Article

Categorizing and organizing data with content tags

Identify topics, actions, objects, and emotions in input text with a content tagging model.

Overview

The Foundation Models framework provides an adapted on-device system language model that specializes in content tagging. A content tagging model produces a list of categorizing tags based on the input text you provide. When you prompt the content tagging model, it produces a tag that uses one to a few lowercase words. The model finds the similarity between the terms in your prompt so tags are semantically consistent. For example, the model produces the topic tag “greet” when it encounters words such as “hi,” “hello,” and “yo”. Use the content tagging model to:

- Gather statistics about popular topics and opinions in a social app.
- Customize your app’s experience by matching tags to a person’s interests.
- Help people organize their content for tasks such as email autolabeling using the tags your app detects.
- Identify trends by aggregating tags across your content.

If you’re tagging content that’s not an action, object, emotion, or topic, use `general` instead. Use the general model to generate content like hashtags for social media posts. If you adopt the tool calling API, and want to generate tags, use `general` and pass the `Tool` output to the content tagging model. For more information about tool-calling, see Expanding generation with tool calling.

Provide instructions to the model

The content tagging model isn't a typical language model that responds to a query from a person: instead, it evaluates and groups the input you provide. For example, if you ask the model questions, it produces tags about asking questions. Before you prompt the model, consider the instructions you want it to follow: instructions to the the model produce a more precise outcome than instructions in the prompt.

The model identifies topics, actions, objects, and emotions from the input text you provide, so include the type of tags you want in your instructions. It's also helpful to provide the number of tags you want the model to produce. You can also specify the number of elements in your instructions.

```
// Create an instance of the on-device language model's content tagging use case.
let model = SystemLanguageModel(useCase: .contentTagging)

// Initialize a session with the model and instructions.
let session = LanguageModelSession(model: model, instructions: """
    Provide the two tags that are most significant in the context of topics.
    """
)
```

You don't need to provide a lot of custom tagging instructions; the content tagging model respects the output format you want, even in the absence of instructions. If you create a generable data type that describes properties with `GenerationGuide`, you can save context window space by not including custom instructions. If you don't provide generation guides, the model generates topic-related tags by default.

> **Note**
>
> For very short input queries, topic and emotion tagging instructions provide the best results. Actions or object lists will be too specific, and may repeat the words in the query.

Create a generable type

The content tagging model supports `Generable`, so you can define a custom data type that the model uses when generating a response. Use `maximumCount( :)` on your generable type to enforce a maximum number of tags that you want the model to return. The code below uses `Generable` guide descriptions to specify the kinds and quantities of tags the model produces:

```
@Generable
struct ContentTaggingResult {
```

```
    @Guide(
        description: "Most important actions in the input text.",
        .maximumCount(2)
    )
    let actions: [String]

    @Guide(
        description: "Most important emotions in the input text.",
        .maximumCount(3)
    )
    let emotions: [String]

    @Guide(
        description: "Most important objects in the input text.",
        .maximumCount(5)
    )
    let objects: [String]

    @Guide(
        description: "Most important topics in the input text.",
        .maximumCount(2)
    )
    let topics: [String]
}
```

Ideally, match the maximum count you use in your instructions with what you define using the `maximumCount(_:)` generation guide. If you use a different maximum for each, consider putting the larger maximum in your instructions.

Long queries can produce a large number of actions and objects, so define a maximum count to limit the number of tags. This step helps the model focus on the most relevant parts of long queries, avoids duplicate actions and objects, and improves decoding time.

If you have a complex set of constraints on tagging that are more complicated than the maximum count support of the tagging model, use `general` instead.

For more information on guided generation, see Generating Swift data structures with guided generation.

Generate a content tagging response

Initialize your session by using the `contentTagging` model:

```
// Create an instance of the model with the content tagging use case.
let model = SystemLanguageModel(useCase: .contentTagging)


// Initialize a session with the model.
let session = LanguageModelSession(model: model)
```

The code below prompts the model to respond about a picnic at the beach with tags like “outdoor activity,” “beach,” and “picnic”:

```
let prompt = """
    Today we had a lovely picnic with friends at the beach.
    """
let response = try await session.respond(
    to: prompt,
    generating: ContentTaggingResult.self
)
```

The prompt “Grocery list: 1. Bread flour 2. Salt 3. Instant yeast” prompts the model to respond with the topic “grocery shopping” and includes the objects “grocery list” and “bread flour”.

For some queries, lists may produce the same tag. For example, some topic and emotion tags, like humor, may overlap. When the model produces duplicates, handle it in code, and choose the tag you prefer. When you reuse the same `LanguageModelSession`, the model may produce tags related to the previous turn or a combination of turns. The model produces what it views as the most important.

See Also

Getting the content tagging use case

`static let contentTagging: SystemLanguageModel.UseCase`

A use case for content tagging.