Using Large Language Models as Recommendation Systems

A review of recent research and a custom implementation

Mohamad Aboufoul · Follow

Published in Towards Data Science · 8 min read · Apr 10, 2023

Photo by Luca Baggio on Unsplash

Large Language Models (LLMs) have taken the data science community and the news cycle by storm these past few months. Since the advent of the transformer architecture in 2017, we've seen exponential advancements in the complexity of natural language tasks that these models can tackle from classification, to intent & sentiment extraction, to generating text eerily similar to humans.

From an application standpoint, the possibilities seem endless when combining LLMs with various existing technologies, to cover their pitfalls (one of my favorite being the GPT + Wolfram Alpha combo to handle math and symbolic reasoning problems).

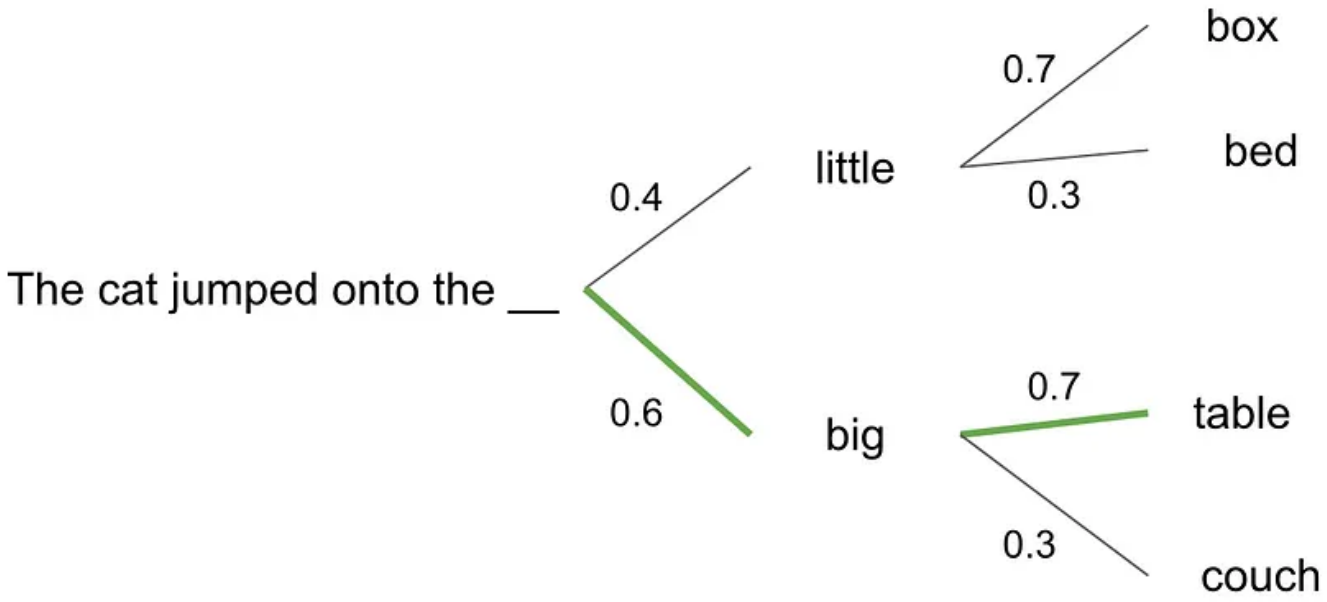
But what surprised me was that LLMs can also be used as recommendation systems, in and of themselves, without the need for any additional feature engineering or other manual processes that go into common rec systems. This capability is largely due to the nature of how LLMs are pre-trained and how they operate.

Table of Contents

- *Code*

A Recap of LLMs and How Transformers Work

Language models are probabilistic models that try to map the probability of a sequence of tokens (words in a phrase, sentence, etc.) occurring. They're trained on an array of texts, and derive probability distributions accordingly. For the various tasks that they can handle (summarization, question-answering, etc.) they are iteratively selecting the most probable token/word to continue a prompt, using conditional probability. See the example below:



An example of probabilities of subsequent tokens based on context (Image by author)

LLMs are language models that have been trained on MASSIVE amounts of

introduced in the famous 2017 paper "Attention Is All You Need" by Google. This architecture utilizes a "self-attention" mechanism, which allows the

model to learn how different tokens relate to one another during the pre-training process.

After pre-training across a large enough set of texts, similar words will have similar embeddings (ex: “King”, “Monarch”) and dissimilar words will have more different ones. Additionally, with these embeddings, we’ll see an algebraic mapping of words in relation to one another, allowing the model to more powerfully determine a correct next token for a sequence.

- Like the classic example of “King — Man + Woman = Queen” in Word2Vec

The added benefit of self-attention embeddings is that they will vary for a word depending on the words around it, making them more tailored to the meaning in that context.

Stanford’s Dr. Christopher Manning gives a great high-level overview of how LLMs work.

LLMs as Recommendation Systems

In 2022, researchers from Rutger’s University published the paper **“Recommendation as Language Processing (RLP): A Unified Pretrain, Personalized Prompt & Predict Paradigm (P5)”** (Geng et. al). In it they introduced a “flexible and unified text-to-text paradigm” which combined several recommendation tasks in a single system: P5. This system is capable of performing the following via natural language sequences:

- Sequential recommendation
- Rating prediction
- Explanation generation

- Review summarization
- Direct recommendation

Let's take a look at an example of the sequential recommendation task from the paper.

```
Input: "I find the purchase history list of user_15466:
4110 -> 4467 -> 4468 -> 4472
I wonder what is the next item to recommend to the user. Can you help
me decide?"
Output: "1581"
```

The researchers assigned the user and each item a unique ID. Using a training set with thousands of users (and their purchase histories) and unique items, the LLM is able to learn that certain items are similar to one another and that certain users have inclinations towards certain items (due to the nature of the self-attention mechanism). During the pre-training process across all these purchase sequences, the model essentially goes through a form of collaborative filtering. It sees what users have purchased the same items and what items tend to be purchased together. Combine that with an LLM's ability to produce contextual embeddings, and we suddenly have a very powerful recommendation system.

In the example above, although we don't know what item each ID corresponds to, we can infer that item "**1581**" was selected due to other users purchasing it along with any of the items that "**user_15466**" already purchased.

With regards to P5's architecture, it **"utilizes the pretrained T5 checkpoints as backbone"** (Geng et. al).

- T5 is another LLM that Google released a few years ago. It was designed to handle multiple types of sequence-to-sequence tasks, so it makes sense to use it as a starting point for this kind of system.

Implementing/Replicating P5 with Custom Data

I was really impressed with this paper and wanted to see if I could replicate the sequential recommendation capability on a smaller scale. I decided to leverage an open-source T5 model from Hugging Face (T5-large) and made my own custom dataset to fine-tune it to produce recommendations.

The dataset I made consisted of over 100 examples of sports equipment purchases along with the next item to be purchased. For example:

```
Input: "Soccer Goal Post, Soccer Ball, Soccer Cleats, Goalie Gloves"
Target Output: "Soccer Jersey"
```

Of course, to make this more robust, I decided to use a more specific prompt. Which looked like this:

> **Input:** *"ITEMS PURCHASED: {Soccer Goal Post, Soccer Ball, Soccer Cleats, Goalie Gloves} — CANDIDATES FOR RECOMMENDATION: {Soccer Jersey, Basketball Jersey, Football Jersey, Baseball Jersey, Tennis Shirt, Hockey Jersey, Basketball, Football, Baseball, Tennis Ball, Hockey Puck, Basketball Shoes, Football Cleats, Baseball Cleats, Tennis Shoes, Hockey Helmet, Basketball Arm Sleeve, Football Shoulder Pads, Baseball Cap, Tennis Racket, Hockey Skates, Basketball Hoop,*

> *Football Helmet, Baseball Bat, Hockey Stick, Soccer Cones, Basketball Shorts, Baseball Glove, Hockey Pads, Soccer Shin Guards, Soccer Shorts} — RECOMMENDATION: ”*
>
> **Target Output:** *“Soccer Jersey”*

Above you can see the specification of items purchased by the user so far, followed by a list of candidates for recommendation that haven’t been purchased yet (this is the entire inventory).

After fine-tuning the T5 model using Hugging Face’s Trainer API (*Seq2SeqTrainer* for ~10 epochs), I got some surprisingly good results! Some example evaluations:

> **Input:** *“ITEMS PURCHASED: {Soccer Jersey, Soccer Goal Post, Soccer Cleats, Goalie Gloves} — CANDIDATES FOR RECOMMENDATION: {Basketball Jersey, Football Jersey, Baseball Jersey, Tennis Shirt, Hockey Jersey, Soccer Ball, Basketball, Football, Baseball, Tennis Ball, Hockey Puck, Basketball Shoes, Football Cleats, Baseball Cleats, Tennis Shoes, Hockey Helmet, Basketball Arm Sleeve, Football Shoulder Pads, Baseball Cap, Tennis Racket, Hockey Skates, Basketball Hoop, Football Helmet, Baseball Bat, Hockey Stick, Soccer Cones, Basketball Shorts, Baseball Glove, Hockey Pads, Soccer Shin Guards, Soccer Shorts} — RECOMMENDATION: ”*
>
> **Model Output:** *“Soccer Ball”*

> **Input:** *"ITEMS PURCHASED: {Basketball Jersey, Basketball, Basketball Arm Sleeve} — CANDIDATES FOR RECOMMENDATION: {Soccer Jersey, Football Jersey, Baseball Jersey, Tennis Shirt, Hockey Jersey, Soccer Ball, Football, Baseball, Tennis Ball, Hockey Puck, Soccer Cleats, Basketball Shoes, Football Cleats, Baseball Cleats, Tennis Shoes, Hockey Helmet, Goalie Gloves, Football Shoulder Pads, Baseball Cap, Tennis Racket, Hockey Skates, Soccer Goal Post, Basketball Hoop, Football Helmet, Baseball Bat, Hockey Stick, Soccer Cones, Basketball Shorts, Baseball Glove, Hockey Pads, Soccer Shin Guards, Soccer Shorts} — RECOMMENDATION: "*
>
> **Model Output:** *"Basketball Shoes"*

This is of course subjective given that recommendations aren't necessarily binary successes/failures, but the outputs being similar to the respective purchases so far is impressive.

Replication Attempt 2 — Arabic

Next, I wanted to see if I could do this for Arabic, so I translated my dataset and looked for some publicly available T5 models that could handle Arabic text (AraT5, MT5, etc.). After trying out a dozen or so variants I found on the Hugging Face Hub, I unfortunately couldn't get it to produce acceptable results.

The model (after fine-tuning) would recommend the same 1 or 2 items, regardless of the purchase history — typically "كرة القدم", a "soccer ball" (***but hey, maybe it knows that Arabic speakers love soccer and are always looking for a new soccer ball***). Even after trying larger versions of these models, like

MT5-xl, I got the same result. This is likely due to the data paucity these LLMs have on languages other than English.

For my last attempt, I decided to try using the Google Translate API in conjunction with my English fine-tuned T5 model. The process was:

- Take the Arabic input → Translate to English → Feed into the English fine-tuned model → Get model prediction in English → Translate back to Arabic

Unfortunately, this still didn't help much since the translator would make some mistakes (ex: "كرة القدم", which we're using in place of "soccer" directly translates to "foot ball"), which threw the model off and resulted in the same 1–2 items being recommended consistently.

Benefits and Pitfalls of LLMs as Recommendation Systems

The most prominent benefits of this technique revolve around the ease of which it can be implemented as a stand-alone system. Because of the nature of LLMs and pre-training techniques discussed above, we can bypass the need for heavy, manual feature engineering — the model should be able to learn the representations and relationships naturally. Additionally, we can sidestep the cold start problem for new items to an extent — the name/description of the item can be extracted and naturally related to existing items that users have already purchased/selected.

There are, however, some pitfalls to this approach (don't throw away your current rec systems just yet!) which primarily entail a lack of control on what's recommended.

- Because there's no weighting for the different actions/events that a user takes to examine, then purchase an item, we're solely dependent on what the LLM predicts is the most probable next token(s) for recommendation. We can't take into consideration what the user bookmarked, looked at for a period of time, put into their cart, etc.
- Additionally, with these LLMs, we do run the risk that most of the recommendations are similarity based (i.e. items that are semantically similar to the items purchased so far), though I do think with extensive training data of user purchase history, we can ameliorate this issue via the "collaborative filtering" approach that this method could mimic.
- Finally, because LLMs can produce any text theoretically, the output could be a string that doesn't exactly match an item in the inventory (though I think the likelihood of this occurring is low).

Final Thoughts

Based on the results of the P5 paper and my attempt to replicate this on a T5 model with some fine-tuning and prompting, I would infer that this technique could be used across MANY language models. Using more powerful sequence-to-sequence models could help dramatically, especially if the fine-tuning data is large enough and the prompting technique is perfected.

However, I wouldn't go as far as to recommend (no pun intended) that anyone use this method in and of itself. I would suggest that it be used in conjunction with other recommendation system techniques, so that we can avoid the pitfalls mentioned above while simultaneously reaping the rewards. How this can be done — I'm not certain, but I think with some creativity, this LLM technique can be integrated in a helpful manner (maybe with extracting embedding-based features to use in collaborative filtering,

maybe in conjunction with the "Two Tower" architecture, the possibilities go on).

Code

- My implementation of T5 as a recommendation system (Github Repository)
- My fine-tuned T5 model on Hugging Face (Hugging Face Hub)

References

[1] S. Geng, S. Liu, Z. Fu, Y. Ge, Y. Zhang, Recommendation as Language Processing (RLP): A Unified Pretrain, Personalized Prompt & Predict Paradigm (P5) (2023), 16th ACM Conference on Recommender Systems

Recommendation System | Large Language Models | NLP | Arabic

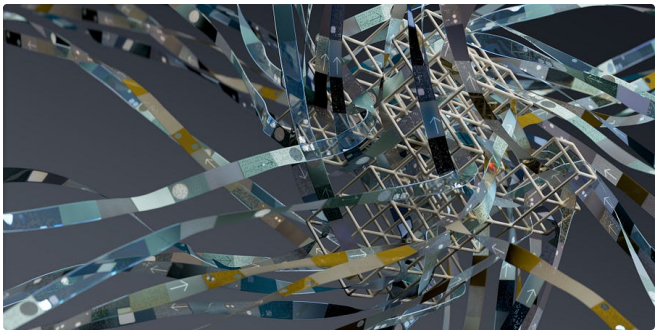
Written by Mohamad Aboufoul

122 Followers · Writer for Towards Data Science

Data scientist who's interested in NLP, Arabic, and building ML applications!

Follow

More from Mohamad Aboufoul and Towards Data Science



Mohamad Aboufoul in Towards Data Science

Despite Their Feats, Large Language Models Still Haven't...

A review of Chomsky's views on linguistics and LLMs

11 min read · Dec 5, 2022

418 4

Sheila Teo in Towards Data Science

How I Won Singapore's GPT-4 Prompt Engineering Competition

A deep dive into the strategies I learned for harnessing the power of Large Language...

· 24 min read · Dec 29, 2023

4.4K 47

Mariya Mansurova in Towards Data Science

Can LLMs Replace Data Analysts? Building An LLM-Powered Analyst

Part 1: empowering ChatGPT with tools

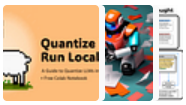
19 min read · Dec 11, 2023

Michael Berk in Towards Data Science

1.5 Years of Spark Knowledge in 8 Tips

My learnings from Databricks customer engagements

8 min read · Dec 24, 2023



Natural Language Processing
1072 stories · 549 saves

The New Chatbots: ChatGPT, Bard, and Beyond
12 stories · 264 saves

AI Regulation
6 stories · 265 saves

Apple's Vision Pro
7 stories · 40 saves

ZIRU

Day 5—Machine Learning System Design: a video recommendation...

Let's talk about the problem statement and metrics for building a video recommendatio...

· 4 min read · Aug 24, 2023

12

Nima Torabi

Deep Dive into Content-Based Recommender Systems: Unveilin...

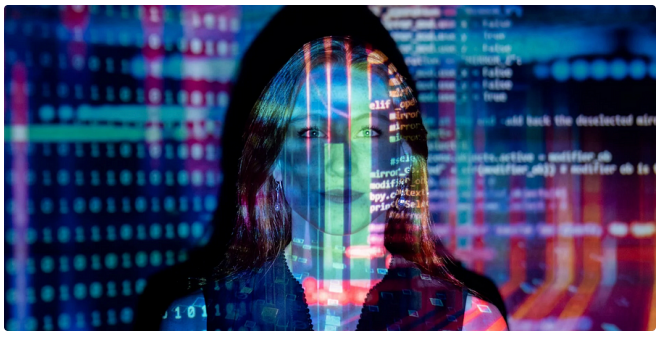
Exploring the Dynamics of Attribute-Based Recommendations in Content-Based...

44 min read · Aug 26, 2023

61

Dr. Robert Kübler in Towards Data Science

Introduction to Embedding-Based Recommender Systems



Python Fundamentals in Stackademic

Recommender Systems with Python Code Examples

Learn to build a simple matrix factorization recommender in TensorFlow

✦ · 13 min read · Jan 25, 2023

485 5

A Comprehensive Guide to Recommender Systems

✦ · 6 min read · Aug 28, 2023

73 1

See more recommendations