

Tackling Generated Datasets Diversity

In the previous [chapter](#), we discussed the potential of using LLM for synthetic dataset generation to further finetune a local Retriever model. This method is possible due to the availability of a large corpus of unlabeled documents. Each document is used to generate one or more synthetic queries and form a query-document pair.

But what if Information Retrieval is not your task? Let's say you are working on a legal document classification problem but are not permitted to send any data to an external API. In this situation, you would need to train a local model. However, collecting data could become a significant obstacle, causing delays in product development.

For simplicity, let's suppose the goal is to generate children's stories. This task was the starting point for research by [Eldan et al. \(2023\)](#). Each story consists of 2-3 paragraphs that follow a straightforward plot and theme, while the entire dataset covers a child's vocabulary and factual knowledge.

Language is not just a system of rules and symbols; it conveys and interprets meaning. The main challenge of using large language models to produce training data is ensuring dataset diversity. Even with high [generation temperature](#), models can produce repetitive datasets lacking the necessary diversity (even for a child's language). Coherence and relevance are other natural language generation challenges.

To tackle the diversity issue, the authors prepared a vocabulary of around 1500 basic words, mirroring a typical child's vocabulary, divided into nouns, verbs, and adjectives. In each generation, one verb, one noun, and one adjective were randomly selected. The model then generates a story integrating these random words.

This technique significantly broadened the dataset's diversity, ensuring stories cover a child's entire vocabulary and diverse concept combinations. Additionally, the authors integrated potential story features (like dialogue, plot twists, bad endings, or moral lessons). A random subset of these features was chosen for each story, and the model was prompted to include them.

Prompt:

Write a short story (3-5 paragraphs) which only uses very simple words that a 3 year old child would likely understand. The story should use the verb "{random.choice(verbs_list)}", the noun "{random.choice(nouns_list)}" and the adjective "{random.choice(adjectives_list)}". The story should have the following features: {random.choice(features_list)}, {random.choice(features_list)}. Remember to only use simple words!

Let's assume the gaps were filled as follows:

Prompt:

Write a short story (3-5 paragraphs) which only uses very simple words that a 3 year old child would likely understand. The story should use the verb "decorate", the noun "thunder" and the adjective "ancient". The story should have the following features: the story should contain at least one dialogue, the story has a bad ending. Remember to only use simple words!

Output:

Once upon a time, in an ancient house, there lived a girl named Lily. She loved to decorate her room with pretty things. One day, she found a big box in the attic. She opened it and saw many shiny decorations. Lily was very happy and decided to use them in her room. As Lily was decorating her room, the sky outside became dark. There was a loud thunder sound, and Lily got scared. She ran to her mom and said, "Mommy, the thunder is so loud!" Her mom hugged her and said, "Don't worry, it will pass soon." But the thunder did not stop. It got louder and louder, and the ancient house started to shake. Suddenly, the roof fell down on the pretty decorations. Lily was sad because her room was not pretty anymore. The end.

Here we rely on the latest text generation models (GPT-3.5 and GPT-4), which can produce vast amounts of synthetic content, following the instructions. Since we formulate the prompt randomly each time, and the model adheres to it precisely, the stories turn out to be incredibly varied. The narrative will be entirely different even if one or two words remain the same. In essence, we apply this technique to inject randomness into the prompt, generating a diverse dataset.

You get the idea:

1. Identify which parameters/entities might vary between different samples in your synthetic dataset;
2. Generate or manually compile a collection of these entities to fill in the gaps;
3. Produce the dataset by randomly selecting entities for insertion. It's best to set the generation temperature higher than the default but below the maximum;
4. Train a local model on the generation results of ChatGPT/GPT-4.

It is important to note that one of the entities used to seed your generation could be a class label. For example, in the sentiment classification task, you can directly mention "positive" or "negative" in the prompt to receive a generated text with the corresponding label. Then, a local classifier is trained on this data.

Iterative (Hierarchical) Synthetic Data Generation

The format can be made more complex by using even more entities, some of which are generated beforehand by the LLM itself. For example, one could start with a request to generate a story summary and pen down one sentence that must appear in the story. Later, just use that generated intermediate data in the final request.

Prompt:

```
Summary: {a short summary generated by LLM, using the approach above}  
Features: {copy the features from the initial prompt}  
Sentence: {a sentence generated by LLM, which should be present in the story}  
Words: {copy the words from the initial prompt}  
Story:
```

Example of a filled-in prompt:

Prompt:

```
Summary: Lily and Timmy build a sandcastle together and learn to compromise, but it gets  
knocked over by a gust of wind. They find beauty in the broken sandcastle and play happily  
with a butterfly.  
Features: Dialogue, Foreshadowing, Twist  
Sentence: One day, she went to the park and saw a beautiful butterfly.  
Words: disagree, network, beautiful  
Story:
```

Output:

```
Once upon a time, there was a little girl named Lily. She loved to play with her toys and her  
friends. One day, she went to the park and saw a beautiful butterfly. She tried to catch it,  
but it flew away. Lily's friend, Timmy, came to play with her. They decided to build a  
sandcastle, but they disagreed on how to build it. Lily wanted to make it tall, while Timmy  
wanted to make it wide. They argued for a while, but finally agreed to make it both tall and  
wide. As they were building the sandcastle, they noticed a group of ants working together.  
They realized that the ants were building a network of tunnels. Suddenly, a gust of wind came  
and knocked over their sandcastle. They were sad, but then they saw the butterfly again. It
```

landed on the remains of their sandcastle and they realized that it was still beautiful, even in its broken state. They smiled and played together happily.

Thus, it's possible to generate hundreds of thousands of very diverse examples to train the model on. Let's say you need to train a classifier that determines whether a text contains a dialogue or a plot twist. As the initial prompt contains labels, it's known which target value needs to be predicted for each generated sample.

Textbooks Are All You Need

A crucial question arising from this approach is whether the synthesis of a dataset can truly provide benefits when training networks for real-world applications. Fortunately, the authors addressed this question by conducting their investigation and validating the efficacy of training smaller language models using synthetic data derived from State-of-the-Art LLMs.

In their study, [Gunasekar et al. \(2023\)](#) emphasize the importance of high-quality training data in their model. They argue that language models would be more effective if they were trained on materials that resemble the characteristics of a well-regarded "textbook": clear, comprehensive, informative, and unbiased.

These principles formed the basis for creating a semi-synthetic dataset to train LLM called Phi-1. The main evaluation task is to generate a Python function that follows a given text description or docstring. The model's quality is evaluated using the HumanEval benchmark ([Chen et al., 2021](#)).

The authors highlight the importance of diversity in this approach for several reasons. It exposes the language model to a variety of coding expressions and problem-solving approaches, reduces the risk of overfitting or relying on specific patterns, and improves the model's ability to handle unfamiliar or innovative tasks.

To address the code-writing challenge, the authors created textbook-like documents that focus on topics promoting reasoning and basic algorithmic skills. They achieved diversity by imposing restrictions on:

- topics
- target audience

Unfortunately, the authors did not provide specific information about the prompt template used to generate the synthetic data. However, they did demonstrate the

resulting output. They chose to use ChatGPT (GPT-3.5) instead of GPT-4, and even this strategy yielded excellent results as they fine-tuned the model using synthetic data.

Image Source: [Gunasekar et al. \(2023\)](#).

It's essential to pay attention to the target audience constraint. Designating a target audience is an excellent way to enhance synthetic data diversity. A high school student, a CS undergraduate, and a PhD candidate certainly phrase their explanations and process information differently, making the generated data even more varied. Let's attempt to recreate the prompt.

Prompt:

Write an extract from a Computer Science textbook for a 1st-year bachelor. The coding language is Python 3.6.

This is an extract from the middle of the following topic: Singular matrices.

The extract starts with a high-level overview of the topic. Then, it presents an example and describes the solution in natural language. After that, it provides 1-2 code snippets, following the example. Each snippet has no more than 10 rows. There should be no text after code snippets.

Keep in mind that the extract should be concise and explain only one concept at a time. The code should represent a Python function & its call. All the common ML/DS libraries are available.

Don't be too verbose. The expected length of the extract is 2 paragraphs at most.

Output:

```
**Singular Matrices**
```

In linear algebra, a matrix is termed as "singular" if it doesn't have an inverse. Another equivalent definition is when the determinant of the matrix is zero. Singular matrices are crucial because they represent systems of linear equations that do not have a unique solution.

Consider a 2x2 matrix `[[a, b], [c, d]]`. If the product of ``ad`` and ``bc`` is the same (i.e., ``ad = bc``), then this matrix is singular. Let's explore this with an example: given the

```
matrix `[[2, 4], [1, 2]]`, you'll find that the product of the diagonals yields the same  
result ( $2 * 2 = 4 * 1$ ), thus it's a singular matrix.  
import numpy as np  
def is_singular(matrix):  
    determinant = np.linalg.det(matrix)  
    return abs(determinant) < 1e-9  
matrix_example = np.array([[2, 4], [1, 2]])  
print(is_singular(matrix_example)) # returns True
```

Quite close!

In total, the authors generated 1B tokens to augment the model's training set, allowing a smaller model (only 1.5B parameters) to rival models ten times its size (for details, refer to the article [Gunasekar et al. \(2023\)](#)).

Image Source: [Gunasekar et al. \(2023\)](#)

For your task, you probably don't need such a large amount of synthetic data (since the authors studied the pretraining, which requires significant resources). However, even as an estimate, at a price of `$0.002` per 1k tokens (standard ChatGPT pricing), it would cost `$2000` for the generated tokens and approximately the same amount for the prompts.

Keep in mind that fine-tuning on synthetic data becomes more valuable as the domain becomes more niche, especially if the language deviates from English (among other factors). Additionally, this method works well with [Chain-of-Thought \(CoT\)](#), helping the local model improve its reasoning capabilities. Other prompting techniques work, too.

And don't forget that open-source models like Alpaca ([Taori et al., \(2023\).](#)) and Vicuna ([Zheng et al., \(2023\).](#)) excel through fine-tuning on synthetic data.

Last updated on September 19, 2024