Training Best Practices

In Training for Natural Language, you learned how to provide natural language utterances to your capsule that teach Bixby how to understand your use cases.

This guide describes the limitations of training for natural language, provides principles to make your training more effective, and suggests best practices.

Platform and Training Limitations

Bixby is a powerful and unique tool, but as the famous quote says, "with great power comes great responsibility." Also, with great power comes some limitations:

- Bixby can be taught by anyone, but it can't be tweaked by true NL experts who want to fine tune results.
- Bixby works best with small, clean data. It can be overwhelmed by big, messy data.
- Bixby is built to scale horizontally with many diverse capabilities, but there are limits to how deep these capabilities can go.

With these limitations, you might wonder how best to do training. Primarily, you will need to use iteration, experience, and common sense. Trial and error is the best teacher for understanding what works and what doesn't. However, relying solely on iteration is incredibly time-intensive. Having a community of developers is also helpful.

Principles of Training

Since you might not have the time or expertise for effective training, we recommend following these three principles:

- **Tightly-Scoped**: Your capsule should strive to minimize the number of features that can be spoken about. Less is more!
- **Well-Discriminated**: Every query you support should clearly indicate that it belongs to your capsule and your capsule alone.

Obviously-Annotated: Values you extract from utterances should be simple and obvious to annotate. There should be no "learning curve" required to train your capsule.

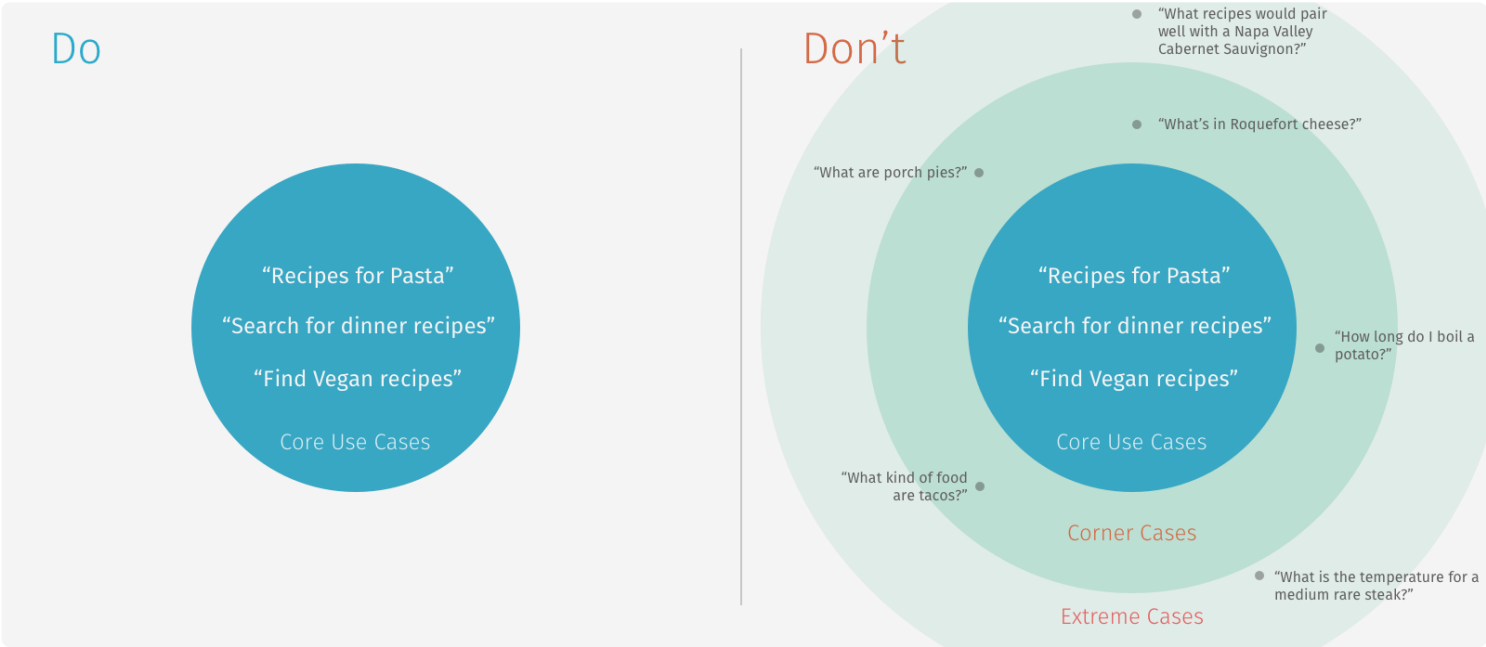
Using "Big Data" is usually sufficient to solve many natural language problems, but for the Bixby platform, you can inject smaller, representative, "smart" data and let it work as long as it follows these three principles: tightly-scoped, well-discriminated, and obviously-tagged.

Tightly-Scoped

Effective capsules minimize surface area.

If a capsule's scope is the surface area of the domain it covers, you should minimize this surface area. Every NL-facing feature that you add increases the amount and variety of data required to address it.

The solution to this is to model and train as little as possible and still have a useful experience. This means striving for a tight foundation:



Note

Keeping your capsule tightly-scoped also means not worrying about utterances that are outside of your capsule's scope. Bixby is responsible for figuring out which NL space the utterance belongs to and would not send utterances that don't fit your scope. For example, if you have a recipe capsule and a user asks "How many calories are in an apple?", you wouldn't need to train Bixby to say *I don't know how to do that* because Bixby will direct this to a more appropriate capsule. Similarly, you shouldn't have to train

something like "Where is the Taj Mahal located?" or "What is the weather like today?" if you're creating a recipe capsule, since these are wildly out of scope.

This circumstance might change with the introduction of Natural Language categories. If your capsule falls under a specific category but doesn't handle certain requests, there are some utterances you might have to train for Bixby to gracefully handle those situations.

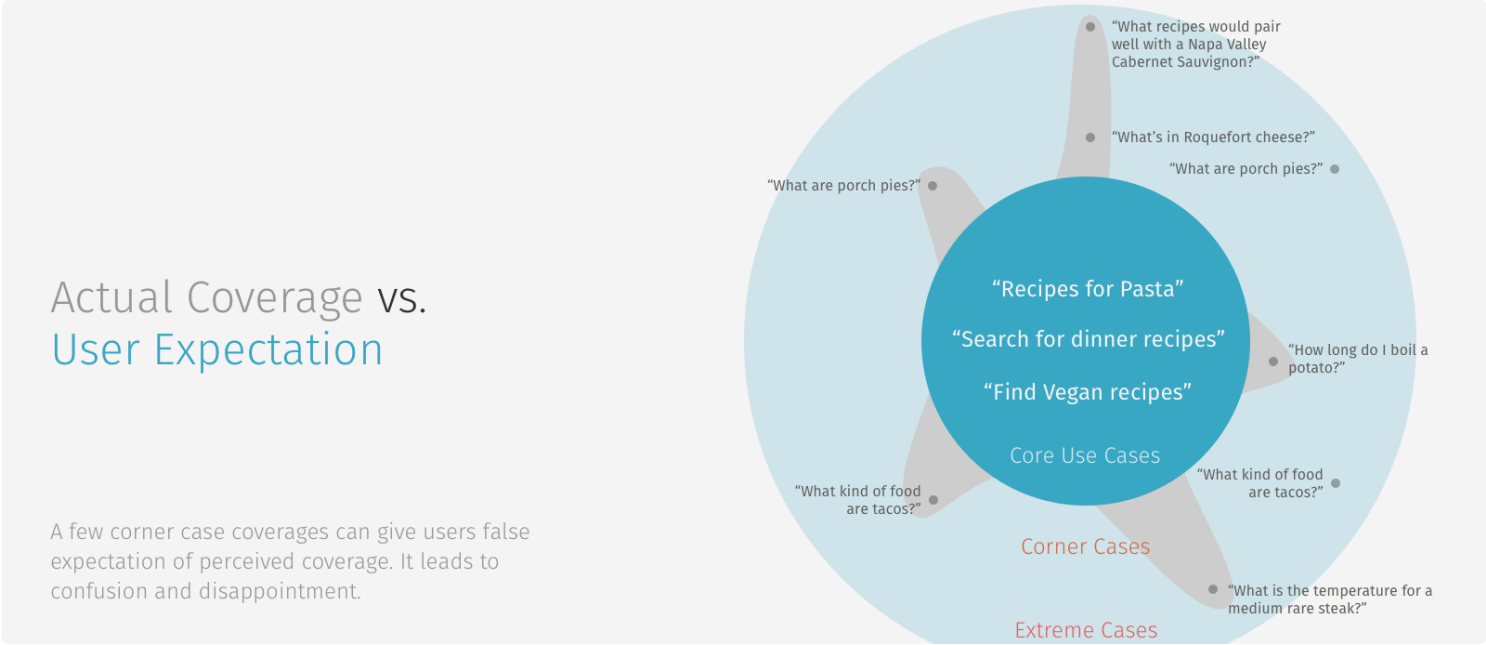
If you go out of scope of your capsule domain, you run into the following possible dangers:

Making it harder to classify the query, the goal, and the signals in them.

Increasing chaotic and less-sensible failures. The entire experience becomes unpredictable when confronted with real-world usage.

Lessening how robust it is to speech recognition errors, unanticipated rewordings, and out-of-vocabulary signals.

Setting up unrealistic user expectations. If a capsule has an uneven capsule surface area, it feels like Bixby should be able to do anything until the user tries it.



In short, design and build capsules that you can repeatedly use. Your capsule should be robust and well-crafted, which in turn makes it trustworthy (you can predict what it does) and understandable (you can mentally build a model of how it works).

You might need to build different capsules to address different use cases. For example, a capsule that returns recipes should be tightly-scoped. But you can build a separate capsule that does food question and answers (such as "how many calories are in yogurt?"). This is one way you can teach Bixby more, even if your use cases are not that tightly scoped.

Well-Discriminated

Effective capsules clearly claim their queries.

Finding the right capsule for a user query can sometimes be a challenge. If all user queries for a capsule obviously belong to that capsule, we can call that capsule **well-discriminated**. If not, you risk chaotic and unnatural failures, lowered confidence in query results, and increased training that is needed.

In general, you should reduce ambiguity by following these two recommendations:

> Make training queries well-discriminated
>
> Make vocabulary well-discriminated

Make Training Queries Well-Discriminated

If possible, make training queries well-discriminated based on their non-signal text alone. This is the best way for Bixby to be able to classify capsules and works best with speech recognition. In the examples below, if you get rid of the signal text (in brackets), you can still figure out which capsule the queries belong to:

> Recipes for [Greek Hummus]
>
> Status of [AA] flight [1530]
>
> What should I tip on a [$50] bill?

Make Vocabulary Well-Discriminated

If vocabulary is required, make the vocabulary itself clean and well-discriminated. In the examples below, the vocabulary (in brackets) is limited to unambiguous choices while more confusing options are struck out:

> Find Starbucks [Starbucks, ~~The~~, ~~Spot~~, Walmart]
>
> Find SFO [~~FOR~~, SFO, SJC, ~~THE~~, LAX]
>
> Find AA 1530 [AA, ~~NO~~, UA, B6, ~~OF~~, WN]
>
> Will it rain? [rain, ~~nice~~, snow, ~~bad~~, sunny]

Note that Bixby should be able to generalize user queries from these training entries and then catch more confusing instances, such as the airport code `THE` or airline code `NO`. If you find that this isn't happening, then you should just train the out-of-vocabulary cases as well, and Bixby will learn when those words mean things like airports or restaurants and also when they don't.

For more information on using vocabulary, see Extending Training With Vocabulary Developers' Guide.

Obviously Annotated

Effective capsules use easy-to-recognize signals.

Value labeling should be robust, in order to handle rewording and, at times, speech recognition errors. It is easier to show examples that aren't annotated well. We include some examples of not obviously-annotated signals, as well as some guidance on annotating primitives, not structures.

Examples of Not Obviously-Annotated Signals

The following examples show utterances that are not annotated well, explain the issues with each example, and give recommendations on how to avoid these issues.

Example 1: Tip Calculator

Consider the following problematic utterances:

```
(Split)[r:ChangeBillSplit] the bill.
What is the total split (4)[v:SplitAmount] ways?
```

The issues with this example are as follows:

Words should have consistent meaning through the domain. Here, *split* is mapped to `ChangeBillSplit` in the first example. In the second, 4 is mapped to `SplitAmount`, but *split* itself is not annotated.

Don't annotate verbs. Annotating verbs is generally a bad idea and is brittle. Goal annotation generally addresses verbs.

Make sure it is clear when text should be annotated. Should "split" be annotated at all? If it's unclear to a human, then it will be unclear to Bixby.

Example 2: Weather

Consider the following problematic utterances:

```
Do I (need an umbrella)[v:WeatherCondition:rain] today?
Will it (rain)[v:WeatherCondition:rain] today?
```

This example has two major issues. The first is that it abuses enums. Don't use enums unless they can be matched exactly with vocabulary. The more words in a match, the more brittle the vocabulary.

Also, as with the last example, the training entry is unclear. Specifically, it is unclear where the match should start and end. Should the match start at "need", "an", or "umbrella"? If a human is unsure, then Bixby will be too!

Example 3: Recipes

Consider the following problematic utterances:

```
(Butter)[v:IngredientName] recipes
(Pot roast)[v:FoodName] recipes
```

This example highlights context issues. Specifically, it is difficult to determine that "Butter" is an `IngredientName` and "Pot roast" is a `FoodName` without using vocabulary. Also keep the following in mind:

- If you annotate a Name with a value that is not in the vocabulary, Bixby won't know which to use.
- You need to be honest with using vocabulary and how clean and complete it can be. For example, it is impractical to list all `IngredientName` items.
- The line between `Ingredient` and `Food` can easily be blurred. For example, consider asking for "chicken recipes". Are you cooking a whole chicken or do you want recipes that use chicken?

The solution to clarify the meanings in this capsule is to to model all the food items as `FoodName`, annotate all out-of-vocabulary instances as `FoodName`, and then distinguish them using Roles when the language is very clear, for example if the utterance says "including Butter".

Annotate Primitives, Not Structures

You should annotate primitive concepts, not structure concepts, in utterances. In an utterance such as "find shirts under $40", you might have a `Currency` structure which includes `CurrencySymbol` and `CurrencyValue`. Instead of trying to annotate "$40" as `Currency`, annotate "$" as `CurrencySymbol` and "40" as `CurrencyValue`:

```
find shirts under ($)[v:CurrencySymbol](40)[v:CurrencyValue]
```

Provide an action in your capsule to combine the two primitive concepts into the structure concept. Bixby will use that action when it needs to during execution.

Note

> The exception to this rule is structure concepts that have been trained as patterns, such as the core `viv.time.DateTimeExpression`. Patterns provide vocabulary to tell Bixby how to interpret structure concepts, so these structure concepts can be used directly in NL training.

General Recommendations and Best Practices

When you are adding training examples, less is more. Let Bixby do as much work as possible. Set the goal, identify any values, and verify the plan.

Here are some other best practices to keep in mind.

Do:

Design and train capsules intentionally.
When designing capsules, you should focus primarily on use cases. It is not generally useful to satisfy every user request. This applies not to just models, but in training as well.

Choose the goal that's most helpful to the user.
Sometimes this means giving users a little more than they ask for. If a user asks "is there Mexican food nearby?", it won't be very useful to answer "yes" or "no". So the goal should probably be a restaurant, rather than a judgment. If the user asks "will I need an umbrella tomorrow?", you should choose a goal that will answer the question and will also show a weather forecast for context.

Prefer a goal that's a concept rather than an action.
Concepts are generally more flexible. The main exception is when an utterance is ordering a product or booking a service. In these cases, an action is more appropriate.

Focus annotations on nouns and noun phrases.
If you need to adjust behavior by tweaking a verb with a signal, then consider using a different goal since the subject of the sentence has changed.

Label only the user's intent, not additional information on how to achieve it.
Label only concepts or actions that are actually apparent or explicit in the user's intent.

Create training examples for each supported language separately.
For more information on how to localize your capsule and localized training, see How to Localize Your Capsule.

Ask yourself "Is it easy for me to explain to someone how any given utterance should be tagged in my domain?".

If the answer is "No", model the problem differently.

Check the Training Evaluation section of your submission reports and try to get the Not Learned rate to zero.
Not Learned training examples usually exhibit critical issues like missing vocabulary, inconsistent training, and misuse of primitive types. Those issues will affect many user-level queries in practice. Despite the fact that your submission might have been accepted, Not Learned training is **not** normal and should not be ignored. The training in a functional capsule should nearly always be 100% Learned.

Don't:

Don't try too hard to find a word in the utterance to tag with your signal so that you can force the intent you want, or it is probably not going to work!
The forced training entry might work for that single instance, but it will not generalize all queries of its type. You should not have to tweak the training to make your feature work.

Don't overextend your vocabulary!
Specifically, remember these tips:

- **Train closed-type vocabulary very sparingly!** Train enums only when the symbol value is similar to the annotated text or can be sensibly substituted for it. We highly recommend that you never use routes.
- **Stick to a small number of open-vocab input types.**
  Make sure that if you use more than one, then the meaning of each word can be distinguished.

Related Resources

Training for Natural Language
Localizing Capsules