PM Question Processing Final Project: 20 Questions

Wellesley Boboc
Universität Potsdam
Matriculation number: 805704
boboc@uni-potsdam.de

Anna-Janina Goecke
Universität Potsdam
Matriculation number: 777707
goecke@uni-potsdam.de

Rodrigo Lopez Portillo Alcocer
Universität Potsdam
Matriculation number: 805606
Rodrigo.Lopez@mpikg.mpg.de

Elizabeth Pankratz
Universität Potsdam
Matriculation number: 804865
pankratz1@uni-potsdam.de

Abstract

For our final project for the Question Processing course, we chose to develop a system that can play the game of 20 Questions and guess which animal a human player has in mind. We implemented a decision-tree-based system that splits our knowledge base along a feature, uses a template to generate a natural language question about that feature, records the user's input, and ultimately makes a guess as to the user's target animal from a probability distribution over all animals. Users can add new animals to the knowledge base, and missing features will be interpolated by one of two methods. Our game can be played in the command line and the 20Q system has won 40.4% of the games it played, a number that is only expected to rise as the system logs more games.

1 Introduction: Task and motivation

Our task was to build a system that will play the game 20 Questions (20Q). A human player will be able to think of a target object, and the system will strategically select questions that allow it to narrow down the candidate objects in its knowledge base. It will incorporate the answers it receives and ultimately make a guess about what that target object could be. If the target object that the human player has in mind is not already in its knowledge base, the system will add it in based on the information the user has provided.

This task is interesting and challenging. On the one hand, it involves the computational-linguistic subtask of question generation (given a feature in the dataset, generate a natural-language question asking about that feature to display to the user), and on the other, the engineering subtask of manipulating the knowledge representation in accordance with the answers that the human player provides. For the latter, we will need to strategically select the best question to ask, incorporate the answers from the user as the game is played, and at the end, add previously-unseen objects into the knowledge base.

We begin by outlining the previous work in Section 2 that inspired and informed various aspects of our approach. Section 3 discusses how we created the knowledge base that our 20Q player draws on, and in Section 4 we turn to the implementation of the player itself. Section 5 evaluates its performance, and Section 6 discusses a few salient limitations of our system that would need to be improved if our system were to hold its own against any of the other more sophisticated 20Q players out there.

2 Related work

We will briefly explore two areas in the literature that are relevant for our implementation: existing approaches to implementing 20Q, followed by rule- and template-based question generation.

2.1 Previous 20Q implementations

Previous approaches to the implementation of a 20Q system have made use of diverse methods including probabilistic models (Dey et al., 2019), reinforcement learning (RL; Hu et al. 2018), and variations of artificial neural networks (Reddy et al., 2017; Burgener, 2006; Tonin et al., 2018). The knowledge that the 20Q system has is often represented in a knowledge graph (e.g. Dey et al., 2019), though some more sophisticated approaches also manage without (e.g. Hu et al., 2018).[1] Here, we will briefly discuss the relative merits of these implementations and how they inform our work on this project.

[1] A knowledge graph is essentially a graph where the nodes are entities and the edges between them are facts that connect the entities. For instance, a node *Macron* might be connected to a node *Paris* by the edge *lives in*, representing knowledge of the fact *Macron lives in Paris* (example from Godin et al., 2019).

We begin with some probabilistic approaches. In general, these are characterised by maintaining a probability distribution over the set of outcomes. Consequently, none of the possible outcomes is ever totally discounted or thrown away—just associated with a lower probability. This approach is good for situations in which the questions are answered by the users in a "noisy" way (e.g. when users answer inconsistently or wrongly). In those cases, the system is still able to choose what the correct target object might be, even though the user has answered in a way that might seem incompatible with that object.

Dey et al. (2019) implemented a probabilistic model which operates on a dataset of weighted edge-node relations of a knowledge graph and updates throughout the course of the game. Here, the main idea of adjusting probabilities at every time step was exploited to generate a model that is able to predict the correct target object in fewer than twenty questions. Of particular interest is the way the model handles incorrect answers from the human player: the question generator does not fully reject or accept a certain object as being the target after every answer. Instead, it rebalances the probabilities at each step in the game. To identify the target object, the model categorizes the questions into two layers, a primary layer (wide range of objects) and a secondary layer (specific range, targeted towards a small set of objects). Even though the model has been proven to perform very well, i.e. half of the target objects could be identified in fewer than ten questions, their work is very limited in that it is designed to only apply to Bollywood movies. However, the use of a concrete domain-specific knowledge base and a probability distribution across outcomes are approaches that we adopted from Dey et al. (2019) in our own 20Q implementation.

Hu et al. (2018) also rely on a probability distribution over all objects which is then updated according to the answers. However, their approach is different insofar as they use an RL framework: they implement a policy-based system of 20Q that uses reinforcement throughout the game. Instead of using a knowledge graph, the model for selecting questions is based on RL procedures trying to find the optimal reward function. Hu et al. (2018) suggest a neural network which learns the immediate reward at each time step to improve the overall performance of the model, since only receiving a reward at the end of the game wouldn't allow the system to learn for each question. The model continually improved its win rate over time and was shown to be able to identify the target object within 14 questions. While this method clearly has excellent performance, incorporating RL into our project was not feasible for various reasons.

Another approach that is too sophisticated for us, but nevertheless useful to know about, is the use of neural networks. For example, Reddy et al. (2017) propose the application of knowledge graphs to generate sets of question-answer pairs within a Recurrent Neural Network architecture by deriving triple relations from given entities. The triples are composed of a subject, an object (both represented as nodes in the knowledge graph), and a predicate (represented as an edge in the knowledge graph). The model consists of two units: the *Question Keywords and Answer Extractor*, which directly selects necessary information about an object from the knowledge graph, and the *Natural Language Question Generator*, which is used as an encoder and decoder of the object's representation. Since this model has been able to outperform comparable approaches on question generation, it would represent the "next step up" from the question-generation methods we apply here.

Perhaps the most widely-known implementation of the 20Q game is that of Burgener (2006), which can be found at `20q.net` (accessed 14.08.2020) and which is also a popular toy. With more than 88 million plays, Burgener's implementation has a precision rate of 80% when it asks twenty questions, and 95% for twenty-five questions. The patent for the game describes the implementation of the deep artificial neural network, which is structured as a matrix of target objects by questions. Each cell of the matrix contains an input-output connection weight, which defines the relationship between the questions/answers and the target objects. The network has two so-called modes: the first takes questions as input and targets as output, and the second takes target objects as input nodes and questions as outputs. The first mode maps answers to weights, while the second mode ranks questions. Similar to Dey et al. (2019), target objects are prioritized, rather than filtered. This is a primary motivation for Burgener's choice of architecture, as it allows the model to correctly predict the target even when given incorrect or inconsistent answers, as mentioned above. As Burgener explains, it also allows the system

to take into account cultural/demographic differences that may result in inconsistent answers about a given target object. This is a consideration we should also be mindful of in our implementation, as our proposed model operates on the assumption that the user is providing truthful answers (and that inter-user agreement would be high). The system of weights also allows for a more complex set of inputs than binary yes/no (e.g. sometimes, maybe, depends, rarely) where the degree of certainty of the answer is reflected in the weights used. Our decision to include three answer choices—"yes", "no", and "unknown"—was inspired by this work, and although our final implementation did not offer as wide a range of possible inputs as Burgener (2006), expanding the range of answer choices of our current model is a potentially fruitful avenue for future work.

Finally, Tonin et al. (2018) describe another artificial neural network implementation of the 20Q framework as part of a brain-computer interface to enable people with motor impairments to communicate. The system uses a weight matrix to store the strength of the connection between target statements and questions (where negative weights indicate that the expected answer to the question is no, and vice-versa for positive weights). After 15 questions, the model checks to see if there is only one target statement with a positive value. If there is no single positive value after twenty questions, the network returns the statement with the highest current value. When the network correctly estimates the target, the weight matrix is updated. This paper is particularly interesting in that it also shows how a 20Q implementation may have useful applications outside of the realm of games and entertainment.

2.2 Template-based Question Generation

We turn now to existing approaches to question generation (QG). The method that we will focus on in particular are rule- and template-based approaches, since this is what we will use in our implementation. We have no need for more complex approaches to this task, because the questions in 20Q are always polar questions (i.e. questions eliciting yes/no answers, cf. Huddleston 1994) and always syntactically quite limited.

Rule-based approaches tend to begin with a declarative sentence and then reformulate it into a question by applying rules to dependency structures in the sentence (e.g. Khullar et al., 2018; Mhatre et al., 2019), while template-based approaches essentially involve constructing a question frame with a placeholder variable (e.g. Zerr, 2014; Mandasari, 2019; Fabbri et al., 2020). In what follows, we will briefly review both.

One approach to rule-based QG is the work of Mhatre et al. (2019), which is based on keyword modelling using NER to generate questions from an input sequence. Each input sentence is preprocessed and parsed to resolve anaphoric reference. Thereafter, NER is used to identify the type of entity, which is important for the choice of *wh*-component for the QG part of the model. Depending on the output of the NER procedure, the appropriate *wh*-pronoun is chosen. One type of question they create is yes-no questions, which is the type of question we will focus on. To construct this kind of question, the authors simply perform subject-auxiliary inversion.

Khullar et al. (2018) concentrate on rule-based QG using relative pronouns to achieve high syntactic accuracy and semantic suitability. Their system uses the spaCy dependency parser to evaluate the syntactic structure of sentences.[2] Firstly, the input sentence is parsed to gain information about the presence of relative pronouns and about several linguistic features. Afterwards, this information is input to one rule within a predefined rule set to create the questions. The correct *wh*-component is then determined according to these rules, resulting in a syntactically coherent question.

Note that these methods start with an input sentence, which is then dependency-parsed or analysed using NER. In our implementation, though, we are not starting with input sentences, but rather non-sentential units (namely, our features, which can be NPs like *the ability to fly*, VPs like *take breaths*, APs like *active mainly at night*, and PPs like *in a group*, to name a few). For this reason, a template-based method will be better-suited for our purposes.

Template-based methods for generating natural language questions involve predefined pieces of text containing a placeholder variable which has to be replaced within the QG step itself (Mandasari, 2019). One major criterion of such a question template is that it has to be suitable for a range of different words. Accordingly, the approach of Mandasari (2019), for instance, exploits the idea of classify-

[2] `https://spacy.io`; accessed 14.08.2020.

animal	have_hair	feathers	produce_eggs
aardvark	1	0	0
antelope	1	0	0
badger	1	0	0
bass	0	0	1
bat	1	0	0

Table 1: The first five rows (instances) and three columns (features) in our knowledge base

ing sentences by combining semantic role labelling with part of speech (POS) tagging and named entity recognition (NER) to construct proper question templates. Another approach to QG can be seen in work by Zerr (2014), which makes use of POS-tagging methods to construct questions. The author points out that QG heavily depends on semantic as well as syntactic accuracy. Our 20Q implementation draws from work by Zerr (2014) in which POS tags are obtained from a corpus and then matched to predefined question templates. Fabbri et al. (2020) also investigate template-based techniques to convert sentences from a corpus into questions.

All of the above-mentioned systems consist of a simple structure by using a POS tagger, dependency parser, or NER methods. By adopting template-based QG using the POS-tagging approach, as will be discussed in Section 4.3 below, we believe we were able to construct simple yet appropriate polar questions for our 20Q model.

3 The knowledge base

Our game uses a tabular knowledge base that, in its base form, contained 152 animals and 63 features for each object (though it is extended through gameplay; see Section 4.4). Each cell in the table is populated with a 1 or a 0 to indicate whether the given animal does or does not have the given feature. An example is shown in Table 1.

Our first implementation of the 20Q game made use of a knowledge base available on GitHub, which consists of 100 animals and 28 features for each.[3] However, this dataset alone is far from exhaustive and does not include many animals that would be likely to come up in a typical 20Q game, such as *dog* or *horse*. In addition, while its 28 features provided a good jumping-off point, we wanted our system to have a wider range of features from which to choose when generating questions, which we believed would both improve the 20Q player's performance and make for more interesting and less repetitive gameplay.

To address this, we chose to concatenate the initial knowledge base and another dataset containing information about animal features. We found the dataset *Animals with Attributes 2*, developed by Xian et al. (2017), to be suitable for our purposes, as it contains 50 common animals and 85 features. Removing duplicate animals and features left us with a fairly sizeable set of animals, and after the manual addition of several missing animals that were deemed to be relatively likely targets, we ended up with the list of 152 objects for our model to draw from.

In terms of choosing which features to include, we made use of a large portion of the existing features from the two base datasets, and then further expanded the knowledge base by adding a number of features that were not in the datasets but might be asked about by a human 20Q guesser, such as *Is it bigger than a microwave?* and *Would you find it on a farm?*, to give two examples. Features that applied to only a small number of animals were also added, as a way to distinguish alike animals or capture idiosyncratic or otherwise distinctive characteristics. In addition, we removed a number of features that could be considered mutually exclusive to avoid redundancy (ostensibly, if an animal has 1 for the feature `predator`, it should have 0 for the feature `prey`). We also removed features that we deemed confusing, unlikely to be common knowledge, or not particularly helpful for our purpose.

Having chosen all the animals and features that we wanted to include, we manually filled in any missing values resulting from either the concatenation of the two sets or the addition of the new features and animals. We debated whether to incorporate some degree of uncertainty into the knowledge base to accommodate for the fact that different human players may have different knowledge or beliefs about the target animals. (For instance, some players might consider spiders to be dangerous, whereas others may not.) However, for reasons of both simplicity and practicality for our implementation, we decided to preserve the binary form of the data in the knowledge base; to the best of our knowledge, the knowledge base contains factual information about each feature (wherever possible). Also, instances where the feature could be ambigu-

[3] `https://github.com/gibbsbravo/20_Questions/blob/master/knowledge_base.csv`; accessed 14.08.2020.

Feature type	Example features
objective	`feathers`, `type_of_insect`, `horns_or_antlers`
rather subjective	`on_a_farm`, `commonly_eaten`, `bigger_than_a_microwave`
subjective	`dangerous`, `smelly`, `sleep_a_lot`

Table 2: Examples of objective, rather subjective, and subjective features in our knowledge base

ous, such as those related to color and size, were marked with 1 to capture the fact that the given feature *could* be said to be true for the target.

We did, however, decide to manually group the features into three categories ranging from objective to subjective, with "rather subjective" as a middle ground, so that questions that would likely have high rates of inter-user agreement (the objective ones) can be posed before those that are more prone to differences in opinion (the rather subjective or subjective ones). Table 2 shows examples of each of these feature types.

Lastly, since human players have the ability to add new animals to the knowledge base in our final implementation, our development knowledge base does not have to be exhaustive. Over time, we hope that animals that are missing from our knowledge base will be added by players, ultimately resulting in a more comprehensive database for our system to draw from.

4 Implementation

In this section, we introduce our implementation of the 20Q gameplay. We opted to implement a relatively deterministic decision-tree-based 20Q system, rather than using a deep learning or RL paradigm. This was so that the inner workings of our system would remain transparent and everybody would be able to understand the code and work on improving it. For those of us with an exclusively linguistic background, such a deep dive into deep learning or RL felt like too much too soon. This naturally means that our system is less sophisticated than it could have been if we had used a more modern paradigm, but on the other hand, its inner workings are also much easier to understand.[4]

4.1 Feature selection

The first question to tackle was, "How will the system decide which feature to ask about at each step?" Critically, our basic motivation was to choose the most informative feature at each step, so that we narrow down the potential animals most quickly. (If the most informative feature were always asked about first during the actual gameplay, the game would be boring and repetitive, so we add some non-determinism later to keep things interesting. Here, though, we focus first on the basic mechanism for determining the informativity of a feature.)

We used a model in the style of a decision tree to tell us which feature is most informative at any given step. A decision tree is "defined by recursively partitioning the input space, and defining a local model in each resulting region of input space" (Murphy, 2012, 545). In our case, the input space consists of the knowledge base described in Section 3 above, and this knowledge base is recursively partitioned by each successive question that the system asks and the user answers.

In standard classification decision trees, the space is split by the feature that minimises the entropy (i.e. the one that maximises the information gain; Quinlan 1986) in each partition. However, that method is not applicable here. That method requires an $n:1$ mapping of instances to each class, which is the usual set-up in classification problems: one class contains multiple instances. In our task, though, each individual object equates to a class (i.e. we have an `aardvark` class, an `antelope` class, and so on), so there is initially only one instance per class. The structure of our data makes our problem a non-typical classification task, so a different method must be used.

Instead of using the information gain measure itself, we use a stand-in: we orient ourselves around the size of the two partitions of the input space that result from splitting on a given feature, and we select the feature that produces the partitions that are closest to each other in size. To illustrate, say that we split on the feature `have_hair`. We would end up with one partition containing 65 animals that have no hair (i.e. where `have_hair` $= 0$), and another partition containing 87 animals with hair

[4] All our code can be found on GitHub at `https://github.com/epankratz/twenty-questions`.

(i.e. where `have_hair` $= 1$).

To see how even this split is, we take a ratio of these two numbers. We want this ratio to be as close to 1 as possible, since that would represent a perfect split of our input space in half: $\frac{50}{50} = 1$. We want an even split because, since we only have two values for each feature (yes or no, 1 or 0), consistently splitting our input space in half results in the highest information gain (cf. Quinlan, 1986; Bishop, 2006).

For illustration, looking at the feature `have_hair` in the full knowledge base, we have

$$\frac{|\texttt{have\_hair} = 0|}{|\texttt{have\_hair} = 1|} = \frac{65}{87} \approx 0.75.$$

We call this value, 0.75, the split cardinality ratio (SCR) for the feature `have_hair`. In general, the SCR for a feature f is defined as shown in Equation 1.

$$SCR(f) = \frac{|f = 0|}{|f = 1|} \tag{1}$$

The best feature f_{best} out of all features f is the one for which the distance of the SCR from 1, i.e. $abs(1 - SCR(f))$, is closest to zero. Formally, it is the solution to Equation 2.

$$f_{best} = \underset{f}{\text{argmin}}\ abs(1 - SCR(f)) \tag{2}$$

However, if we were to always choose f_{best} to ask about at every step, the gameplay would be rigid and repetitive. To make for a more varied gameplay, we decided to randomly sample a feature to ask about in proportion to its $abs(1 - SCR(f))$ score. This meant transforming the distribution over f of $abs(1 - SCR(f))$ into a probability distribution, such that the features with the lowest $abs(1 - SCR(f))$ score would have the highest probability of being chosen at each step. This means that any reasonably informative features are reasonably probable candidates.

To do this, we first transformed the $abs(1 - SCR(f))$ values into probabilities of the opposite magnitude (so that small distances from one have a high probability of being chosen) by determining the maximum distance from one and subtracting every feature's distance from that (and adding one, so that the feature with that maximum distance value does not have a probability in the end of zero). Then we scaled these transformed values into a probability distribution and sampled a feature from this distribution proportional to its probability.

This method does have certain drawbacks (see Section 6), but in addition to its simplicity, it also comes with one further advantage. Not all of the features in the knowledge base are independent; for example, if you know that an animal is a mammal, you probably also know that it has hair. So, an intelligent system should not ask about both of these features, since it should have gained the knowledge contained in both of them by asking about only one. And, indeed, because of the way we bisect the database based on the features that best distinguish the animals still contained in it, we automatically avoid asking about highly correlated features.

For example, let us pretend that the five animals and three features in Table 1 constitute the entire knowledge base. If we tell the system `have_hair` $= 1$, then it bisects the database and removes all animals with `have_hair` $= 0$. However, since all animals with `have_hair` $= 0$ also have `produce_eggs` $= 1$, the feature `produce_eggs` now also only contains the value 0. This makes it unsuitable as a feature to distinguish animals, so it will not be asked about in subsequent rounds. (Even if two features are not perfectly correlated like these two are in this tiny knowledge base, after bisecting the data on one of them, the mixture of 0s and 1s in the other feature will be less even, so it will be less likely to be selected as a feature to ask about.)

4.2 Guessing the animals

Eventually, the gameplay will reach a point when the remaining animals in the pared-down knowledge base can no longer be distinguished by any of the features, since all the features will contain all 0s or all 1s. At this point, the game moves on to the stage of guessing individual animals.

We were inspired by the approaches of Dey et al. (2019), Hu et al. (2018), and Burgener (2006), who all maintain a probability distribution over outcomes that is updated over the course of the game based on the answers that the user provides.

The motivation behind this approach, as we mentioned above, is to improve the flexibility of the system, and in particular to not reject outright any outcomes that are incompatible with the user's answers. Instead of removing these from the pool of potential answers immediately, their probability is simply reduced, but this means that they are still "in the running" to be guessed by our system, after it has guessed the higher-probability items.

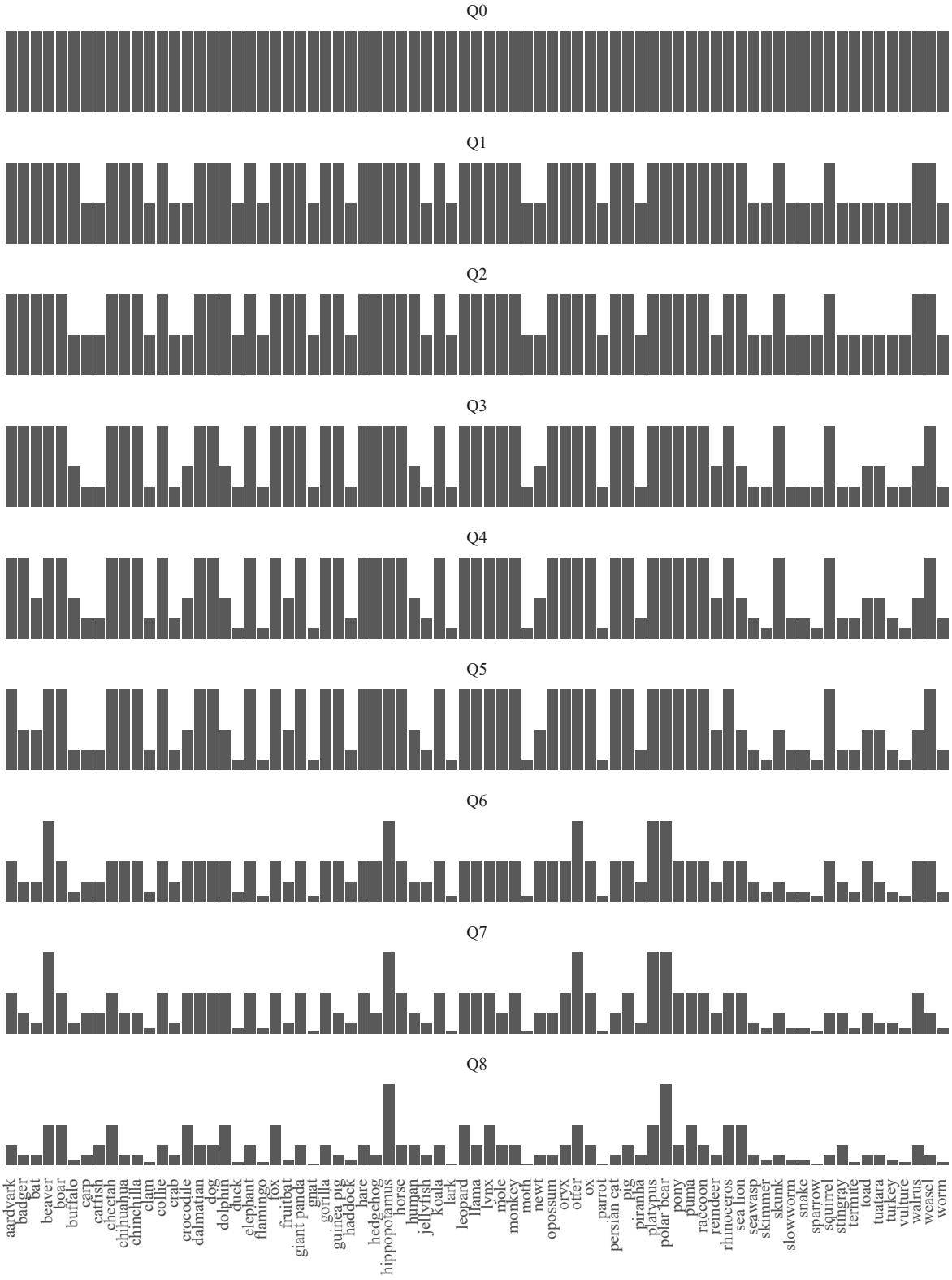

Figure 1: Updating of probability scores across nine questions (only shown for a subset of 75 animals)

Technically, the distribution that we maintain over animals is not a probability, since it does not sum to one; rather, it is a probability score that keeps track of how (in)compatible each animal is with the answers that the user has provided so far. It can be thought of as bargain-basement Bayesian updating.

We initialise a uniform prior probability score of 20 (an arbitrary choice) across all animals at the beginning of the game and update it after each question based on the user's answers. Specifically, we divide the current probability for each animal in half (also an arbitrary choice) if it is incompatible with the answer the user just gave. Once the features have all been used up, we have a distribution of probability scores across animals that reflect how compatible they are with the user's answers. Figure 1 illustrates how this distribution changes over the course of a game.

The animals that are perfectly compatible with everything the user has entered have the same probability score that they started with; those that are incompatible with only one answer have half of the prior score, and so on. At the end of the game, the system guesses the animals in decreasing order of probability. For the example shown in Figure 1, it would guess `polar bear` and `hippopotamus` (the only ones with a probability score of 20), followed by the animals with a probability score of 10, and so on.

4.3 Template-based question generation

Now that we know which features or animals to ask about when, we come to the front-facing part of the system where questions about these entities are generated. The QG section of this project consists of two sub-parts, namely

1. preprocessing the Switchboard Dialog Act Corpus (SwDA; Jurafsky et al. 1997; Shriberg et al. 1998; Stolcke et al. 2000) to identify structures of polar questions,[5] and

2. feature-to-question generation.

We decided to work with the SwDA to be able to investigate the different structures of polar questions and to use those as a reference point for the questions we will generate.

[5]The code used for preprocessing can be found here: `https://github.com/epankratz/twenty-questions/blob/master/SWDA/swda_preprocessing.ipynb`.

Since the 20Q gameplay is based exclusively on the use of polar questions, which correspond to the tag `qy` in the SwDA, we extracted questions marked by this tag. Then, we analysed the POS tags in these sentences, since the question templates we will use are chosen based on the POS of the variable element. Table 3 shows some examples of the question from the SwDA, its POS tagging there, and the POS tags from spaCy, which we have chosen to work with because of their simplified structure.[6]

The next step in the preprocessing was to remove uninformative POS tags such as `PUNCT` (punctuation) and `SPACE` (blank spaces). Then, in order to investigate the general structure of polar questions, we concentrated on the first five POS tags in each sentence and counted the frequency of each POS tag sequence. We ended up with 1332 unique POS tag sequences, the five most frequent of which are shown in Table 4. The most frequent one, `(VERB, PRON, VERB, DET, NOUN)`, would for example correspond to a question like "Does it have a tail?".

In addition to the structures that we gathered from the SwDA, we chose to add one further twist and incorporate a subtype of polar questions: tag questions. These are interrogative constructions formed by a main clause (i.e. anchor) and a peripheral question tag (Tottie and Hoffmann, 2006; Bonsignori, 2007; Bawden, 2017), e.g.

$$\underbrace{\text{"You are happy,}}_{\text{anchor}}\ \underbrace{\text{aren't you?"}}_{\text{question tag}}.$$

In a discourse, tag questions can have particular pragmatic power (Tottie and Hoffmann, 2006), and we are using them to indicate bias on the part of the 20Q player. If the system chooses a feature, it might expect that the user's answer of whether their animal has that feature will be "yes" if most of the values for that feature are 1. In that case, it might ask, say, "Your animal is big, isn't it?" with a tag question, rather than the neutral "Is your animal big?". This would give the 20Q player a more human, playful voice, and introduce some variety to the question templates as a whole.

For each feature, we compute an extremeness value that symbolizes how out-of-balance the values for a given feature are. If the extremeness value exceeds a certain threshold (which we set

[6]We also tried performing NER on the feature names using spaCy, as an alternative to the POS tags, but this did not work on the out-of-context elements at all.

Text	SwDA POS	spaCy POS
Were you –	Were/VBD you/PRP –/:	`['VERB', 'PRON']`
Are you in Texas?/	Are/VBP you/PRP in/IN Texas/NNP ?/.	`['VERB', 'PRON', 'ADP', 'PROPN']`
I probably would —have done, D you know, just	I/PRP probably/RB would/MD have/VB done/VBN ,/...	`['PRON', 'ADV', 'VERB', 'VERB', 'VERB', 'NOUN'...]`
Are you a Vietnam veteran, Dudley?	Are/VBP you/PRP a/DT Vietnam/NNP veteran/NN ,/...	`['VERB', 'PRON', 'DET', 'PROPN', 'NOUN', 'PROP'...]`
Do you have family who were in the Vietnam War?	Do/VBP you/PRP have/VB family/NN who/WP were/V...	`['VERB', 'PRON', 'VERB', 'NOUN', 'PRON', 'VERB'...]`

Table 3: The first five rows of the SwDA dataframe, including the spaCy POS tags

Count	POS tag sequence
153	`(VERB, PRON, VERB, DET, NOUN)`
90	`(VERB, PRON)`
74	`(VERB, PRON, CCONJ, VERB, PRON)`
69	`(PROPN, VERB, PRON, VERB, DET)`
61	`(VERB, PRON, VERB)`

Table 4: Unique POS tag patterns and their frequency counts, extracted from the SwDA Corpus

at 0.65), a biased question is chosen over the non-biased version.

We initially considered implementing not only positive-anchored biased questions, as in (1), but also negative-anchored biased questions, as in (2).

(1) Your animal *is* big, isn't it?

(2) Your animal *doesn't have* feathers, does it?

This would have been possible, but we realized that if negative-anchored biased questions like the one in (2) are answered affirmatively, ambiguities arise. For example, it is unclear whether the answer "yes" to the question "Your animal doesn't have feathers, does it?" means that the animal indeed has feathers or that it does not. Being faced with negative-anchored biased questions like this might confuse the human player, which would negatively affect the functioning of our system. Therefore we opted to only include positive-anchored biased questions.

So, we now have question templates based on the POS tag of the variable element that are based on structures found in the SwDA, in both biased and unbiased versions (see Table 5). Notice that one question template, namely the biased question with a verb or auxiliary, requires that the verb or auxiliary be in the third person singular form in order make the question syntactically coherent. To take care of this, we made use of the `LemmInflect` library, a useful natural language processing library that can be used in combination with spaCy and that provides all inflectional forms for a given word.[7]

Now that the templates are in place, the QG proceeds according to the following steps.

1. Preprocessing and tokenising the feature names, i.e. changing underscores to spaces and tokenising the string in a format that can be POS-tagged.

2. POS-tagging of the first word in the feature string using spaCy's `pos_` method.

3. Generating a question by choosing one of the pre-defined question patterns depending on the extremeness value of the feature and the POS tag obtained in the previous step.

We did face some challenges with Step 2 during development. Specifically, spaCy's POS tagger didn't correctly tag all of our features, resulting in some ungrammatical questions. For instance, it would tag `nocturnal` only as a proper noun, for some reason, and not an adjective. We also tried WordNet's POS tagger (Fellbaum, 2010), but it still had poor accuracy with our feature names. Presumably, these problems can be attributed to the fact that we used out-of-context tagging.

We ultimately decided that the best solution would be to simply rename the mis-tagged features (e.g. changing `nocturnal` to `active_mainly_at_night`, which would be correctly tagged as an adjective, giving "Is your animal active mainly at night?").

[7] `https://github.com/bjascob/LemmInflect`; accessed 14.08.2020.

POS tag	Unbiased	Biased
VERB (+ed), ADJ, ADV	Is your animal {feat}?	Your animal is {feat}, isn't it?
NOUN (pl), DET, NUM	Does your animal have {feat}?	Your animal has {feat}, doesn't it?
NOUN (sg)	Is your animal a/an {feat}?	Your animal is a/an {feat}, isn't it?
VERB, AUX	Does your animal {feat}?	Your animal {feat.3SG}, doesn't it?
ADP	Does your animal live {feat}?	Your animal lives {feat}, doesn't it?

Table 5: Question patterns based on POS tag of the first token in the feature name and bias status

4.4 Incorporating out-of-database objects

Before concluding the implementation section, we turn to our last challenge: successfully incorporating out-of-database objects into our knowledge base. We adopted a strategy inspired by the following quote from Rosch (1978, 29):

> The perceived world is not an unstructured total set of equiprobable cooccurring attributes. Rather, the material objects of the world are perceived to possess ... high correlational structure. That is, given a knower who perceives the complex attributes of feathers, fur, and wings, it is an empirical fact provided by the perceived world that wings co-occur with feathers more than with fur.

Following this logic, for animals that are not in our knowledge base, we can exploit the correlations between similar animals to expand our knowledge base and improve the system over time. Every time the 20Q player loses, the human player is asked to give the name of the animal they were thinking of. If said animal is not already in the knowledge base, a new entry will be created for it, and the features that were asked about will be populated with the user's input. The system can gather at most 20 features for each animal, since it can ask at most 20 questions, which means that it will need to fill in the 43 or more empty fields in the new entry itself. This is where the interpolation of unknown features comes in.

Our goal was to find the most highly correlated animal to the out-of-database target (given the user's input) and subsequently populate the empty fields with the values from the highly-correlated animal's entry in the knowledge base. For example, given *coyote* as an out-of-database object, one would expect it to have many shared features with an animal like *wolf*. Our aim in implementing similarity measures is to find the existing animal in the knowledge base that most closely matches the user's input and therefore is likely a similar animal.

When initializing the 20Q game, the human user is given the option of specifying which of two correlation methods is to be used to interpolate any out-of-database objects. The first correlation method takes Rosch's (1978) words to heart, tallying the total number of features that a pair of animals has in common. We called this method OURS. The second method, which we called CORR, computes a pairwise Pearson correlation coefficient between the out-of-database animal and each remaining animal. The performance of both of these methods is evaluated in Section 5.2 below.

Having discussed our strategy for handling cases in which the target animal is not in the knowledge base, we turn now to our strategy for games that were lost by the 20Q player but for which the target animal was, in fact, in the knowledge base. In these instances, we still add a new row for the animal to preserve the human player's answers, but instead of using a correlation method, we fill the empty cells with the values from the animal's existing row in the knowledge base. The result is two entries for the given animal in the knowledge base, one with the newly-acquired human judgments and another with more factual information.

As we discussed in class, 20Q is a game in which it's more interesting to note what the players *think* about animals, even if this deviates from what is actually true. In this way we are able to capture both the objective information and more subjective, human judgment. In order to prevent the system from guessing the same target animal more than once in a single game, we implemented a check in the gameplay code to ensure that each animal name could only be guessed one time (only the highest-probability instance of each animal is guessed).

Figure 2: Call graph of non-utility class methods in `TwentyQuestions`. Methods in (1) are for sampling features; (2) for asking users about features, getting user input, and taking the respective action; (3) for guessing animals; and (4) for the endgame and adding out-of-database animals

In sum, the call graph in Figure 2 provides a visual summary of how the (non-utility) methods in the `TwentyQuestions` class fit together.

5 Evaluation

We evaluated two aspects of the performance of our 20Q player. The first, as one might expect, was the win and loss rate of our system over time. The second was how well our player could interpolate from the existing knowledge base entries to fill in missing data for newly-added animals. We discuss these aspects in detail below.

5.1 Win/loss rate

Before we could calculate our system's win/loss rate, we first had to play enough games to have meaningful statistics. All group members played against our system with the same 24 target animals: twelve randomly-selected in-database target animals and twelve out-of-database animals. The intuition behind this choice was that the system would learn about the out-of-database animals from the first player's inputs and interpolate values for the features that weren't asked about in the game. For an out-of-database animal that had never been introduced before, the system would be able to gather information about features of that new animal, which could then be utilized by the interpolation measure to fill in the remaining feature values. For subsequent games, then, the system would have this new animal in its knowledge base and would have the opportunity to guess it correctly.

As expected, for the first two players who played with the out-of-database animals, the system could not guess them correctly. However, by the third and fourth plays for the out-of-database targets, the system was able to win a game for six of the twelve newly-introduced animals. Interestingly, even if the system managed to guess an out-of-database animal for one player, it often was unable to win again on the same animal for the next player. We speculate that this may be due to the interaction of our random question selection and our interpolation measures. It could be the case that the human player's answers are incompatible with the interpolated data, and thus the probability distribution would be biased away from the correct target animal. However, after a significant number of plays, we expect that the system would be able to gain enough information to improve its win rate for these items as well.

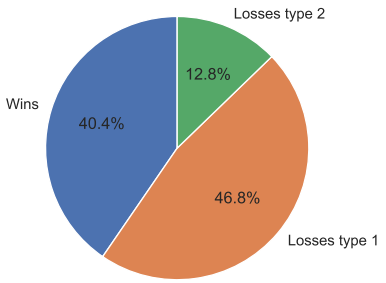

Figure 3: The win/loss rate after 94 games; Type 1 losses are losses on in-database items, and Type 2 losses are losses on out-of-database items

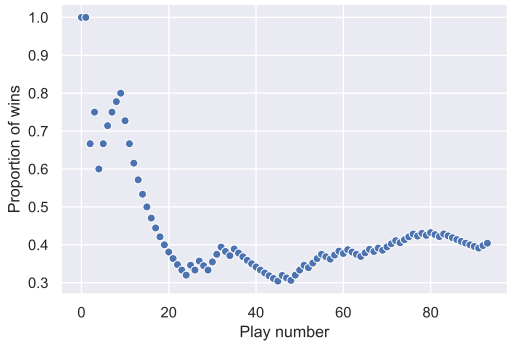

Figure 4: The proportion of wins to losses show an increasing tendency over time

At the time of writing, our model had logged 94 total plays. Although this is not a large number from a statistical standpoint, it is enough to allow us to do some analysis and assess how the model might be expected to improve over time. Of these 94 plays, 40.4% were wins and 59.6% were losses. The losses can be categorized into two types: those for which the animal was in the database but not guessed within 20 questions, and those due to out-of-database items. Of the losses, 78.5% were of Type 1, and 21.5% were of Type 2. Figure 3 illustrates these results.

We plotted the win/loss rate over time to see how it evolved with respect to the number of games played. Although our total number of plays is too low to give us an accurate picture of how the 20Q player would perform after many more plays, it seems reasonable to expect an upward trend (see Figure 4).

5.2 Quality of interpolated out-of-database items

Recall from Section 4.4 the two methods for filling in the unknown features of out-of-database animals: `OURS` counts the number of features that a pair of animals shares, while `CORR` computes a pairwise Pearson correlation coefficient between the features of a pair of animals. We compare these methods in two ways. First, we look at how comparable their similarity rankings of animals in the database are. Second, we consider how similar the values that each method interpolates are to a gold standard.

5.2.1 Similarity rankings

Our aim was to see how comparable the two methods are in the animals that they consider to be similar. To this end, we hand-selected seven pairs of animals that common sense would deem highly correlated. We then removed one first animal in each pair from the database and reintroduced it as an out-of-database item. Running both methods on the now-out-of-database animal, we generated the top five most similar animals.[8]

Two of the most interesting results can be found in Tables 6 and 7. For the pair (*dog*, *German shepherd*) in Table 6, the `OURS` method returned animals that match quite nicely with human intuition, performing far better than the `CORR` method, which returned animals that are very different both from *dog* and from each other.[9] We speculate that this may be due to the overly general nature of the object *dog*, which, given our decision to assign 1s to features for which one might answer with either yes or no, as we discussed in Section 3, shares features with a wide range of different animals.

However, turning to the pair (*chimpanzee*, *monkey*) in Table 7, we see that both correlation measures returned the same animals in the same order. This is promising and shows that both methods do a fair job at pinpointing similar animals in a way that is (usually) similar to human judgments.

5.2.2 Accuracy of interpolated features

To test the quality of the interpolation for the out-of-database items, we randomly selected twelve animals from the knowledge base to serve as a

[8]The code used for evaluating the similarity measures can be found here: `https://github.com/epankratz/twenty-questions/blob/master/code/Sim_comparison.ipynb`.

[9]In the Jupyter Notebook, *deer* is given as number 5 for `CORR`. *dog* it received the same score, but appears lower alphabetically, and it is substituted here for illustrative purposes.

	OURS		CORR	
1.	collie	61	cockroach	0.92
2.	dalmatian	59	chicken	0.84
3.	chihuahua	59	crow	0.84
4.	raccoon	59	raccoon	0.84
5.	dog	58	dog	0.83

Table 6: Correlation scores for *German shepherd* as the out-of-database item and *dog* as in-database reference

	OURS		CORR	
1.	gorilla	59	gorilla	0.81
2.	baboon	58	baboon	0.74
3.	human	57	human	0.73
4.	wallaby	57	wallaby	0.70
5.	baboon	57	baboon	0.70

Table 7: Correlation scores for *chimpanzee* as the out-of-database item and *monkey* as in-database reference (*monkey* was scored 56 by OURS and 0.69 by CORR)

gold standard. Then, one at a time, we removed each animal from the knowledge base and played a game in which the system was made to guess the (now-missing) animal. Of course, it will not be able to guess the animal, since it is no longer in the knowledge base. Our goal is to see how close the system's interpolations are to the true values for that animal, and whether the accuracy of interpolation varies greatly between the two similarity measures OURS and CORR.

For OURS, the mean accuracy rate of the interpolation across all animals was 82.1%, i.e. the system interpolated the correct value in 82.1% of cases. And for CORR, the mean accuracy rate of the interpolation was 77%.

For each game, a different number of features was asked about. Figure 5 suggests a weakly positive correlation between the number of features that were asked about and the accuracy of the interpolations, which makes sense. The more features that the user provides, the more information the system can use to make an educated guess about the remaining features.

6 Limitations of our system

There are several aspects of our system that could be improved upon in future work. Although we did implement some strategies to better handle games with so-called "noisy input", in which a user's answers did not exactly match the entry for the target animal in our knowledge base (for instance, accepting "unknown" as an answer, ranking features by their relative objectivity, and the probability distribution over all animals, as discussed above), our model's performance in such cases still has room for improvement.

Our feature selection method is such that the game may not be asking questions that will lead to the correct animal being guessed. Because the feature selection method chooses the best feature for the remaining partition of the knowledge base, if the target animal is not in that partition (i.e. if the user's input was incompatible with the knowledge base's representation of that animal), the system's questions are biased away from the target animal. The probability score distribution across animals is a strategy we implemented to address this problem, but it is still less likely that the correct animal will be guessed.

Another limitation of our system, as alluded to in Section 2, is the set of answer inputs available to the human player. While the current iteration of our 20Q game only allows for inputs of 0, 1, and 2 for "no", "yes", and "unknown", respectively, an avenue for future work would be to incorporate a wider range of answer choices, as in Burgener (2006)'s implementation. Adding inputs that are less black and white than this binary split (for instance, "I think so", "I don't think so", "sometimes", or "it's possible") would make the game easier to play and positively impact the probability distribution, perhaps leading to a higher win rate.

Admittedly, our feature selection system does not learn to ask better questions over time; the method of choosing the best feature to ask about at a given time does not follow a strategy that extends beyond each individual turn. We chose to focus our efforts on other parts of the project instead, since the current strategy already works fairly well. If we were to develop the system further, though, this would certainly be a point of improvement, e.g. by learning from previous games which sequences of questions tend to perform well. This approach would also potentially rectify another limitation of our system, namely that questions that should be mutually exclusive: if a user answers "yes" to the question "Does it have four legs?", the system need not subsequently ask if the animal has two legs.

Another potential point of improvement concerns the questions we generated. For future

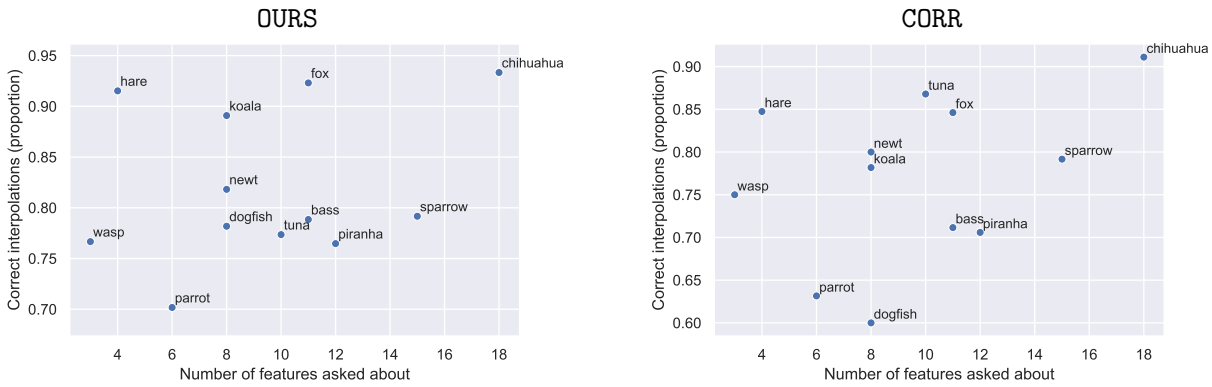

Figure 5: For both similarity measures, the proportion of correctly interpolated features shows a weakly positive correlation with the number of features the system asked about

projects, it would be interesting to perform POS-tagging in combination with NER on in-context elements in consonance with Mandasari (2019), which seems like a promising approach that would yield correctly-analyzed named entities that could be fitted into question templates without further preprocessing. We would also be interested to try a rule-based approach for QG, by developing a question template composed of syntactic categories to be filled later on, or to work with a more complex linguistic architecture.

Recall also from Section 3 that instances where the feature could be ambiguous, such as those related to color and size, were marked with 1 to capture the fact that the given feature *could* be said to be true for the target. Future iterations might modify the question template for this subset of features to reflect this sense of possibility or uncertainty, e.g. to something along the lines of "Could your animal be black?".

Finally, our interpolation measures could be more sophisticated. While our measures perform relatively well, as shown in Section 4.4, our system could be improved by adopting an interpolation method that could take into account feature data for more than one animal. Another approach could focus more on the features themselves than the animal, giving relative likelihoods of one feature given another.

Furthermore, since the interpolation tends to match the gold standard more closely when more questions have been asked, we could engineer the system to recognise that it will probably not guess the user's animal and, in that case, to ask as many probing questions as possible, to get more information that will improve the quality of its guessing.

7 Conclusion

Our fairly humble 20Q system may not hold its own against the top players on the market, but in working on this project we have taken on tasks that are both engineering-oriented and linguistic in nature. This has taught us a lot about question generation, manipulation of knowledge bases according to the provided answers, and (certainly not least) how to work in a team.

References

Rachel Bawden. 2017. Machine translation, itsa question of style, innit? The case of English tag questions. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2507–2512.

Christopher Bishop. 2006. *Pattern Recognition And Machine Learning*. Springer.

Veronica Bonsignori. 2007. *Tag Questions in English. A Syntactic, Pragmatic and Prosodic Account*. Ph.D. thesis, University of Pisa.

Robin Burgener. 2006. Artificial neural network guessing method and game. US Patent App. 11/102,105.

Alvin Dey, Harsh Kumar Jain, Vikash Kumar Pandey, and Tanmoy Chakraborty. 2019. All it takes is 20 Questions!: A knowledge graph based approach. *arXiv preprint arXiv:1911.05161*.

Alexander R Fabbri, Patrick Ng, Zhiguo Wang, Ramesh Nallapati, and Bing Xiang. 2020. Template-based question generation from retrieved sentences for improved unsupervised question answering. *arXiv preprint arXiv:2004.11892*.

C. Fellbaum. 2010. WordNet. In *Theory and applications of ontology: computer applications*, pages 231–243. Springer, Dordrecht.

Fréderic Godin, Anjishnu Kumar, and Arpit Mittal. 2019. Learning when not to answer: A ternary reward structure for reinforcement learning based question answering. In *Proceedings of NAACL-HLT 2019*, pages 122–129.

Huang Hu, Xianchao Wu, Bingfeng Luo, Chongyang Tao, Can Xu, Wei Wu, and Zhan Chen. 2018. Playing 20 question game with policy-based reinforcement learning. *arXiv preprint arXiv:1808.07645*.

Rodney Huddleston. 1994. The contrast between interrogatives and questions. *Journal of Linguistics*, pages 411–439.

Daniel Jurafsky, Elizabeth Shriberg, and Debra Biasca. 1997. Switchboard SWBD-DAMSL shallow-discourse-function annotation coders manual, draft 13. Technical Report 97-02, University of Colorado, Boulder Institute of Cognitive Science, Boulder, CO.

Payal Khullar, Konigari Rachna, Mukul Hase, and Manish Shrivastava. 2018. Automatic question generation using relative pronouns and adverbs. In *Proceedings of ACL 2018, Student Research Workshop*, pages 153–158.

Yani Mandasari. 2019. *Follow-up Question Generation*. Ph.D. thesis, University of Twente.

Kaksha Mhatre, Akshada Thube, Hemraj Mahadeshwar, and Avinash Shrivas. 2019. Question generation using NLP. *International Journal of Scientific Research & Engineering Trends*, 5(2):394–397.

Kevin P. Murphy. 2012. *Machine learning: A probabilistic perspective*. MIT Press, Cambridge/London.

J.R. Quinlan. 1986. Induction of decision trees. *Machine Learning*, 1:81–106.

Sathish Reddy, Dinesh Raghu, Mitesh M. Khapra, and Sachindra Joshi. 2017. Generating natural language question-answer pairs from a knowledge graph using a RNN based question generation model. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 376–385, Valencia, Spain. Association for Computational Linguistics.

Eleanor Rosch. 1978. Principles of categorization. In Eleanor Rosch and Barbara B. Lloyd, editors, *Cognition and categorization*, pages 27–48. Lawrence Erlbaum, Hillsdale, NJ.

Elizabeth Shriberg, Rebecca Bates, Paul Taylor, Andreas Stolcke, Daniel Jurafsky, Klaus Ries, Noah Coccaro, Rachel Martin, Marie Meteer, and Carol Van Ess-Dykema. 1998. Can prosody aid the automatic classification of dialog acts in conversational speech? *Language and Speech*, 41(3–4):439–487.

Andreas Stolcke, Klaus Ries, Noah Coccaro, Elizabeth Shriberg, Rebecca Bates, Daniel Jurafsky, Paul Taylor, Rachel Martin, Marie Meteer, and Carol Van Ess-Dykema. 2000. Dialogue act modeling for automatic tagging and recognition of conversational speech. *Computational Linguistics*, 26(3):339–371.

Alessandro Tonin, Niels Birbaumer, and Ujwal Chaudhary. 2018. A 20-questions-based binary spelling interface for communication systems. *Brain sciences*, 8(7):126.

Gunnel Tottie and Sebastian Hoffmann. 2006. Tag questions in british and american english. *Journal of English Linguistics*, 34(4):283311.

Yongqin Xian, Bernt Schiele, and Zeynep Akata. 2017. Zero-shot learning-the good, the bad and the ugly. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4582–4591.

Jacob Zerr. 2014. Question generation using part of speech information. *Final Report for REU Program at UCCS*.

A Individual contributions

All group members were in regular, active communication about ideas and directions in which to take the project. It was a fully collaborative effort throughout. We summarise here how we divided the workload between the four of us.

Wellesley Literature work, prepared and held the in-class presentation, created the knowledge base, wrote and proofread sections of the project plan and the final paper.

Anna Literature work, implemented the question generation, wrote sections of the project plan and the final paper.

Rodrigo Implemented the handling of out-of-database items, evaluated performance of the system and handling of out-of-database items with `CORR`.

Elizabeth Implemented the `TwentyQuestions` class and its core methods, wrote and proofread sections of the project plan and the final paper, evaluated handling of out-of-database items with `OURS`, created graphics.