Towards Automatically Answering Video Game F.A.Q.'s

Michael Coulombe

December 13, 2015

Abstract

A large portion of video game discussion online is initiated by players who are stuck and ask a forum for guidance. While this is effective for most players, the time taken waiting for a response is usually enough to force the player to stop playing, and as multiple players ask similar questions over time, it can be a waste of time for the experienced users to repeat their answer and to shift through the "polluted" forum to find more advanced discussions.

An alternate solution is to read a text walkthrough of the game, but searching the documents requires long linear scans or using "Ctrl-F" to locate exact word matches. NLP techniques can be used to marry the two solutions by taking in natural language questions and using the walkthroughs as a knowledge corpus to provide proposed answers to users and reduce forum pollution.

In exploring this problem, I used a dataset of 8.4 million word-units worth of walkthroughs and question-answer pairs collected from `http://www.gamefaqs.com/` from eight games of varying genre and data amounts. I compare to a basic, baseline "Ctrl-F" algorithm which takes question Q and weighs substring responses R by estimating $\sum_{w \in R \cap Q} P(w \in A \mid w \in Q)$. I improved upon the baseline with a "Segment-F" algorithm which first segments the walkthroughs (which are mostly implicitly-structured ASCII text) using a variant of a convolutional neural network trained by a semi-manual supervision method inspired by Prorogued Programming [ABS12].

The algorithms were evaluated by creating a fake walkthrough by concatenating the known answers and testing how often the question's true answer was chosen. "Ctrl-F" is surprisingly effective given its simplicity, achieving 22.61% average correct, but it was beat by "Segment-F" with 25.93% correct and extreme improvement on 3 games. These results demonstrate the promise of using automation to expediate the help process and reduce unnecessary forum posts.

1 Introduction

1.1 Background

Video games, while often fun and challenging, can frustrate a player who is stuck and does not know how to progress through a certain point in the game. To help them, game developers publish official strategy guides and experienced players write text walkthroughs hosted online, both of which aim to provide sufficient detail of the mechanics, strategy, and steps of the game to allow a player to solve their problem. Another common way players get help is to ask questions on community forums to get answers directly from the experienced players. Both of these methods of assistance are effective, but both have a unique set of issues.

When a player looks to a walkthrough for help, they are faced with the task of searching a long, not-explicitly-structured document for the section corresponding to their current game-state, so they can learn the next steps to take. This search is generally done by using the browser's (or the website's) exact string search function, which I will denote as the "Control-F" method. This method is only effective if the user knows what keyword to search for which will most likely identify their game-state, and if the author has written the walkthrough to be clear and easy to navigate.

When a player looks to a forum for help, the player may have an easier time because they may query experienced users using natural language questions rather than keywords. However, while they wait for a response, they may have to stop playing for an indeterminate amount of time, which can vary a lot depend on how big the community is. From the other perspective, a flood of new users asking the same questions over and over can bother the experienced users and dilute the forum of more in-depth conversation.

Given that walkthroughs are an pre-existing, mostly-complete knowledge base of the game in question, a natural, natural language processing solution arises: to use walkthroughs as a corpus to automatically answer these questions. This allows for users to easily ask questions and quickly get answers while also relieving the experienced users.

1.2 Problem Statement

The corpus I used was extracted from `www.gamefaqs.com`. The CBSi Terms of Use governing the site state that one may "access and view the Content for personal, non-commercial purposes," so I believe using the public content to train this system for a class project is an acceptable use, although publishing it as a paper may require written permission. To test on a variety of genres and data amounts, I investigated the following games:

Game	# Walkthroughs	# Questions
The Elder Scrolls V: Skyrim	24	395
Fallout: New Vegas	24	152
Minecraft	6	401
Fallout 3	37	355
Grand Theft Auto V	14	90
World of Warcraft	30	488
Super Smash Bros Melee	91	256
The Legend of Zelda: Skyward Sword	17	473
Total (MB of text in memory)	243 (35.974 MB)	2610 (2.619 MB)

For a given game, walkthroughs were taken from the "FAQs" tab and training question-answers pairs were extracted from the "Answers" tabs. Some questions have multiple posted answers. Given walkthroughs $W_1, ..., W_k$ and Q&A pairs $(Q_1, \mathcal{A}_1)$, ... , $(Q_n, \mathcal{A}_n)$, the goal is to produce a function which takes a question Q_i and returns the substring of some W_j which best "contains" some $A_{i,j} \in \mathcal{A}_i$ in some reasonable sense. The goal is to maximize how well the response location corresponds to the given answer, in order to answer unseen questions.

2 Ctrl-F Baseline

Inspired by the common, manual walkthrough-search method, Ctrl-F attempts to be a very basic but flexible keyword search algorithm. This algorithm only trains on the question-answer pairs for the games, and learns just the following parameter via a maximum likelihood estimate:

$$\theta_w = P(w \in A^* \mid w \in Q) \approx \frac{\sum_{Q,\mathcal{A}} [w \in Q \wedge \exists A \in \mathcal{A} : w \in A]}{\sum_{Q,\mathcal{A}} [w \in Q]} \tag{1}$$

Given a word w in a question, θ_w tries to capture the likelihood that it is also a word in the correct answer to the question, thus the word should be considered a good search keyword. To find a response to a question using the walkthroughs, the following algorithm is used:

1. Determine Q = the set of words in the question.
2. For each walkthrough W:
    3. For every contiguous range R of `range_len` words in W:
        4. Calculate $\textsc{score}(R) = \sum_{w \in R \cap Q} \theta_w$
5. Return top 5 scoring R

`range_len` is a hyperparameter which determines the length of the response in words (as defined by Python's `unicode.split` method). If a response is too short then it cannot contain much information in itself and will miss long-distance co-occurrences, but if it is too long then the imprecise nature of this model may skew towards common words rather than keywords. I started with `range_len` = 20 as a reasonable but arbitrary length, but other values were explored as well.

3 Segment-F Algorithm

3.1 Segmentation

One issue with the Ctrl-F baseline is the existence of `range_len` as a hyperparameter. To solve this problem, consider that a text walkthrough is not simply a list of words but a implicitly-structured document with headers, paragraphs, tables, and lists. Because all of the structure is encoded with human-readable but non-standard ASCII art, the problem can be simplified into that of segmenting the lines (typically up to 80 characters) of the walkthrough into semantically-related groups which are inferred by their character patterns.

The segmentation is performed used a discrete variant of a convolutional neural network. Given a window of four contiguous lines, the kernel takes a pair of adjacent columns, classifies each character as {letter, digit, punctuation, whitespace, other}, then returns a learned weight for the observed arrangement. The average weight of every kernel operation is lastly clamped into [0,1] then compared to $\frac{1}{2}$ to determine whether or not there should be a separator between the top two and bottom two lines of the window.

One obvious missing factor in the machine-learning of a segmentation algorithm is the lack of ground truth annotations to train on. In place of annotations, an existing classifier was used: myself. This was inspired by Prorogued Programming [ABS12], a programming language paradigm in which unimplemented functions may be marked as "prorogue" to indicate that a call will query the user to supply the return value. To make the process practical, training consisted of uniformly random initialization then sampling only 30 windows per walkthrough per game (one pass) and only querying myself for the ground truth if the CNN prediction was in [0.1, 0.9], otherwise the prediction was assumed correct.

In the end, just a small percentage of possible user queries were made and qualitative analysis of the CNN segmentation shows that it does a great job given the fact that each walkthrough is (mostly) written by a different author with a different style and given little possibility of over-fitting.

3.2 Model

After segmentation, the Segment-F also extends the modeling of the texts beyond θ_w 1 . θ^p_w is used to identify words which do not appear in many answers except the correct answer.

$$\theta^p_w = |\{i = j \mid w \in Q_i \wedge \exists A_{j,\ell} \in \mathcal{A}_j : w \in A_{j,\ell}\}| \tag{2}$$

Given a segment S of walkthrough W_i, θ^S_w intends to capture the distribution of words within W_i by using a unigram model of W_i as a whole and unigram models of each segment.

$$\begin{aligned} \theta^S_w = P(S|w) &= P(w|S) \times P(S) \div P(w) \\ &\approx \frac{|\{w = w_j | w_j \in S\}|}{|S|} \times \frac{|S|}{|W_i|} \div \frac{|\{w = w_j | w_j \in W\}|}{|W_i|} \\ &= \frac{|\{w = w_j | w_j \in S\}|}{|\{w = w_j | w_j \in W\}|} \end{aligned} \tag{3}$$

To reduce the influence of extremely frequent words, a Tf-Idf [1] transformation was applied to θ^p_w create $\hat{\theta}^p_w$ and applied to the unigram models to create $\hat{\theta}^S_w$. Additionally, segments, questions, and answers are divided into words using `nltk.sent_tokenize` and `nltk.word_tokenize`. The final question-answering algorithm of Segment-F combines all of these parameters as such:

1. Determine Q = the set of words in the question.
2. For each segmented walkthrough W:
    3. For every segment S of W:
        4. Calculate $\textsc{score}(R) = \sum_{w \in Q} \hat{\theta}^S_w + \hat{\theta}^p_w + [w \in S] \times \theta_w$
5. Return top 5 scoring S

4 Evaluation

As with segmentation, evaluating the two models by finding answers in walkthroughs suffers from the lack of ground-truth, but, unlike segmentation, it would be a conflict of interest to judge answers dynamically with user queries. Instead, the evaluation proceeds by splitting the Q&A data into a training set and eval set, training on the former, and creating a fake walkthrough document by concatenating the answers in the latter. Each algorithm is judged by the percentage of questions from the eval set that are responded to by answers which positionally overlap one of the actual answers by at least one word. For Segment-F, answers are in their own segments, so it will never return a partial answer.

[1] Term-Frequency times Inverse Document-Frequency, via: `sklearn.feature_extraction.text.TfidfTransformer`

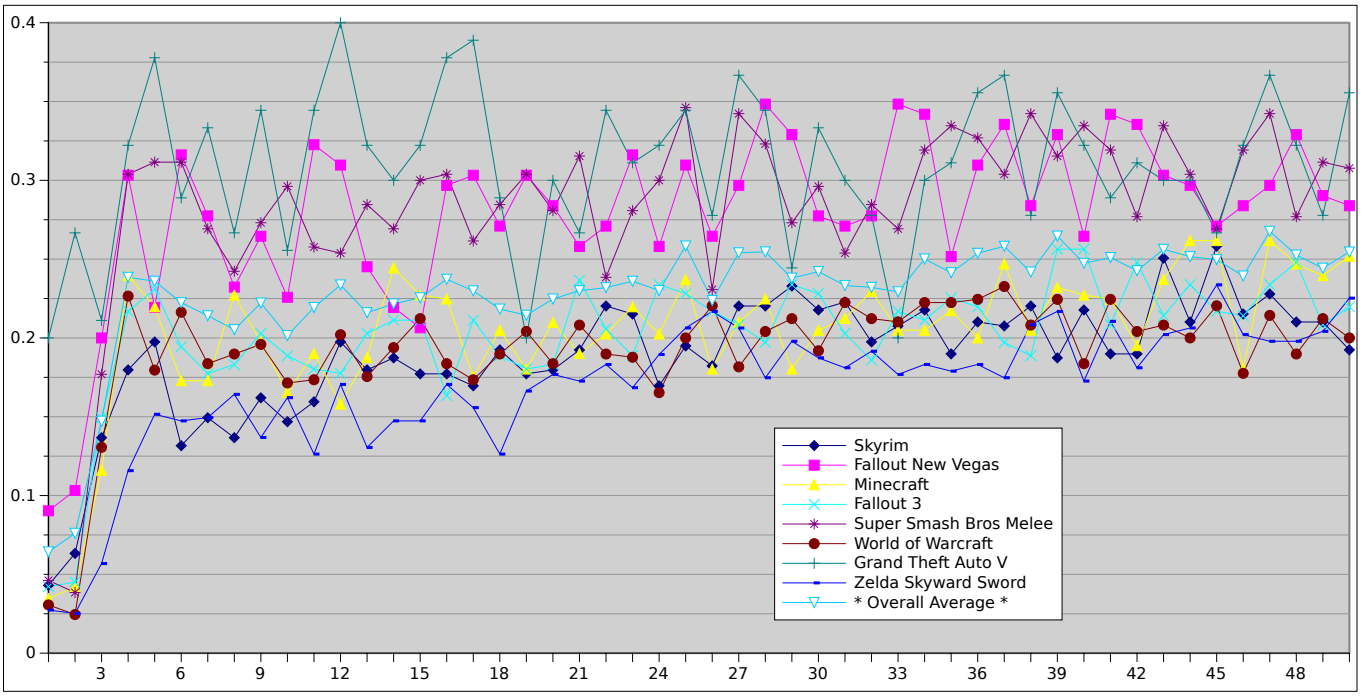

Figure 1: Ctrl-F evaluation results varying `range_len` in [1,50]. 5 trials with a 80%-20% training-eval split.

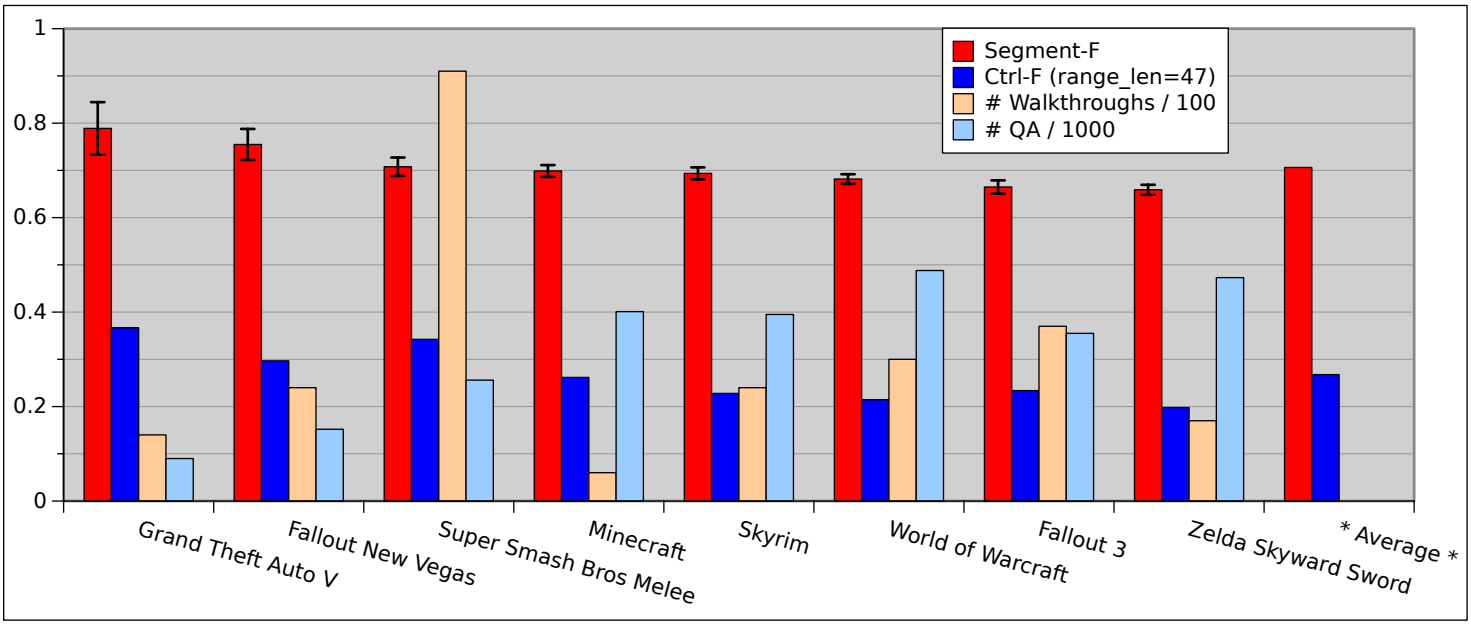

Figure 2: Segment-F evaluation results, compared to best Ctrl-F results and data set sizes. Error bars on Segment-F show the precision, the change if there was one more correct or incorrect answer on average. 5 trials with a 80%-20% training-eval split.

Figure 1 shows the results of the Ctrl-F baseline. The first clear pattern is the difference in accuracy depending on the game: GTAV, New Vegas, and Melee are consistently and similarly above the average (∼30%) while the other five are consistently and similarly below the average (∼20%). Interestingly, the higher-accuracy games also happen to be the three games with the fewest number of Q&A pairs, and the overall lowest-performing game, Skyward Sword, has the second-largest number of questions.

The next clear pattern is that the accuracy rapidly reaches over 20% at just `range_len` = 4, then slowly converges to around 25% up to `range_len` = 50. This seems to support the greater importance of small sets of keywords in a row to signal the answer to a questions as opposed to longer-range co-occurrences.

Figure 2 shows the significant improvement of Segment-F in comparison to Ctrl-F. [2] With an average accuracy of 70.62%, the new model improves by 43.85% over the best observed Ctrl-F average of 26.77% when `range_len` = 47. By training the model only partially, I found that the primary improvement in accuracy comes from $\hat{\theta}_w^S$ rather than $\hat{\theta}_w^p$, which is strong evidence that grouping words into semantically-related segments is a powerful tool for understanding the walkthrough and answering questions beyond simple keyword search.

It is notable that while the accuracy of Ctrl-F was somewhat negatively correlated with the number of Q&A pairs in the data, the additional parameters of Segment-F allows it to be more uniformly accurate across different games, having a standard deviation of 4.47% rather than 6.18% for optimal Ctrl-F (other values of `range_len` may differ). It is important to note that the precision of the average is dependent on the number of Q&A pairs per game, and that both models appear to do better when there are fewer questions, so those differences between games may be explainable by the low precision rather than by the nature of the games themselves.

5 Conclusion

The evaluation experiment's results demonstrate the benefit of intelligently segmenting the walkthroughs in order to apply more sophisticated models, to eliminate hyperparameters, and to respond to questions with full coherent answers. The performance of the Segment-F algorithm shows that it is practical to augment a video game walkthrough and forum host with a question-answer system to reduce the volume of unnecessary traffic to the forums and to allow players to easily get answers to most of their basic game questions using existing resources.

Code repository: `https://github.mit.edu/mcoulomb/6.864-nlp-project`

[2] These new numbers are different than the slides due to a bug in computing θ_w^S in the initial testing which severely reduced the accuracy of the model. Slides have been marked to mention this.

References

[ABS12] Mehrdad Afshari, Earl T. Barr, and Zhendong Su. Liberating the programmer with prorogued programming. In *Proceedings of the ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, Onward! 2012, pages 11–26, New York, NY, USA, 2012. ACM.