Automatic Text Summarization using Reinforcement Learning

Ali Emami
260412586
ali.emami@mail.mcgill.ca

Yue Dong
260408330
yue.dong2@mail.mcgill.ca

***Abstract*—In this project, we tried to reproduce the paper [1] to perform automatic summarization with reinforcement learning. The empirical results show that RL can be used in automatic text summarization. In this specific problem of sentence selection searching, TD(λ) control (a little bit different algorithm than SARSA(λ) is more effective in learning than actor-critic with eligibility trace. We believe this is due to the large action space and the fact that value function is easier to approximate than the policy.**

I. Introduction

Automatic text summarization is amongst the most popular tasks in Natural Language Processing (NLP). Its aim is to automatically generate short and structured summaries for single or multiple topic-related documents. Given a significant upsurge of publicly available information, the original content of many, and potentially lengthy, documents can be understood quickly and effectively by readers by way of these summaries, aiding both in the retrieval and organization of information.

Two types of summarization techniques have been thus-far explored: extraction and abstraction-based summarization. In extraction-based techniques, selected textual units (either characters, words, clauses, or sentences) are directly extracted from the original documents, and, without modification, are pasted together in the attempt of producing a coherent summary. On the other hand, abstraction-based techniques paraphrase sections of the source document and can, in general, condense a text more strongly than extraction. These techniques are naturally much more difficult to develop, requiring the use of natural language generation technology (which is, itself, at its outset), as well as being difficult to evaluate, since it no longer guarantees the linguistic quality of the produced summary. Accordingly, extractive techniques remain the most popular and well-known approaches to the automatic summarization task.

One of the most common extractive approaches is maximal marginal relevance (MMR), which scores each textual unit by the MMR criteria in order to greedily extract the units with the highest score to generate the summary [2]. A crucial draw-back of this technique is that it fails to evaluate the quality of the summary as a whole - the inclusion of certain textual units may render others, despite having a high score, as either redundant or ill-timed. Accordingly, global inference algorithms have in recent years gained popularity [3]. By formulating the problem as an integer linear program (ILP) in order to optimize a score, these approaches allow for a more holistic consideration of the goodness of a summary. On the other hand, ILP is well-known to be non-deterministic polynomial time hard (NP-hard), and in consequence presented a problem of efficiency.

These impediments have inspired recent attempts from the Machine Learning (ML) community. In the following paper, we seek to re-implement the learner from Ryang and Abekawa [1] and develop a reinforcement learning (RL) framework largely inspired by theirs, further exploring and adjusting reward, state, and action spaces. In addition, we will test various learners including actor-critic methods, surveying the performance difference of these learners and framework settings.

our contributions could be summarized as follows:

- Since there is no public available code for paper [1]. We code the algorithm (TD(λ) control with linear function approximation) by ourselves to reproduce the results of the paper. This includes implementing the score function [1], Features, Summarization environment, and the TD(λ) control method.
- Paper [1] proposed TD(λ) control method with linear function approximation to solve summarization [2]. However, they fixed their $\lambda = 1$ throughout the whole learning process. From the RL class, we know this is usually not a good idea due to the high

[1]The rewards and final return of our RL algorithms are based on this score function.

[2]The algorithm proposed, called Automatic Summarization using Reinforcement Learning (ASRL), looks like SARSA(λ). However, in SARSA(λ) we approximate and update $Q(s, a)$ directly and in ASRL, we use linear function approximation to estimate and update $V(s)$.

variance of Monte Carlo methods, we properly redid the experiments with different λ.

- We implemented policy gradient based algorithm – actor-critic algorithm with linear function approximation for the tasks of summarization and compared the results with TD(λ) control. We conclude that in this specific environment, actor critic is not performing better than TD(λ) control which we attrbute to the hypothese that the value function is easier to approxiate than the policy in this environment.

II. Motivation and Previous Work

Previous techniques such as the one considered by Ryang and Abekawa [1] have allowed for a natural and simple formulation of the problem in the reinforcement learning (RL) framework and have resulted in a reinforcement learner, specifically TD(λ), to generate summaries. Subsequently, the adjustment of the reward functions and feature sets in their framework, as well as using alternate RL algorithms has been further explored, showing some improvement of the summarization quality according to similarity metric calculations (with respect to gold-standard human summaries) used in ROUGE [4].

The learning algorithms considered in these previous attempts have included TD (λ) control, SARSA, and approximate policy iteration. While, according to both findings, the former algorithm seemed most promising during evaluation, nothing to our knowledge has been mentioned about exploring various settings of λ. In fact, for both Ryang and Abekawa [1] and Rioux et al.s TD(λ) learners [4], λ was fixed at 1, rendering the algorithms essentially as a Monte Carlo policy evaluation. As it turns out, intermediate values of λ may in fact allow for more rapid learning given few episodes, especially for high-variance episodes [5]. Furthermore, recently popular policy gradient methods have remained to be considered for which many problems that mar the previous RL approaches have been resolved.

III. Problem Formulation

A. Formulation of Extractive Approach

The given document/s in the extractive approach can be reduced to a set of textual units:

$$D = \{x_1, x_2, x_3, , x_n\} \tag{1}$$

Here, n denotes the size of the set, and x_i, the individual textual units that can be characters, words, clauses, or sentences. From now on, we will assume these textual units are sentences as this is the most popular setting for extractive techniques. Furthermore, we can call such a set D of related documents a document cluster.

We now define the score function, $score(S)$, subject length limitation K, and length function $L(S)$, for any subset S of D, that is, $S \subset D$. In fact, any such subset S is considered to be a summary of D. The goal of the summarization problem is to find an S that maximizes the given score function (while obeying the length limitation), typically capturing the trade-off between relevance and redundancy.

More formally, the problem can be formulated as:

$$\begin{aligned} S^* = argmax \quad & score(S) \\ s.t \quad & L(S^*) \leq K. \end{aligned}$$

One such score function considered by Abekawa and Rioux [1], [4] and ultimately the one we will adopt for our framework can be formulated as:

$$score(S) = \sum_{x_i \in S} \lambda_s Rel(x_i) - \sum_{x_i, x_j \in S, x<j} (1-\lambda_s) Red(x_i, x_j) \tag{2}$$

where

$$Rel(x_i) = Sim(D, x_i) + Pos(x_i)^{-1},$$

$$Red(x_i, x_j) = Sim(x_i, x_j).$$

Here, x_i and x_j are sentences with indexes i and j from the set of ordered sentences in the document cluster (top to bottom, single origin), λ_s the trade-off parameter between relevance $Rel(x_i)$ and redundancy $Red(x_i, x_j)$, for which $Sim(D, x_i)$ corresponds to the cosine similarity between the sentence and the document clusters set of sentences, $Sim(x_i, x_j)$ the cosine similarity between the two sentences, and $Pos(x_i)$ the index of the sentence with respect to the set of ordered sentences in the document cluster D.

The length limitations set by the Document Understand Conferences with respect to their datasets (that we will later be using) outline a limitation of 665 bytes per summary[3].

B. State Definition

The first key component of an RL framework is the state; that is, a formal description of the environment that can depend on both previous states and actions, ultimately used by the policy to generate future actions.

For our summarization task, a state can be considered, quite naturally, as a summary, S. Each state s can thus be represented

[3]During our initial experiments, we set the length to be three sentence for quicker learning

as a tuple of the summary S, history of actions, and termination status of the state $f \in \{0,1\}$, as such: $s = (S, A, f)$, where an initial "empty page" state s_0 is $(\emptyset, 0)$.

We assume a feature representation with dimension d of the state s, $\phi(s) \in R^d$, based only on features of the summary, $\phi'(S) \in R^{d-1}$:

$$\phi(s) = \begin{cases} (\phi'(S), 0)^T & (L(S) \leq K) \\ (0,1)^T & (L(S) > K) \end{cases} \tag{3}$$

Given that the feature representation of the state depends only on the summary and not the history of actions, $\phi'(s)$ should be designed in a way so as to generalize well, contributing to a reduced search space and provide efficient learning.

$\phi'(s)$ is a 104- dimensional vector defined as follows:

- Feature 1-100 - **Coverage of important words:** the elements of the top 100 words in the document cluster D in terms of $tf*idf$ for Ryang and Abekawa [1] and bigrams for Rioux et al. [4] with binary representation.
- Feature 101 - **Coverage ratio:** The total number of top 100 important words covered in the summary divided by 100.
- Feature 102 - **Redundancy ratio:** The total number of times a top-100 important word is occurs after first occurrence in the summary.
- Feature 103 - **Length ratio:** The number of words in the summary divided by the length limitation K (in our case, K=3 sentences or K=250 words).
- Feature 104 - **Position score:** This feature sums up (1/index) of each sentence in the summary with respect to their place in the original documents so as to yield a high score for sentences in the beginning of the documents.

C. Action

An action defines a transition from one state to a subsequent one. For our summarization task, these would include extracting sentences from the document cluster and inserting them into our summary.

Accordingly, our action space consists of $insert_i (1 \leq i \leq n)$ actions, that take sentence x_i from D and result in a transformation of S (originally), of the form $S_t \cup x_i$. In addition to these insertion actions, we define a $finish$ action that asserts the completion of the summary and renders the state terminal. These can be described by the following state-transition diagrams:

$$[S_t, A_t, 0] \xrightarrow{a_t = insert_i} [S_t \cup x_i, A_t \cup a_t, 0]$$
$$[S_t, A_t, 0] \xrightarrow{a_t = finish} [S_t, A_t \cup a_t, 1]$$
$$[S_t, A_t, 1] \xrightarrow{a_t} [S_t, A_t, 1]$$

Note that the purpose of the final transition is to emphasize that actions take no effect if the state is terminal.

D. Reward

The reward function provides the agent a reward that correlates to how good the executed action is. Accordingly, for the automatic summarization task, the reward function corresponds quite naturally to the score function defined earlier. In this sense, we expect that the rewards are to be delayed for summarization tasks, only distributing the reward of $score(S)$ for a summary S if that summary S is declared finished. Note that Rioux et al. adjusted Ryang and Abekawas $score$ function (particularly, their similarity function) to be based on the n-gram concurrence score metric, and the longest-common-subsequence recall metric contained within ROUGE [4] (accordingly, it is no surprise that they report increased ROUGE scores for their optimized learner).

Additionally, if the summary length is inappropriate and the action is $finish$, we expect the agent to receive a significant negative reward.

Formally,

$$\phi(s) = \begin{cases} score(S), & \text{if s is terminal} \\ -LargePenalty, & \text{if} \quad L(S) > K \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

Note that we neither reward nor penalize actions while the actions produce valid summaries and the summary isnt finished.

E. Value Function Approximation

We tried both TD(λ) control and actor-critic for the task of summarization. In either case, we will need to estimate the *state value function* with parameters $w \in \mathbb{R}^d$:

$$V(s) = w^T \phi(s). \tag{5}$$

1) TD(λ) control: Since the task we have is deterministic based on the action we choose at a particular state (so we know the transition probability), we could represent and estimate the *action value function* $Q(s,a)$ based on $V(s)$ directly (as in paper [1]):

$$Q(s,a) = r + \gamma V(s') \tag{6}$$

where performing action a at the state s will lead the state transit to s' with a probability 1 and the agent receive a reward r. The γ $(0 \leq \gamma \leq 1)$ is the discount rate.

With the action value function for each state s and action a, we could then define a policy taking into account the trade-off between exploring and exploiting, such as ϵ - greedy algorithm. Since our first goal is to reproduce the paper [1], we used the Boltzmann selection strategy as in [1]:

$$p(a|s;w,\tau) = \frac{e^{Q(s,a)/\tau}}{\sum_b e^{Q(s,b)/\tau}}, b \in A_s \tag{7}$$

where A_s is the action set at state s. Note that during learning, we decrease τ with more episodes. This cause the policy to be greedier which is desired since we want less and less exploration throughout the learning.

2) Actor-Critic with eligibility trace and policy approximation: In policy gradient learning, instead of using $Q(s,a)$ to choose the best action in a certain state, we select an action based on the *learned parametrized policy* $\pi(a|s,\theta)$:

$$\pi(a|s,\theta) = \frac{exp(h(s,a,\theta))}{\sum_b exp(h(s,b,\theta))}, b \in A_s \tag{8}$$

where the *preferences* $h(s,a,\theta)$ is represented by a linear function in features:

$$h(s,a,\theta) = \theta^T \phi(s,a).$$

The features $\phi(s,a)$ is constructed based on $\phi(s) \in \mathbb{R}^d$. Suppose there are n sentences in the document cluster for summarization, then our action set contains $n+1$ actions $\{insert_1, ..., insert_n, finish\}$. We therefor could define $\phi(s,a) \in \mathbb{R}^{d*(n+1)}$ as:

$$\phi(s,a) = \begin{cases} concat\big(\phi(s), \{0\}^{d*(n)}\big) & if \quad a = finish \\ concat\big(\{0\}^{d*(i)}, \phi(s), \{0\}^{d*(n-i)}\big) & if \quad a = insert_i \end{cases} \tag{9}$$

IV. Learning Algorithms

1) TD(λ) algorithm: The first algorithm we tried is based on paper [1]. It is a TD control method although the linear approximation estimate the state value function directly. The goal of this algorithm is to learn w (or θ in ASRL pseudocode), where $V(s)$ and $Q(s,a)$ can be both derived from as equation (5) and (6).

2) Actor-Critic: We also implement the standard actor-critic algorithm based on Sutton's book [5] where at time t,

$$\nabla_\theta log\pi(A_t|S_t,\theta) = \frac{\nabla_\theta \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)}$$

and with linear action preferences,

$$\nabla_\theta log\pi(A_t|S_t,\theta) = \phi(s,a) - \sum_{b \in A_s} \pi(b|s,\theta)\phi(s,b).$$

Algorithm 1 ASRL

Input: document $D = \{x_1, x_2, \cdots, x_n\}$,
score function $score(S)$

initialize $\theta = \mathbf{0}$
for $k = 1$ **to** N **do**
  $s \leftarrow (\emptyset, \emptyset, 0)$
    // initial state
  $e = \mathbf{0}$
  while s is not terminated **do**
    $a \sim p(a|s;\theta,\tau_k)$
      // selects action with current policy
    $(s', r) \leftarrow \text{execute}(s,a)$
      // observes next state and receive reward
    $\delta \leftarrow r + \gamma\theta^{\mathrm{T}}\phi(s') - \theta^{\mathrm{T}}\phi(s)$
      // calculates TD-error
    $e \leftarrow \gamma\lambda e + \phi(s)$
      // updates the eligibility trace
    $\theta \leftarrow \theta + \alpha_k \delta e$
      // learning with current learning rate
    $s \leftarrow s'$
  end while
end for
$s \leftarrow (\emptyset, \emptyset, 0)$
while s is not terminated **do**
  $a \leftarrow \max_a Q(s,a)$
    // selects action greedily
    with the learned policy
  $(s', r) \leftarrow \text{execute}(s,a)$
  $s \leftarrow s'$
end while
return the summary of s

Fig. 1: The pseudo-code of ASRL proposed in [1]

The goal is to learn optimal θ which decides the policy. Since the policy π (based on θ) is a soft policy, it will still exhibit some exploration when we want to use the algorithm for choosing the sentences for final summary. To do so, we also implement an greedy algorithm which selects action based on the hard policy π' derived from π:

$$maxP = max\big(\pi(a|s,\theta) \quad \forall a \in A_s\big)$$

$$\pi'(a|s,\theta) = \begin{cases} 1 & if \quad \pi(a|s,\theta) = maxP \\ 0 & otherwise \end{cases} \tag{10}$$

Note if there are multiple actions lead to $\pi(a|s,\theta) = maxP$, we randomly choose one.

V. Experiments

In our experiment, we coded all the algorithms by ourselves including the data preprocessing steps. We used sentences

as textual units and the goal is to choose sentences for the extractive summary. Each sentence and document were represented as a bag-of-words vector with tf*idf values, with stopwords removed and all tokens were stemmed.

We first used the document cluster with three documents for the prototype development. The document clusters are in appendix A. There are 15 sentences in this cluster. The rewards are given as equation (2) and (4). We set the summary length to be 3 sentences in our experiments.

Here are a few tricks we did to speed up our learning:

- *We remove an action from the action set when this action is selected.* During the experiments, we found that if we don't reduce the action set every time an action is selected, both algorithms will pick up the same actions frequently. Although the score function penalize the redundancy, but it does not offset the relevance increased by choosing the same sentences.
- *We forced both algorithms to choose action 0 as the forth action.* We found if we do not do so, most of the time, the algorithms will go over the length limit and not learn. Therefore, we forced the algorithms to stop before it goes over the limit and receive a penalty of -1. We used this trick to speed up the learning.

We compared the performance of TD(λ) control and actor-critic on this documents cluster. We also experimented different parameters for λ for different algorithms.

In both algorithms, we set the the number of episodes $N = 300$ and the discount factor $\gamma = 1$. The penalty was fixed to 1. In ASRL (TD(λ) control), we set up $\alpha_k = 0.001 \cdot 101/(100 + k^{1.1})$, and the temperature $\tau_k = 1.0 \cdot 0.987^{k-1}$ where k was the number of episodes run so far. In actor critic, we set the learning rate of $\alpha_k = 0.01 \cdot 101/(100 + k^{1.1})$ and $\beta_k = 0.01 \cdot 101/(100 + k^{1.1})$ to ensure that the value function is changing faster than the policy.

Ideally, we were planning to use DUC2004 task2 for our evaluation. There are 50 document clusters and each cluster has 10 documents. We first set the length limit to be 3 sentence. Then we set up the length limitation to be 665 bytes (which is approximately 200 words), which is used in the evaluation with ROUGE score of DUC2004.

We then train both algorithms on these clusters for 300 episodes. After, we generate the greedy summary based on the greedy policy (equation (10) for actor-critic or the greedy policy in ASRL). Then we evaluate both algorithms based on the ROUGE score computed with the golden standard summary.

However, at the time of this report is written, we were not able to produce meaningful result from DUC clusters because it takes our algorithms very long to run one episode with 3 selected sentences. For example, the cluster 50 contains over 300 sentences and it takes averagely 30 minutes to run one episode with 4 steps (selecting three sentences and finish). We believe there are rooms to modify our code to improve the speed and we were able to identify that the part of getting features for a sentence and calculating soft-max policy and preference for a large action set also contribute to the slow speed of running.

At the time this report is written, these algorithms are still running on our machine.

VI. Preliminary Results

The following table summarize our preliminary results based on the prototype documents cluster.

	ASRL	AC
score	3.6904	3.6904
run-time	497s	942s

TABLE I: Results from running ASRL and Actor Critic. The results are averaged after 3 runs with 300 episodes on each run. Both algorithms seem converged. However, the running time of actor critic is almost the twice of the running time of ASRL.

After running 300 episodes, both algorithm seem converged. When we print out the greedy summary, they both choose the first sentences of each document in the cluster. We think this is because that the three documents are very irrelevant and it is very hard to have a good cosine similarity score and therefore, both decided to exploit the position score (POS$(x_i)^{-1}$) because earlier sentences in a document are preferred.

To compare the learning speed of the two algorithms, we plotted the following graph based on one run.

From **??** and **??**, it is interesting to see that actor critic algorithm is actually learning slower than the ASRL (TD(λ) control). We believe this is because the value function is easy to approximate than the policy in this task. The action set is usually large due to the number of sentences in a document clusters and therefore, there are more parameters to approximate in policy approximation than value approximation.

To compare the performance with various λ in TD(λ) and actor critic algorithm with eligibility trace, we run each algorithms with 3 runs of 300 episodes with various λ. The resulting plot is averaged over 3 runs with a running mean of 10 episodes.

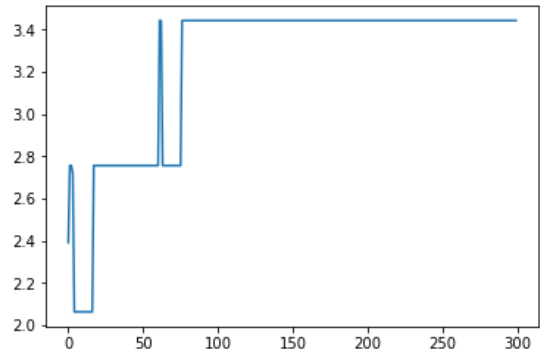

Fig. 2: ASRL (TD(λ) control) in one run with 300 episodes. The x-axis is the number of episodes and the y-axis is the greedy return based on the greedy selection algorithm. Within 100 episodes, ASRL seems already converged.

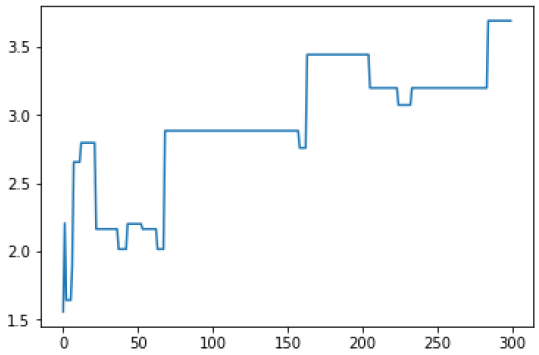

Fig. 3: Actor critic (AC) in one run with 300 episodes. The x-axis is the number of episodes and the y-axis is the greedy return based on the greedy selection algorithm. AC seems only converge after 200 episodes in this run.

VII. Conclusion and Discussion

In this project, we implemented the TD(λ) control method based on paper [1] for the task of automatic text summarization. We implemented all code from scratch including the score function, Features encoding, Summarization environment, and the TD(λ) control method.

In addition, we implemented the policy gradient method, actor critic with eligibility trace, in the hope of developing a better algorithm for the task. Actor critic has been used successfully (faster learner than SARSA(λ) from our experience in the environment of mountain car problem. However, it is not a better algorithm than TD(λ) control method in the present setting of extractive sentence selection summarization. We believe this is due to the deterministic environment and the large action set.

Although we were not able to get the result from the full

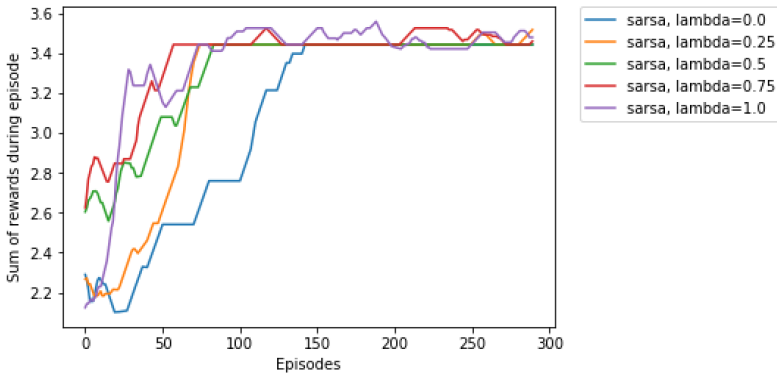

Fig. 4: ASRL (TD(λ) control) averaged over 3 runs with 300 episodes in each run. All λ seem converge at episode 300. Although we can see that TD(0) control converge slower than the other lambda. Surprisingly that TD(1) control which is basically a Monte Carlo method actually perform very well in the task of summarization.

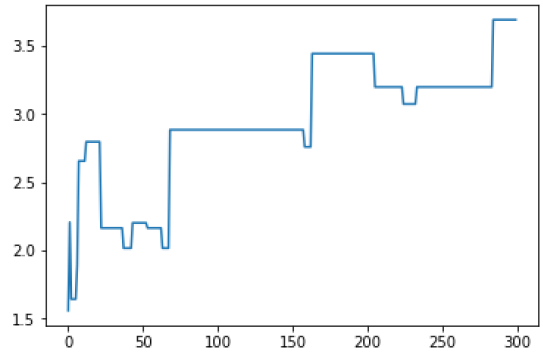

Fig. 5: Actor critic (AC) in one run with 300 episodes. The x-axis is the number of episodes and the y-axis is the greedy return based on the greedy selection algorithm. AC seems only converge after 200 episodes in this run.

experiment on the DUC2004 task2 dataset due to the time constraints. We are expecting to produce some meaning results after we improve the speed of our codes. The future works includes

Appendix A
Document Cluster Used for Prototype

document1 = "Python is a 2000 made-for-TV horror movie directed by Richard Clabaugh. The film features several cult favorite actors, including William Zabka of The Karate Kid fame, Wil Wheaton, Casper Van Dien, Jenny McCarthy, Keith Coogan, Robert Englund (best known for his role as Freddy Krueger in the A Nightmare on Elm Street series of films),

Dana Barron, David Bowe, and Sean Whalen. The film concerns a genetically engineered snake, a python, that escapes and unleashes itself on a small town. It includes the classic final girl scenario evident in films like Friday the 13th. It was filmed in Los Angeles, California and Malibu, California. Python was followed by two sequels: Python II (2002) and Boa vs. Python (2004), both also made-for-TV films."

document2 = "Python, from the Greek word, is a genus of nonvenomous pythons[2] found in Africa and Asia. Currently, 7 species are recognised. A member of this genus, P reticulatus, is among the longest snakes known."

document3 = "The Colt Python is a 357 Magnum caliber revolver formerly manufactured by Colt's Manufacturing Company of Hartford, Connecticut. It is sometimes referred to as a Combat Magnum.[1] It was first introduced in 1955, the same year a Smith & Wesson's M29 44 Magnum. The now discontinued Colt Python targeted the premium revolver market segment. Some firearm collectors and writers such as Jeff Cooper, Ian V Hogg, Chuck Hawks, Leroy Thompson, Renee Smeets and Martin Dougherty have described the Python as the finest production revolver ever made."

greedy selection ac:
(["the colt python is a 357 magnum caliber revolver formerly manufactured by colt's manufacturing company of hartford, connecticut", 'python, from the greek word, is a genus of nonvenomous pythons[2] found in africa and asia', 'python is a 2000 made-for-tv horror movie directed by richard clabaugh'], 3.6904930278752697)

documents = [document1, document2, document3] use section* for acknowledgement

ACKNOWLEDGMENT

The authors would like to thank...

REFERENCES

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.

REFERENCES

[1] S. Ryang and T. Abekawa, "Framework of automatic text summarization using reinforcement learning," in *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Association for Computational Linguistics, 2012, pp. 256–265.

[2] J. Carbonell and J. Goldstein, "The use of mmr, diversity-based reranking for reordering documents and producing summaries," in *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 1998, pp. 335–336.

[3] R. McDonald, "A study of global inference algorithms in multi-document summarization," in *European Conference on Information Retrieval*. Springer, 2007, pp. 557–564.

[4] C. Rioux, S. A. Hasan, and Y. Chali, "Fear the reaper: A system for automatic multi-document summarization with reinforcement learning." in *Proceedings of the 2014 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2012.

[5] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, vol. 1, no. 1.