Understanding why intent resolutions are sometimes not what you expect

Grant Ronald, May 2018

One of the most common questions that gets asked when someone starts building a real bot is "Why am I getting strange intent resolutions". For example, someone tests the bot with random key presses like "slkejfhlskjefhksljefh" and finds an 80% resolution for "CheckMyBalance". The first reaction is to blame the intent resolution within the product. However, the reality is that you've not trained it to know any better. This short article gives a high level conceptual explanation of how model do and don't work.

What is intelligence

It is worth making clear that what we call "artificial intelligence" within any chatbot platform is, in the most part, just a whole load of mathematical and statistical probabilities. A well trained bot only knows that, for example, "taxi" and "cab" may be considered related terms in language not because it has knowledge of taxis, cars and cabs but because there is a mathematical "closeness" between how these words are represented within the platform based on the fact they are used in similar ways within a training corpus such as Wikipedia.

Goal of a good model

Thus, a good chatbot model should be able to deal with all of the variances in language and grammar. It should be able to cope with the fact that you say "order me a cab" when it was trained with phrase such a "phone a taxi for me".

The flip side to that, however, is simply because words appear in the training data is no guarantee to correct intent resolution – "balance on the river bank" has nothing to do with a financial transaction even through "balance" and "bank" might be part of the training data for a banking bot. In this example "bank" and "balance" have a different meaning when used in the context of "river".

Knowledge the bot doesn't have

Finally, it's also worth making clear that when you say "*Why am I getting strange intent resolution?*" it is only strange because you inherently know that "slkejfhlskjefhksljefh" is a nonsense input. If you have no inherent knowledge of the language then the model (which only knows what you teach it) is going to do its best based on the training it has been given.

Consider if YOU were the model and you were being trained in Klingon (for those who know Star Trek) and you got the input nuqDaq 'oH tach'e' –you have not specifically been trained in that phrase or even some of the words, but how would you know if that was an expected input to be resolved or was it random key presses?

Understanding a simple model

Let's consider a simple example. You have a bot which can deal with two intents, banking or pizza and let's assume that you have every possible phrase imaginable to define each of these two intents. We might visualize this as in figure 1 where each red dot is a representation of a phrase related to ordering pizza. The blue dots represent phrases for performing a bank transaction. We might further imagine that where red and blue dots are closer together MIGHT represent phrases with high levels of commonality between the two intents.

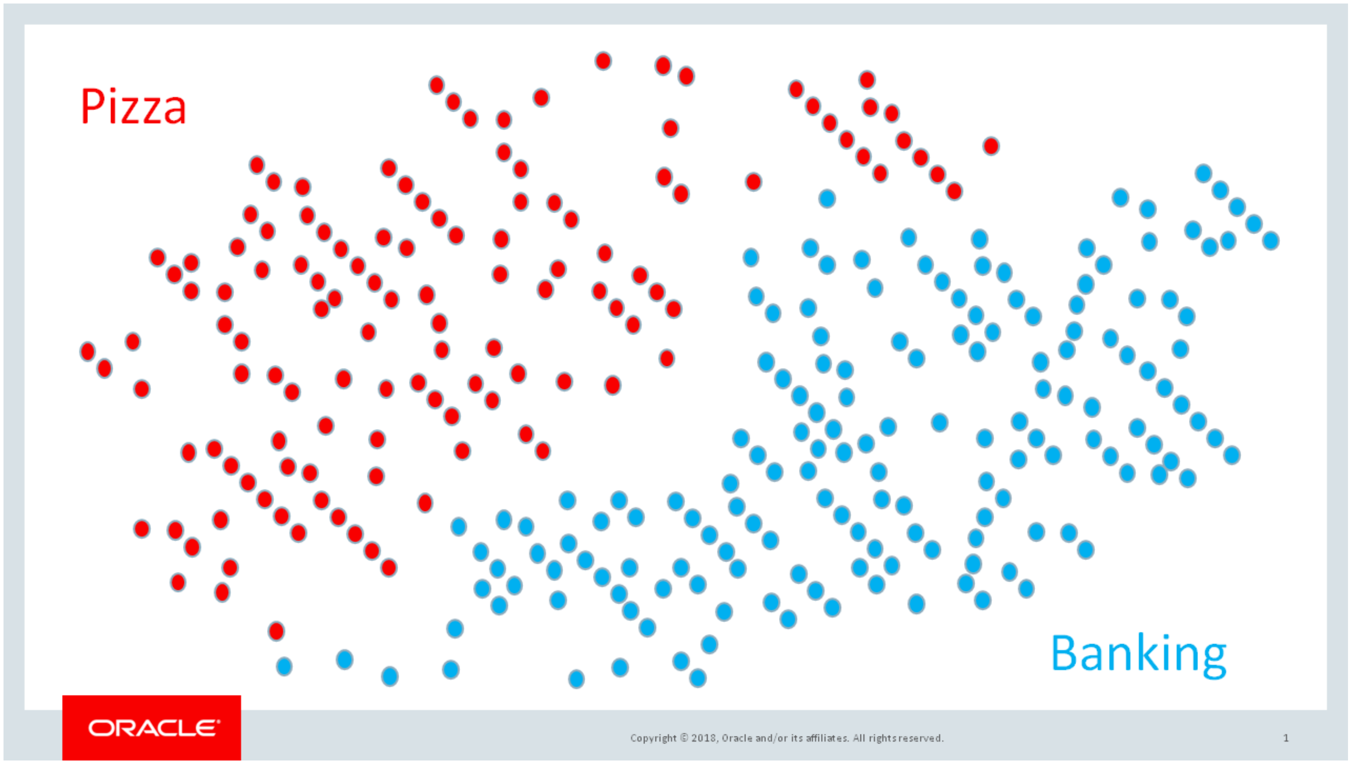

Figure 1 - a two intent model

Now, given all of this training data, the job of a model is to try and represent all of that data and the relationships between it, as a mathematical function. So one model might be represented as a straight line function that diagonally splits the data as show in figure 2. This gives a pretty good split although we can see some data points which fall outside the desired intent.

However, with a richer mathematical function the model could be as per figure 3 where a curve line is used to model the intents.

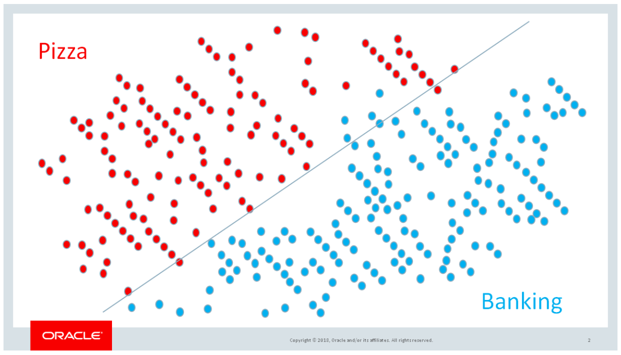

Figure 2 - straight line model

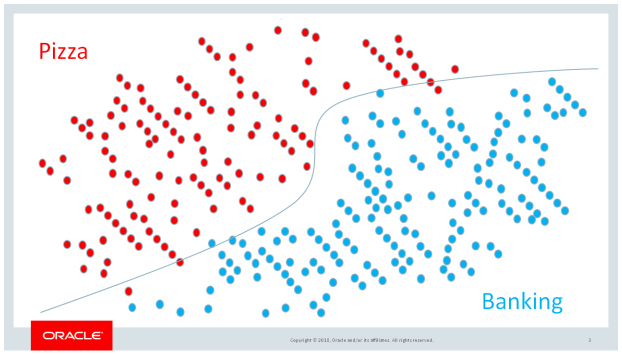

Figure 3 - curved line model

The first point to make is that when the user inputs a sentence, it's not being checked against the data. Instead it's being plotted against the model. The second point is that EVERYTHING the model knows is being represented in the above figures. That means that in this simple case, any input sentence will result in a probability that lies simply above the line (Red/pizza) or below the line (Blue/banking). There is no third option because this model has been trained to understand only two possibilities. So the sum of the probabilities will be 100% (this is true for TrainerTm in Oracle Intelligent Bots, for Trainer Ht it is rules based on so does not work on the same principle).

Poorly trained models

The above example involved a model with a rich set of training data with over 100 training points. However let's consider the EXACT same use case but this time the bot developer has only supplied a fraction of all possible training data. Figure 4 below show the smaller data set with the original "best fit/perfect model", however based on the small data set the actual model is more likely to a straight horizontal line as shown in figure 5.

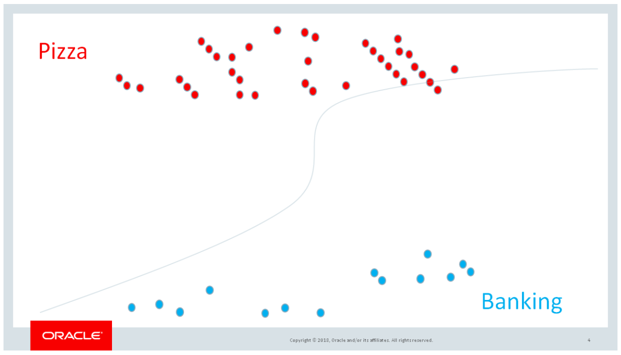

Figure 4 - limited training data

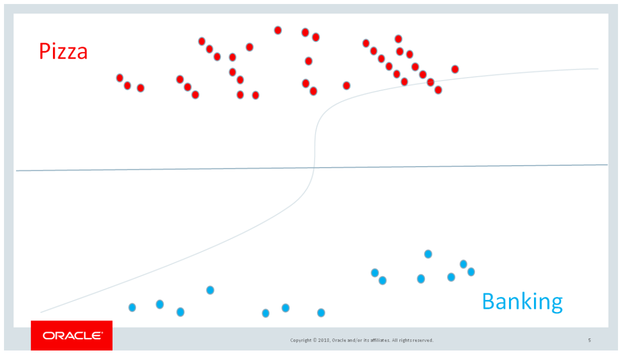

Figure 5 - Straight line model based on limited training data

Whilst the model looks ideal with the training data supplied, when we look at the list of all possible phrases the user might ask (figure 6), we can see that for this model there is a considerable number of phrases which are going to be misclassified when compared against the "best fit/perfect model"

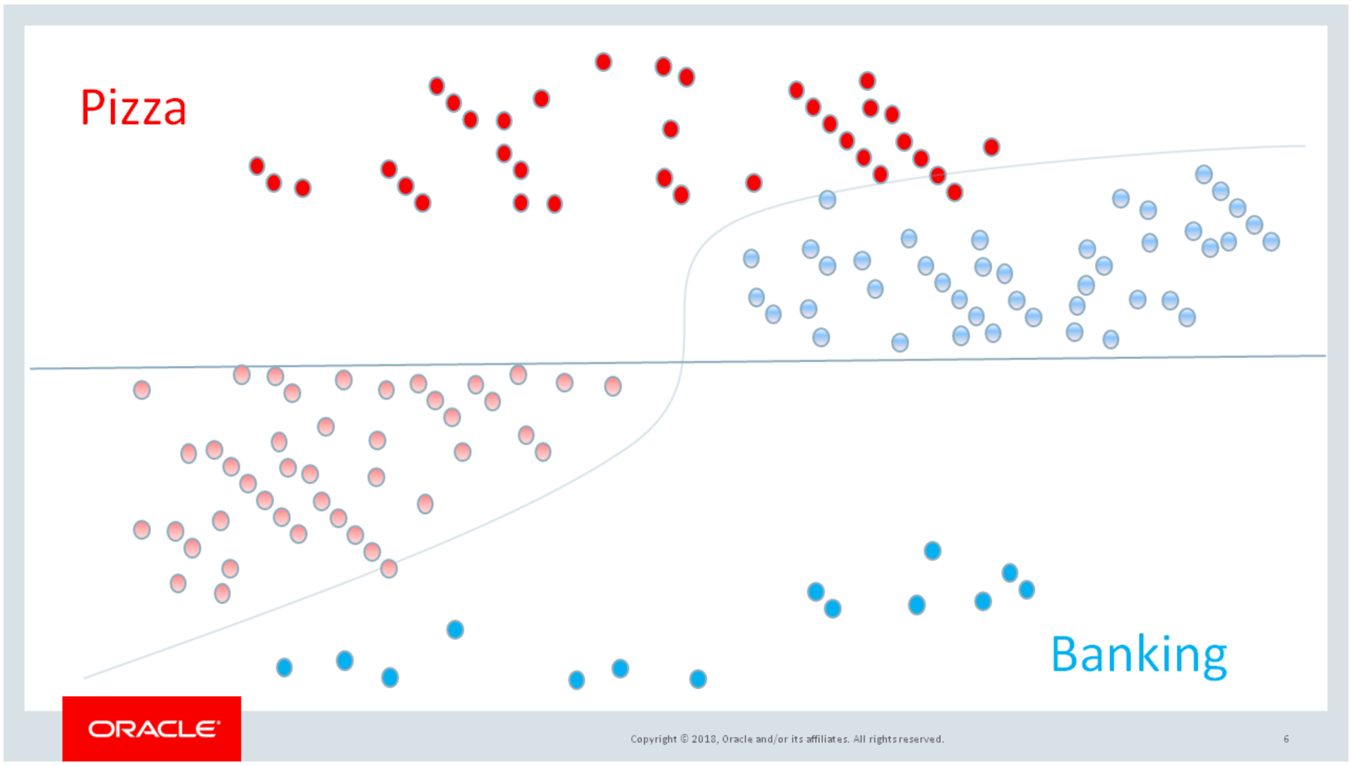

Figure 6 - Misclassification based on a poorly trained model

From the above explanation the main takeaway is that the model is only as good as the data you give it. In the above example the key problem was obviously the lack of enough data, but more importantly the lack of a spread of data. Most of the training data was “outlying examples” where the data points might be viewed as definitely being related to a specific intent. However there were less data points at the border of both intents meaning the model failed at that point.

Resolving unresolvable inputs

Going back to the original “best fit/perfect model” let us now assume that the user inputs some totally random phrase such as “kljdghklsdjfgh”. The problem here is that he model has been trained to know only two things: pizza or banking. Or more precisely the model has been trained to give a probability based on where the data point is plotted versus the line as represented by the model. Both you and I know that “kljdghklsdjfgh” has nothing to do with banking or pizza but that is only because we have knowledge of language (or pizzas and banking) that the model has not. And the reason the model doesn’t know this is because it has not be trained to know otherwise.

The model is going to do exactly what it was trained to do and that is plot the input phrase against the model and give a probability – since the model can only discriminate two possible intents, then that is what it is going to do.

In essence you need to be able to train the model with examples which represent data which should be regarded as unresolved. Now, you might argue that the list of all training phrases not related to pizza or banking is almost infinitely huge but still, if we want the model to understand phrases which are unresolved we have to give it data to learn from.

Note, that this guidance is not something which is particulat to Oracle Intelligent Bots. This tends to be a general recommendation – see the following recommendation for the Microsoft platform https://docs.microsoft.com/en-us/azure/cognitive-services/luis/luis-resources-faq#is-it-important-to-train-the-none-intent,

To visualize this lets consider a model which is now represented in three dimensions. The existing model of pizza and banking exists exclusively on the x and z plane, but now we can start defining a third intent which represents phrases which will be handled as unresolved in a pizza/banking use case.

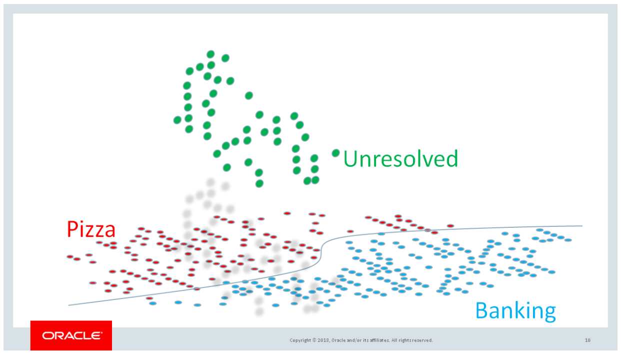

Figure 7 - plotting data which should be unresolved

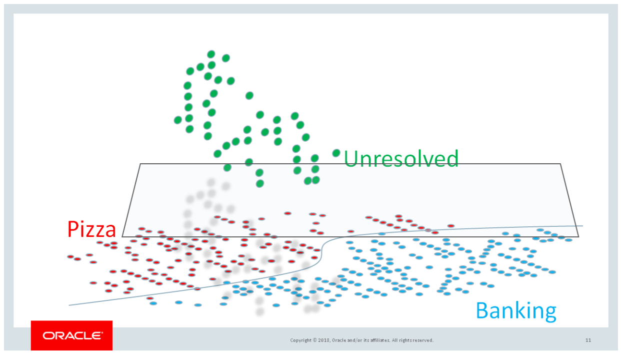

Figure 8 - The model now operates on multiple dimensions

In this simplified example we can now extend our model to be not only a curved line through the x and z plane, but to also include a plane which runs parallel to the current model and discriminates pizza and banking from other possible phrases.

Now, whilst you will never be able to train the unresolved intent with all the combinations of keys that people could hit on the keyboard, by giving it training data such as “djkghkdjfghkdj”, or “The quick brown fox jumps over the lazy dog” or “I love Bon Jovi” you are at least giving it information which not only strengthens it finding those or similar phrase as unresolved, but more importantly, you are training the model to understand that there is an 3rd option and thus reducing the probability that random or non-uses cases will be incorrectly resolved to pizza or banking.

Handling small talk

It is also worth acknowledging that not all non-use case specific conversation is spam. Greetings such as “hi” or “hello”, or conversational smalltalk such as “are you bot”, and even expletives or requests for help are all elements of a bot conversation that any good bot should handle. This conversational “smalltalk” would typically be defined as multiple intents which in themselves will enrich the model in that they define not only typical smalltalk, but also help to define what a bot use case is not.

Summary

One of the first steps in getting hands on with Bots is to try out the natural language processing to gauge how well it works. That is a natural thing to want to do given it’s the core of the product. However, (particularly with Trainer Tm) limited intents, limited training utterances and not handling unresolved phrases are likely to lead to unexpected results.

For a detailed description on best practices for training your model check out https://blogs.oracle.com/mobile/techexchange-first-step-in-training-your-bot