):,}\n")

plt.xlabel('Price ($)')

plt.ylabel('Count')

plt.hist(prices, rwidth=0.7, color="orange", bins=range(0, 1000, 10))

plt.show()

```

Some items have very high price (can impact the results)

A white rectangular object with black lines

AI-generated content may be incorrect.

Note: when looking to train model there might be a Cost factor or Memory Factor.

The # of tokes we pass to an Open-Source is a constrain, the more token the more memory we need and harder will be to train. If working with Closed-Source the constrain will be the cost $$

In this scenario we are working with Closed-Source models. Therefore, we need to be mindful of this parameter. Here we are going to limit the numbers of tokens use to train the model, so we need to curate each item to fit into a certain # of tokens.

4**. Curate data:**

This task will be repetitive for each of the items of the dataset. We create an **Item class** with certain methods to handle this.

Check code example here []

Time to curate our dataset

We select items that cost between **1 and 999 USD**

We will be creating Item instances, which truncate the text to fit within **180 tokens using the right Tokenizer** And will create a prompt to be used during Training.

A close-up of a text

AI-generated content may be incorrect.

We used Meta-Llama-3.1-8B for the tokenizer to check the # of tokens we are passing in

Items will be rejected if they don't have sufficient characters.

**But why 180 tokens??**

How did we determine that number?

The answer: this is an example of a **"hyper-parameter".** In other words, it's **basically trial and error**! We want a sufficiently large number of tokens so that we have enough useful information to gauge the price. **But** we also want to keep the **number low** so that we can **train efficiently**. You'll see this in action in **Week 7.**

I started with a number that **seemed reasonable and experimented with a few variations before settling on 180**. If you have time, you should do the same! You might find that you can beat my results by finding a better balance. This kind of trial-and-error might sound **a bit unsatisfactory**, but it's a **crucial part of the data science R&D process**.

There's another interesting reason why we might favor a lower number of tokens in the training data. When we eventually get to use our model at inference time, we'll want to provide new products and have it estimate a price. And we'll be using short descriptions of products - like 1-2 sentences. For best performance, we should size our training data to be similar to the inputs we will provide at inference time.

**But I see in items.py it constrains inputs to 160 tokens?**

The description of the products is limited to **160 tokens** **because we add some more text before and after the description to turn it into a prompt. That brings it to around 180 tokens in total.**

Note: I noticed

<details><summary><strong> a potential issue in the Item code </strong></summary>

Let's break down why and how it can impact the model:

1. Chopped Sentences and Words:

• As you pointed out, the code uses contents = contents[:CEILING\_CHARS] to truncate the product content if it exceeds CEILING\_CHARS. This character-based truncation can indeed lead to sentences or words being cut off mid-way.

• Then, the code tokenize the chopped content.

• Then a further limitation is applied based on the token count, tokens = tokens[:MAX\_TOKENS].

• Finally, the code decodes the truncated tokens back into text using text = self.tokenizer.decode(tokens).

2. Misleading Content for the Model:

• Yes, this process can definitely result in misleading content. If a sentence is abruptly cut off, the remaining portion might not convey the intended meaning or could even suggest a different context.

• For example, a sentence like "This product is highly durable and..." could be truncated to "This product is highly dur..." The model would then be trained on this incomplete and potentially misleading information.

• Also, the tokenization process can cause that the meaning of a word is lost, if the word is chopped. For example, the word "application" could be chopped to "applicat" and then tokenized, losing the original meaning.

3. Impact on Training:

• Reduced Accuracy: Training on misleading or incomplete data can negatively impact the model's accuracy. The model might learn incorrect patterns or fail to capture the true relationships between product descriptions and prices.

• Bias: If the truncation consistently affects certain types of products or descriptions, it could introduce bias into the model.

• Instability: The model's training process might become unstable, leading to fluctuations in performance or difficulty in achieving convergence.

• Reduced performance: The model will be less accurate in predicting the prices.

In summary:

You've correctly identified a potential flaw in the code's text processing. The character-based and token based truncation can lead to misleading content, which can negatively impact the model's training and performance.

Possible solutions:

• Sentence-Based Truncation: Instead of truncating based on characters, the code could be modified to truncate at sentence boundaries. This would ensure that the model receives complete sentences.

• Word-Based Truncation before tokenization: Before tokenization, the code could truncate at word boundaries.

• Summarization: If the product descriptions are too long, the code could use a summarization technique to generate a concise and accurate summary.

• Increase MAX\_TOKEN and CEILING\_CHARS: if the model allow it, increasing the maximum number of tokens and characters, could reduce the risk of chopped words and sentences.

Note From Ed:

*“Hey Luis - if I remember right - my intention with CEILING\_CHARS was to initially discard text that was extremely unlikely to make the cut. I did it to improve the performance of that text processing, so that functions like scrub\_details don't need to process lots of text that's going to get discarded anyway. So if I understand right, the downside of your change is that it wouldn't have the same performance improvements.*

*I intended to pick a CEILING\_CHARS that was well beyond the number of tokens - some large multiple of MAX\_TOKENS. I ran a few tests to make sure that I wasn't discarding anything that could make the cut.*

*Having said that, I realize that later I added more and more string parsing. If you're seeing examples of text making the cut that's been truncated, then that would be a problem! The simple fix would be to make CEILING\_CHARS much higher - triple it - it's only there to improve the performance of the string processing; not to truncate the final result.*

*It's also worth mentioning a subtle point: it's actually probably not a problem if a word like APPLICAT gets tokenized, because generally models are smart enough to interpret even word fragments based on context.*

*But regardless - if CEILING\_CHARS is truncating actual content that would fit into MAX\_TOKENS then that's a bug for sure.. let me know!”*

</details>

Now let’s create an item object for each data point with a price

```python

# Create an Item object for each with a price

items = []

for datapoint in dataset:

try:

price = float(datapoint["price"])

if price > 0:

item = Item(datapoint, price) # creating an Item

if item.include:

items.append(item)

except ValueError as e:

pass

print(f"There are {len(items):,} items")

```

Output:

There are 29,191 items

```python

# Look at the first item

items[0]

```

Output:

<WD12X10327 Rack Roller and stud assembly Kit (4 Pack) by AMI PARTS Replaces AP4980629 PS3486910 1811003 = $8.99>

```python

# Investigate the prompt that will be used during training - the model learns to complete this

print(items[1].prompt)

```

Output:

How much does this cost to the nearest dollar?

Door Pivot Block - Compatible Kenmore KitchenAid Maytag Whirlpool Refrigerator - Replaces - Quick DIY Repair Solution

Pivot Block For Vernicle Mullion Strip On Door - A high-quality exact equivalent for part numbers and Compatibility with major brands - Door Guide is compatible with Whirlpool, Amana, Dacor, Gaggenau, Hardwick, Jenn-Air, Kenmore, KitchenAid, and Maytag. Quick DIY repair - Refrigerator Door Guide Pivot Block Replacement will help if your appliance door doesn't open or close. Wear work gloves to protect your hands during the repair process. Attentive support - If you are uncertain about whether the block fits your refrigerator, we will help. We generally put forth a valiant effort to guarantee you are totally

Price is $17.00

```pyhton

# Investigate the prompt that will be used during testing - the model has to complete this

print(items[0].test\_prompt())

```

Output:

How much does this cost to the nearest dollar?

Rack Roller and stud assembly Kit (4 Pack) by AMI PARTS Replaces

PARTS NUMBER The dishwasher top rack wheels and stud assembly Kit （4 pcs） SCOPE OF APPLICATION The dishwasher works with most top name brands,If you are not sure if part is correct, ask us in Customer questions & answers section or visiting the AMI PARTS storefront.We’re happy to help ensure you select the correct part for your Rack Roller and stud REPLACES PART FIXES SYMPTOMS Door won’t close | Not cleaning dishes properly | Noisy | Door latch failure QUALITY WARRANTY The replacement part is made from durable high quality material and well-tested by manufacturer.For any reason you’re not satisfied,you can ask for a replacement or full refund Brand Name AMI PARTS, Model

Price is $

Let’s visualize the token distribution after filtering:

```python

# Plot the distribution of token counts

tokens = [item.token\_count for item in items]

plt.figure(figsize=(15, 6))

plt.title(f"Token counts: Avg {sum(tokens)/len(tokens):,.1f} and highest {max(tokens):,}\n")

plt.xlabel('Length (tokens)')

plt.ylabel('Count')

plt.hist(tokens, rwidth=0.7, color="green", bins=range(0, 300, 10))

plt.show()

```

A green bar graph with numbers

AI-generated content may be incorrect.

Let’s have a look into the new Price distribution:

```python

# Plot the distribution of prices

prices = [item.price for item in items]

plt.figure(figsize=(15, 6))

plt.title(f"Prices: Avg {sum(prices)/len(prices):,.1f} and highest {max(prices):,}\n")

plt.xlabel('Price ($)')

plt.ylabel('Count')

plt.hist(prices, rwidth=0.7, color="purple", bins=range(0, 1000, 10))

plt.show()

```

A graph with numbers and lines

AI-generated content may be incorrect.

notice now the highest price is $10,960

## Sidenote

If you like the variety of colors that matplotlib can use in its charts, you should bookmark this:

<https://matplotlib.org/stable/gallery/color/named_colors.html>

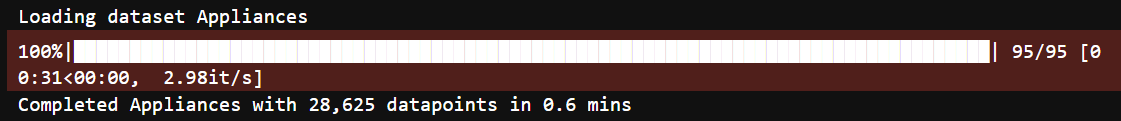
```python

# Load one dataset at a time -> In this case only the Appliances

items = ItemLoader("Appliances").load()

```

Output:



## Now to SCALE UP

Let's look at all datasets of all the items that you might find in a large home retail store - electrical, electronic, office and related, but not clothes / beauty / books.

```python

dataset\_names = [

"Automotive",

"Electronics",

"Office\_Products",

"Tools\_and\_Home\_Improvement",

"Cell\_Phones\_and\_Accessories",

"Toys\_and\_Games",

"Appliances",

"Musical\_Instruments",

]

```

# Now, time for a coffee break!!

# By the way, I put the biggest datasets first.. it gets faster.

```python

items = []

for dataset\_name in dataset\_names:

loader = ItemLoader(dataset\_name)

items.extend(loader.load(workers=10))

```

```pyhton

print(f"A grand total of {len(items):,} items")

```

Output: A grand total of **2,811,408** items

NOTEs: Recommendation of HW setting for handling heavy task like this.

In most cases, for an LLM engineer, a machine with **50GB drive space** and **16GB memory** is sufficient; you can survive on less, and more is nice but a luxury. Personally I'm an advocate for using cloud servers for GPUs. Some people like to invest in hefty GPU boxes for local training, and I get that -- it's satisfying to run everything locally. But you could end up dropping **$4,000** for **a high end GPU box**, and in 12 months time it might be obsolete. I personally prefer to run on the cloud, and for **under $20 a month** I get access to a lot of power.

# Plot the distribution of prices

```python

prices = [item.price for item in items]

plt.figure(figsize=(15, 6))

plt.title(f"Prices: Avg {sum(prices)/len(prices):,.1f} and highest {max(prices):,}\n")

plt.xlabel('Price ($)')

plt.ylabel('Count')

plt.hist(prices, rwidth=0.7, color="blueviolet", bins=range(0, 1000, 10))

plt.show()

```A graph with numbers and a line

AI-generated content may be incorrect.

Let’s count items per categories

Note: python method Counter() is super useful to count things and classify them.

```pyhton

category\_counts = Counter() #a special dictionary designed for counting things. It starts empty.

for item in items:

category\_counts[item.category]+=1

categories = category\_counts.keys() #This creates a list of all the categories that were found (the keys in our counter dictionary).

counts = [category\_counts[category] for category in categories] #This creates a list of the counts for each category

# Bar chart by category

plt.figure(figsize=(15, 6))

plt.bar(categories, counts, color="goldenrod")

plt.title('How many in each category')

plt.xlabel('Categories')

plt.ylabel('Count')

plt.xticks(rotation=30, ha='right')

# Add value labels on top of each bar

for i, v in enumerate(counts):

plt.text(i, v, f"{v:,}", ha='center', va='bottom')

# Display the chart

plt.show()

```

A graph of a bar chart

AI-generated content may be incorrect.

# Objective

When we analyze the training data, we are looking for a balanced dataset. If there is a subset that prevails, we risk our Model to be too specific data driven in that group and have lower performance in other subsets. This could lead into “Memory Issues” where model start “forgetting” and performance decrease in task related to other categories.

A balanced dataset ensures that the model generalizes well across all categories. If one subset of data is significantly more frequent, the model **may overfit** to it, learning patterns that do not apply to less common subsets. This imbalance can cause the model to struggle with rare cases, leading to poor performance in real-world applications. The term "**memory issues"** refers to a phenomenon where the model prioritizes frequently seen data and gradually performs worse on less frequent categories, resembling a form of **"catastrophic forgetting"** seen in machine learning.

In our scenario, Automotive is the most dominant categories. Let’s craft a dataset which is more balanced in terms of prices. Less heavily scewed to cheap items, with an average that's higher than **$60.** Try to balance out the categories - fewer Automotive items.

```python

# Create a dict with a key of each price from $1 to $999

#THIS IS WHY WE HAD A PRICE ROUNDED!!!

# And in the value, put a list of items with that price (to nearest round number)

slots = defaultdict(list) # is a special dictionary from Python's collections module that provides default values for keys that don't exist yet

for item in items:

slots[round(item.price)].append(item)

```

```python

# Create a dataset called "sample" which tries to more evenly take from the range of prices

# And gives more weight to items from categories other than Automotive

# Set random seed for reproducibility

#This ensures that the random selections will be the same each time the code runs

np.random.seed(42)

random.seed(42)

sample = []

for i in range(1, 1000):

#This gets all items from the defaultdict that have a rounded price of 'i'.

slot = slots[i]

# If the price is 240 or higher, all items at this price point are added to the sample

if i>=240:

sample.extend(slot)

#If there are 1200 or fewer items at this price point, all of them are added to the sample

elif len(slot) <= 1200:

sample.extend(slot)

#For price points with more than 1200 items, this creates weights for each item.

#Items in the "Automotive" category get a weight of 1, while all other categories get a weight of 5.

#This means non-Automotive items are 5 times more likely to be selected.

else:

weights = np.array([1 if item.category=='Automotive' else 5 for item in slot])

# weight/ total weight = normalizon weight [from 0 to 1] required for probability-based selection.

weights = weights / np.sum(weights)

#This randomly selects 1200 items from the range (0 to len(slot)-1)) based on the weights. "replace=False" means no duplicates are allowed.

selected\_indices = np.random.choice(

len(slot), #means "choose from integers between [0 and total # of items - 1]"

size=1200, #means "select 1200 of these integers"

replace=False, #means "don't select the same index twice"

p=weights #means "use these probabilities when making the selection"

)

#This creates a list of the selected items using the randomly chosen indices.

selected = [slot[i] for i in selected\_indices]

sample.extend(selected)

print(f"There are {len(sample):,} items in the sample")

```