Textual Data Augmentation for Efficient Active Learning on Tiny Datasets

Husam Quteineh[1*], **Spyridon Samothrakis**[2], **and Richard Sutcliffe**[3*]

[1]Business and Local Government Data Research Centre, School of CSEE, University of Essex
husam.quteineh@essex.ac.uk
[2]IADS, University of Essex
ssamot@essex.ac.uk
[3]IPMI Group, School of IST, Northwest University China
rsutcl@nwu.edu.cn

Abstract

In this paper we propose a novel data augmentation approach where guided outputs of a language generation model, e.g. GPT-2, when labeled, can improve the performance of text classifiers through an active learning process. We transform the data generation task into an optimization problem which maximizes the usefulness of the generated output, using Monte Carlo Tree Search (MCTS) as the optimization strategy and incorporating entropy as one of the optimization criteria. We test our approach against a Non-Guided Data Generation (NGDG) process that does not optimize for a reward function. Starting with a small set of data, our results show an increased performance with MCTS of 26% on the TREC-6 Questions dataset, and 10% on the Stanford Sentiment Treebank SST-2 dataset. Compared with NGDG, we are able to achieve increases of 3% and 5% on TREC-6 and SST-2.

1 Introduction

Active learning (AL) is a well applied approach in areas where unlabeled data is abundantly available, but labels are either scarce or costly to obtain (Settles, 2009). In AL, a classifier is improved upon through an iterative learning process; at each cycle, a subset of the original dataset with the most informative examples is selected to be labeled, typically by a human expert, before it is then added to the existing training data (Settles, 2009).

Previous active learning research on textual data to our knowledge has always assumed the availability of datasets containing large pools of unlabeled data. In cases where the available data is insufficient for active learning, the burden is transferred to the data collection process, where additional data must either be manually created, or collected from real world interactions. One strategy for creating data is to transform existing examples in certain ways in order to produce new data items and hence increase the size of the training dataset. This approach has been applied successfully in computer vision for example, by manipulating existing images while preserving the label to create additional data points (Shorten and Khoshgoftaar, 2019). However, in NLP, augmenting data is a very difficult task due to the complex nature of language (Wei and Zou, 2019). In this paper we assume a real-life scenario where the data at hand is insufficient for running a typical active learning algorithm. We introduce a method that enables us to automatically generate artificial text examples that complement an existing dataset. Our approach minimizes the human factor in data creation by automating the process through a guided searching procedure.

Once a set of examples is generated, it is required to be manually labeled before it is added to the original training set. The classifier is then retrained on the new, augmented training set. This procedure is repeated multiple times until the desired performance is either achieved or until performance no longer improves. We do this in active learning cycles, where only a subset of the generated data is used; this involves selecting examples in terms of how much information they would add to an existing classifier. In our experiments we use entropy as a measure of informativeness.

As our aim is to find the most informative examples, we tackle this problem by applying a search approach. Given that text examples are generated from an extremely large number of possible combinations, we apply the Monte Carlo Tree Search MCTS algorithm (Browne et al., 2012) to limit the

[*] Corresponding author

search space. Here MCTS is expected to guide a language generation model to output informative examples. In this context, MCTS assigns values to previously generated examples using a scoring function that incorporates the learning classifier of the previous active learning cycle, starting with a baseline classifier trained on the initial dataset for the first active learning run. In our experiments, we test MCTS with two different scoring functions: one that only measures the uncertainty of the generated examples through entropy, and another that combines the measure of uncertainty with a measure of diversity by computing the cosine similarity of every newly generated example with the previous content. These scores determine the text premise that is passed to the language model when generating newer examples.

We compare MCTS to Non-Guided Data Generation (NGDG), an approach where the knowledge of the learning classifier is not involved in the data generation process. Here, for each newly generated example, the text premise is always a token representing the beginning of a sentence, <bos>.

The remainder of this paper is organized as follows; Section 2 provides a background as well as an overview of related literature. Section 3 describes the proposed approach. Section 4 presents the experiments which were carried out. Section 5 gives conclusions and plans for future work.

2 Background

2.1 Active Learning

In this paper we consider the pool-based AL model, a commonly adapted approach in text classification problems (Hu et al., 2016; Krithara et al., 2006; Tong and Koller, 2001; Nigam and McCallum, 1998). This approach assumes the availability of all the data from the beginning of the process. We start with a set of data S_D, where a large pool of it is unlabeled S_U, leaving only a small subset S_L with labels $l_1, l_2, .., l_n \in L$. Hence, $S_D = S_U + S_L$. A classifier is first trained on S_L. Then, at each AL iteration, a selection strategy is applied to select a pool of data S_P from S_U to be labeled by the expert. Examples in S_P are chosen on the basis of being the most informative of S_U, such that, if added to the training data, an improvement in the classifier's performance is to be expected.

As described by Yoo and Kweon (2019); Siddiqui et al. (2019), there are three main selection strategies that can be applied to obtain S_P: uncertainty-based approaches, diversity-based approaches, and expected model change. In an uncertainty-based selection strategy, the active learner chooses the examples that it is most uncertain about. This assumes a probabilistic framework where the learner predicts a probability distribution $P = (p_1, p_2, ..., p_n)$ for labels $L = (l_1, l_2, ..., l_n)$ for a given example $e_i \in S_U$. In a binary classification setting, Lewis et al. presume that the most uncertain examples have a posterior probability closest to 0.5 for any label $l_i \in \{0, 1\} \forall i$ (Lewis and Gale, 1994; Lewis and Catlett, 1994). In a multi-class setting, a selection strategy could choose examples with the lowest posterior probability or be based on entropy (eq. 5) as in (Hwa, 2004; Settles and Craven, 2008; Joshi et al., 2009). Given that the difference in degree of certainty for similar examples can be small, uncertainty selections are prone to return similar examples (Wang et al., 2017). To address this issue, some works incorporate measures to exploit the diversity information of the examples in the selection process (Sener and Savarese, 2017; Wang et al., 2017; Sinha et al., 2019). Finally, expected-model change selects examples that would cause the greatest change to a model's output if their labels were known (Freytag et al., 2014; Roy and McCallum, 2001; Settles et al., 2008). This approach however, can be computationally expensive for big data and large feature spaces (Settles, 2009). Hence, this approach has not been very successful with deep learning models (Siddiqui et al., 2019).

In sum, research on active learning has focused on applications where a large pool of unlabeled data already exists. However, we are interested in real-life scenarios where this data may not be available. Other approaches such as Snorkel[1] use heuristics to generate data (Ratner et al., 2017) but this can prove impractical for text. In this paper we consider the case were the number of available data S_D is extremely small, so that typical active learning approaches become inapplicable due to the absence of S_U. Our aim is to generate synthetic data for S_U that can then be queried by an active learning algorithm to select an informative subset S_P for labeling. The selection process we apply can be classed as an uncertainty approach, except for the Diversity-Based MCTS (described in section 3.1.4) which incorporates a similarity check that could also be classed as a diversity approach.

[1]https://www.snorkel.org/

2.2 Language Models

The year 2018 proved to be an exceptional one for the NLP community as research shifted rapidly from pretrained shallow embeddings to more complex pretrained language models adopted from the computer vision field. This is evident in developments such as Embeddings from Language Models (ELMO) (Peters et al., 2018), Universal Language Model Fine-tuning for Text Classification (ULMFIT) (Howard and Ruder, 2018), generative pretraining (Radford et al., 2018), Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al., 2018) and many more.

In the same year, OpenAI released a transformer-based language model, GPT (Radford et al., 2018), which was trained with a traditional language modeling approach by predicting the next word in a sequence. In language modeling, the objective is to estimate the probability of a next token in a sequence conditioned on the context tokens (Bengio et al., 2003). In a unidirectional training approach such as GPT-2, the context is the history or past tokens. The probability of the sequence for tokens $w = w_1, ..., w_n$ can be defined as:

$$p(w) = \prod_{i=1}^{i=n} p(w_{(i)}|w_{(1)}, \ldots, w_{(i-1)}) \tag{1}$$

As such, given a set of sentences as input, the objective of the language model is to find the parameters θ that maximize the log-likelihood:

$$\theta^* = \arg\max_{\theta} \{\log p(w;\theta)\} \tag{2}$$

This training procedure did not give GPT the edge over other state-of-the-art models like BERT on classification tasks, possibly due to BERT taking advantage of a bidirectional architecture. However, this did not stop OpenAI's GPT from prevailing in other departments. As it turns out, compared to BERT, GPT is able to generate text sequences of higher quality (Wang and Cho, 2019). In 2019, OpenAI released a successor, GPT-2 (Radford et al., 2019) that included slight adjustments to the original GPT model and was trained on larger data collections. Although its architecture was very similar to its predecessor, GPT-2 presented significant progress in language generation. The main contribution of GPT-2 was its ability to scale up training parameters from 110 Million to 1.5 Billion. In our experiments we were able to achieve satisfactory results with the smallest version of the GPT-2 model; 12 hidden layers and 124M parameters.

2.3 Monte Carlo Tree Search MCTS

MCTS is a tree search method that attempts to find compelling solutions without having to run to completion. It does so by walking through random paths in the search space while constructing a tree using the results of a predefined reward function. Due its ability to find paths leading to an optimal solution when the search space is infinitely large, MCTS has been widely adopted by the AI gaming community (Arneson et al., 2010; Perez et al., 2013; Chang et al., 2016).

The longer MCTS runs, the stronger its moves get. This is because it manages to balance between two main criteria: exploring new search paths and exploiting paths that have been already explored. MCTS consists of four major steps: Selection, Expansion, Simulation, and Backpropagation (Chaslot et al., 2008). When applied to board games, MCTS constructs a tree to determine a winning strategy. In this setting, a node represents a board position, an edge represents a move, and a path represents a sequence of moves.

Selection: Starting from a root node R, the algorithm selects a move that leads to a node N that has no identified children. On the one hand, the selected move could be random, ignoring the scores of already visited paths; in this case, some paths are not visited. On the other hand, the selected move could be completely based on already visited nodes (e.g. by storing average wins for each node); here, the algorithm might miss other nodes that could lead to higher win rates. In order to balance between the benefits of exploration and exploitation, a selection policy such as the Upper Confidence Bound (UCB) can be used (Auer et al., 2002). UCB makes sure that as many nodes as possible are explored while still favouring branches that are visited more often than their counterparts. The selection is then done by choosing the nodes with the highest UCB value. In our implementation, given that the scoring criterion is an entropy value ranging between 0 and 1, we apply a small adjustment to the vanilla UCB policy, described in section 3.1.1:

$$UCB = \frac{W_i}{S_i} + C\sqrt{\frac{2 \times lnS_p}{S_i}} \tag{3}$$

where W_i is the number of simulations generated from node i which resulted with a win, S_i the total number of simulations generated from node i, S_p

the total number of simulations generated from the parent node, and C an exploration parameter.

Expansion: In this phase, a new child node is added to the node selected in the previous step. This new node is based on a random selection of one of the possible moves. The values for this new node are initialized to 0 wins out of 0 simulations where $W_i = 0$ and $S_i = 0$.

Simulation (Roll-out): A simulation is run from the root node R until a terminal node T is found. The terminal node will output a value that is then passed upwards in the backpropagation phase.

Backpropagation: A simulation stops when a terminal node T is reached. The values for each node leading to T are then updated by adding 1 to the number of visits S_i and number of wins W_i.

2.4 Related Work

In previous work, our group applied a similar framework to a private dataset, where instead of GPT-2, a recurrent neural network was used to generate words, and the reward function was solely based on entropy (Sankarpandi et al., 2019). Furthermore, experiments were based on a much larger initial training set, and there was an added burden on the user to manually correct ill-formed outputs. To our knowledge, the next closest work to ours is Anaby-Tavor et al. (2019), where GPT-2 and a classifier are applied to generate new weakly-labeled examples. This process involves fine-tuning GPT-2 on existing training examples while providing the class labels as part of the input. Examples are then selected and kept as training data based on the classifier's confidence score. Kumar et al. (2020) further explores this approach with different transformer based models (Vaswani et al., 2017) for data generation. This approach however, relies on GPT-2 to provide weak labels as it generates data. It also does not employ a guided search to generate the best examples at a given stage. By excluding the process of generating weak labels, this approach could be considered analogous to our Non-Guided Data Generation method in section 4.2.

Other data augmentation work in NLP relies on generating examples that are simply different forms of the existing text. Two approaches are word manipulation and the use of back-translations. Word manipulation involves techniques like randomly swapping words, replacing words with their synonyms, or deleting random words (Wei and Zou, 2019). In the same vein, Wang and Yang (2015) randomly replace words with neighboring ones from an embedding space. Kobayashi (2018) uses a bidirectional language model to randomly replace words with alternatives. Here the language model is fed the sentence input excluding the word at position x, to predict an alternative word at position x. Another word replacement approach is applied by Wang et al. (2018b) in a machine translation task, where words in both the source and target sentence are replaced with other random words. In back-translation, also known as round-trip translation (Aiken and Park, 2010), an input text is translated into an intermediate language and then the result is translated back to the original language. This technique is applied in the works of Sennrich et al. (2015); Aroyehun and Gelbukh (2018).

3 Proposed Method

In games, MCTS can be applied to predict moves in order to counter an opponent's strategy so that a winnable state is reached. However, text generation is more similar to a single player scenario, where decisions are based on which token to select when moving from one state to the other.

A language model calculates a probability distribution over a sequence of words. When passing over a stream of text, each vocabulary token is assigned a probability score for occurring next. Hence, tokens with higher probability scores are more likely to appear next in the sequence. In our setting, we are interested in multiple token candidates for all remaining words in the sequence. To achieve this we use a top-k sampling scheme as used by Fan et al. (2018). At each time step, each token in the vocabulary is assigned a probability score for coming next in the sequence. To get the top k candidates, vocabulary tokens are sorted by their probability scores and anything below the k'th token is then zeroed out. The probability mass is then redistributed among the k token candidates.

This process can be modeled as a tree where each node represents a token linked to k child nodes representing the top k candidate tokens that are likely to appear next in the sequence. Hence, this is similar to a board game where each board position is represented by a node: The root node corresponds to an empty board while a terminal node is where no further moves can be made. In our setting, we use a special token for both the root and terminal nodes. We represent the starting token with <bos>

and the ending token with <eos>.

As an example, a language model that is fine-tuned on a survey on pet adoption could be used to generate the tree of predictions in Figure 1. A full version of this tree would represent all the possible combinations of text that can be generated by the language model. In an ideal setting, we would search this tree for the paths that represent the most informative examples. However, given that the tree will grow exponentially as the number of next token candidates is increased, it would be computationally expensive to apply a brute force search algorithm where every path is examined. For this reason, we apply the Monte Carlo Tree Search (MCTS) algorithm in the data generation process, as discussed next.

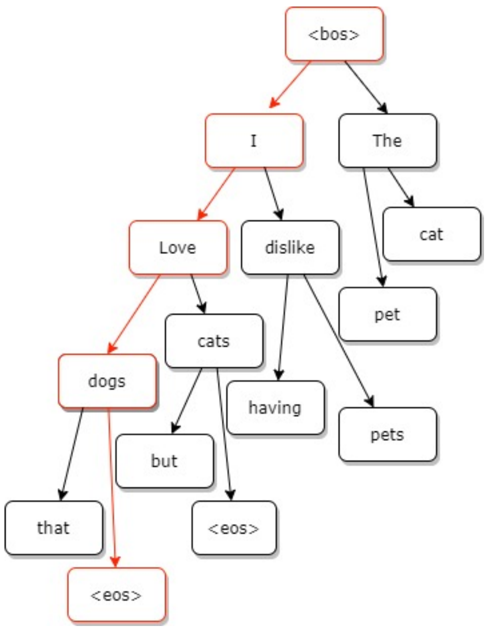

Figure 1: MCTS traverses down the tree as it creates paths spanning from the root node <bos> until a terminal node <eos> is reached. Tokens of the same path form a sentence when concatenated e.g. the path in red.

3.1 MCTS for Data Generation

In a typical MCTS application, a separate tree is formed for every decision. Applying this to our approach would require us to build a tree for each next word in the sequence. This would be computationally expensive due to the overhead of generating candidates and computing reward values. An alternative would be to construct a single tree only, while allowing MCTS to run for a longer period. This would result in a tree where each path is a possible output. However, given the nature of MCTS where the paths generated in the roll-out phase are not stored, we would be left with many incomplete paths which did not reach a terminal node (see Figure 1). To account for this, we keep track of all the simulations without impacting the selection policy. Thus we still have the same tree as in Figure 1, but we also have a record of the paths generated from non-terminal leaf nodes. The Selection, Expansion, Simulation and Backprogation phases for each MCTS iteration are described in the following sections.

3.1.1 Selection

The vanilla UCB function is mostly adopted in strategies where the outcome is chosen from a fixed set of categorical values, win, lose or tie. The objective is to reach a winnable state with the minimum number of visits. This is reflected in the UCB equation (eq. 3). By contrast, our purpose is to maximize the importance of nodes that lead to higher reward values (section 3.1.4), as shown in eq. 4, adopted from Chaudhry and Lee (2018):

$$UCB = max(N_i) + C\sqrt{\frac{2 \times lnS_p}{S_i}} \qquad (4)$$

where $max(N_i)$ is the maximum reward at node i, C is an exploration constant, S_i is the total number of visits to node i, and P_i is the total number of visits to the parent node for node i.

3.1.2 Expansion

Once a node is selected, we add all its immediate child nodes. These are the allowed moves from a given state, that is the top k token candidates generated by a language model, given the state's context history. Figure 2 illustrates the process. For $k = 3$, the context history for the state at the root node is the token <bos>. When passed to a language generation model, the words “Where”, “What” and “Who” are examples of the top three candidates to follow <bos>. Assume the node “Who” is picked in the selection phase at $i = 1$. Its state context history “<bos> Who” returns the candidate tokens “discovered”, “is”, and “invented”; these are added as child nodes in the expansion phase.

3.1.3 Simulation (Roll-out)

A simulation starts from the added child node in the expansion phase. During this process, a sequence is generated by picking at random a possible candidate for each next token until a terminal state is reached. Given that candidate tokens are generated over a probability distribution, we apply a

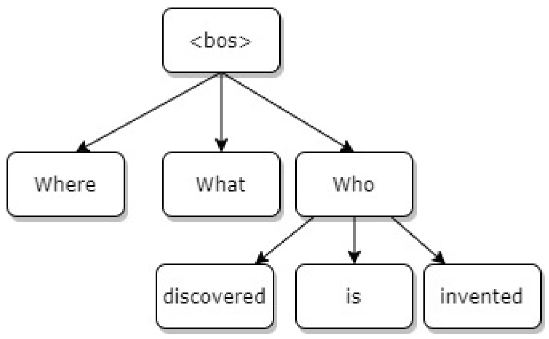

Figure 2: A possible MCTS output after 2 iterations

weighted choice method, enforcing non-uniform randomness. As explained earlier in 3.1, simulations are tracked without affecting the growth of the tree. Taking this into account, we are able to modify the value of k for the tokens, without affecting the functionality of MCTS. We will refer to this as K_s to distinguish it from the K in the expansion phase. If $K_s >> K$, we are able to achieve higher variance in the generated data while maintaining the width of the tree.

3.1.4 Backpropagation

Once an example is generated, its reward value is computed. The path of the expanded node is then updated by backpropagating the reward value and increasing the number of visits by one.

For the reward function, we implement two variants of MCTS, hereafter referred to as Uncertainty-Based MCTS and Diversity-Based MCTS, where the only difference is in the reward function.

In **Uncertainty-Based MCTS**, given the learning classifier's softmax probabilities over the possible class labels, we compute the normalized form of Shannon's entropy as shown in eq. 5:

$$H_n(P) = -\sum_{i=1}^{n} p_i \log_b p_i \cdot \frac{1}{\log_b n} \tag{5}$$

where P is a set of probabilities $P = \{p_i; i = 1, ..., n\}$, with $\sum_{i=1}^{n} p_i = 1$ for n labels, normalized by $log_b n$. We expect meaningless content in regions of higher entropy, and so limit the search space to a predefined value for maximum entropy, θ_{ent}. Examples with an entropy above this threshold become less important by returning a lowered reward value (e.g. 0), as shown in eq. 6.

$$f(x_{ent}) = \begin{cases} 0, & \text{if } x_{ent} \geq \theta_{ent} \\ x_{ent}, & \text{otherwise} \end{cases} \tag{6}$$

where x_{ent} is the entropy value for example x, and θ_{ent} the entropy cut-off threshold.

In **Diversity-Based MCTS**, in addition to entropy, we compute the cosine similarity between each generated candidate and a comparison list initialized with the classifier's training data. This is to ensure the diversity of the generated examples. If the similarity score is above a certain threshold θ_{sim}, the reward for the candidate will be set to 0, as shown in eq. 7. Conversely, if it is below θ_{sim}, the candidate is added to the comparison list, so that future candidates will be penalized if they are too similar to it.

$$f(x_{ent}, x_{sim}) = \begin{cases} 0, & \text{if } x_{ent} \geq \theta_{ent} \\ 0, & \text{if } x_{sim} > \theta_{sim} \\ x_{ent}, & \text{otherwise} \end{cases} \tag{7}$$

where x_{sim} is the maximum cosine similarity score between example x and the comparison list, and θ_{sim} the cosine similarity threshold.

3.2 Data Selection and Active Learning

Once MCTS reaches completion, all leaf nodes from the final tree are selected. Given that we have kept track of the generated simulations from a node, each non-terminal leaf node is now linked to a generated sequence of text. The final set of text examples is then sorted by the values from their corresponding nodes. The top n examples are selected, labeled by hand and appended to the original training set. We then retrain the learning classifier on the new dataset.

4 Experiments

4.1 Datasets

In our experiments, we attempt to emulate real life scenarios where training data is scarce. So from each dataset below, we create an initial training set by randomly selecting a very small subset of the available training data. We then fine-tune GPT-2 on the created subset and use our method from section 3 to generate new training examples. Once data is generated, we label the top n examples, sorted by the max reward value, as described in 3.2.

We study the effectiveness of our methods on two different tasks, question classification and sentiment analysis.

Question Classification: For this task we use the 6-label version of the TREC Questions dataset, TREC-6 (Li and Roth, 2002). TREC-6 divides questions into 6 categories: HUM, DESC, ENTY, LOC, NUM, and ABBR. From the available training data, we randomly select only 5 examples per label, making a total of 30 examples for training the baseline classifier. Evaluation is done over the provided test set of 500 questions.

Sentiment Analysis: For this task we use the Stanford Sentiment Treebank SST-2 Dataset (Socher et al., 2013), with sentiments divided into 2 labels, positive and negative. We use the data split from the GLUE SST task (Wang et al., 2018a), and evaluate on the provided development set. From the available training data, we only select 10 random examples per label and discard the rest.

4.2 Model Comparison

In our experiments we compare two variants of MCTS with only a minor difference in the reward function, one with the effect of θ_{sim} as described in section 3.1.4, and one without its effect. We further test the effectiveness of MCTS for data augmentation by comparing it to a data generation approach that does not optimize for a reward value. We refer to this approach as non-guided data generation (NGDG). Similar to MCTS, NGDG applies a top-k sampling procedure to generate candidate tokens. However, unlike MCTS, the selection of the next token is entirely based on the distribution of the candidate tokens. This is exactly the same procedure as the simulation phase in MCTS, but instead of constructing a tree search, simulations are run independently of one another. To emulate the flexibility of having higher variance over the latter parts of the generated text in MCTS (section 3.1.3), we increased k for the number of candidate tokens after the first n output tokens in the sequence. Here n is fixed at 3 in all our experiments. After the data is generated, we apply the classifier of the previous active learning cycle to compute an entropy value (eq. 5) for each example. Data is then sorted by the entropy, and the top n examples below θ_{ent} are then selected for labeling.

The classifier used for our experiments is a relu layer neural network with the Universal Sentence Encoder (USE) for the embedding layer. We implement this classifier using the Keras[2] toolkit. The classifier contains an embedding layer with 512 neurons, a 600-neuron fully-connected dense layer, a dropout layer with a 0.2 dropout rate, a softmax activation output and optimized using Adam (Kingma and Ba, 2014). We fixed the classifier's hyper-parameters following a hyper-parameter search to a batch-size of 2, 0.0001 learning rate, and trained over 15 epochs.

[2]https://keras.io/

4.3 Data Generation Parameters

We fix the MCTS UCB policy constant C to 2, θ_{ent} to 0.95, and θ_{sim} to 0.9 for all experiments. To achieve fairness in the comparison, when using the NGDG method (section 4.2), we discard examples with entropy above $\theta_{ent} = 0.95$ in the experiments.

4.4 Data Selection

For MCTS, as the learning classifier is part of the data generation process, the output examples are already mapped to their reward values and to the classifier's predicted labels. For NGDG, however, because the classifier does not take part in the generation process, it must be applied to the generated data afterwards, to predict labels and compute values for entropy. After classification, the data is sorted by entropy (for NGDG) or reward value (for MCTS), and the labels of the top n examples are corrected manually. Finally, to limit the effect of an imbalanced dataset, we restrict the number of the selected examples x_{max}, to the first 10 per label. In the event where all labels have more than 10 examples, x_{max} corresponds to the count of the label with the least number of examples.

4.5 Experiment 1: TREC-6 Question Data

For this experiment, we fixed the number of simulations at 3000 and the top n examples for labelling to 50 for both MCTS and NGDG. We set the number of candidate tokens K to 6 and K_s to 20. For NGDG, K changes from 6 to 20 after the first 3 tokens are generated from the sequence. Table 1 shows the average accuracy achieved over the 6 labels throughout 8 Active Learning runs on the TREC-6 test set, as well as giving the added number of examples after each AL cycle.

4.6 Experiment 2: SST-2 Sentiment Data

Similar to experiment 1, we kept the number of simulations at 3000 and top n at 50 for both MCTS and NGDG. Whereas, we set the number of candidate tokens $K = 15$ and $K_s = 30$ for MCTS. In NGDG, K changes from 15 to 30 after the first 3 tokens are generated. Results are in Table 2.

AL Run	MCTS		NGDG
	Diversity	Uncert.	
Start	65 (30#)	65 (30#)	65 (30#)
1	68 (48#)	78 (49#)	78 (47#)
2	86 (68#)	82 (52#)	86 (61#)
3	92 (73#)	87 (55#)	87 (72#)
4	91 (76#)	89 (59#)	88 (83#)
5	92 (83#)	91 (71#)	86 (89#)
6	91 (91#)	90 (76#)	84 (103#)
7	90 (94#)	89 (87#)	84 (113#)
8	91 (98#)	90 (94#)	88 (126#)

Table 1: Classification results after each Active Learning (AL) run for the TREC-6 question classification task. Before AL, 30 training examples result in 65% classification accuracy. After AL 1, under Diversity-Based MCTS for example, 18 new examples are added (total 48#), giving 68% accuracy, while under Uncertainty-Based MCTS (Uncert.), 19 new examples are added (total 49), giving accuracy 78%. The rest of the table is analogous.

AL Run	MCTS		NGDG
	Diversity	Uncert.	
Start	73 (20#)	73 (20#)	73 (20#)
1	74 (34#)	77 (34#)	69 (32#)
2	79 (41#)	76 (44#)	72 (43#)
3	79 (50#)	78 (48#)	75 (55#)
4	80 (60#)	80 (54#)	76 (79#)
5	80 (65#)	80 (55#)	75 (92#)
6	80 (79#)	80 (62#)	76 (103#)
7	83 (87#)	80 (64#)	79 (116#)
8	83 (95#)	79 (69#)	78 (124#)

Table 2: Classification results after each AL run for the SST-2 sentiment analysis task with top $n = 50$.

AL Run	MCTS		NGDG
	Diversity	Uncert.	
Start	73 (20#)	73 (20#)	73 (20#)
1	77 (26#)	72 (24#)	68 (30#)
2	74 (29#)	74 (27#)	75 (41#)
3	78 (37#)	74 (34#)	77 (49#)
4	79 (43#)	73 (38#)	76 (56#)
5	80 (46#)	76 (39#)	81 (60#)
6	80 (49#)	78 (42#)	81 (64#)
7	81 (52#)	76 (44#)	79 (72#)
8	81 (57#)	78 (45#)	80 (77#)

Table 3: Classification results after each AL run for the SST-2 sentiment analysis task with top $n = 20$.

#	Example
1	Why did Einstein lose a fight with cancer?
2	Why did Lincole Ljungberg retire?
3	Why was Lorne L. Huntington's IQ so low?
4	What are three fundamental principles of socialism?
5	What is D.C.'s major metropolitan area?
6	When was Antarctica formed?
7	When did animals roam the earth?
8	Where can a geologist find fossils?
9	Where can an electrician find work?
10	How did Moses rule the ancient tribes?
11	How often have animals been killed by car crashes?
12	Which is Fordham's largest engineering college?

Table 4: Some examples generated on TREC-6 through the Diversity-Based MCTS for experiment 1.

4.7 Experiment 3: SST-2 Sentiment Data

We repeated experiment 2 with the same configurations, except that top n, is now 20 (not 50) for both MCTS and NGDG. Results are shown in Table 3.

4.8 Discussion

Table 4 shows twelve sentences generated by the Diversity-Based MCTS. These can give us insights concerning our approach and the role of GPT-2 in it. First, consider example 1 in the table ("Why did Einstein lose a fight with cancer?" – type DESC). In the initial training set, there is only one mention of Einstein ("What was Einstein's IQ?" – NUM), and one of cancer ("How do doctors diagnose bone cancer?" – DESC). Nevertheless, example 1 combines information from two different sentence types NUM and DESC in a coherent way. Example 3 again demonstrates a form of 'cross-type' learning: The Einstein training sentence above is the only mention of IQ and is of type NUM. Yet example 3 is a well-formed DESC sentence. For example 4, perhaps the most related training instances are "What are the four elements?" and "What are the chemicals used in glowsticks?". These are asking for lists but concerning elements and chemicals, not abstract concepts like socialism.

Interestingly, even though the training set contains no 'When' sentences, examples such as 6 and 7 could still be created; because MCTS pushes GPT-2 to generate novel sentences as it constructs the tree, those of the form "What kind, when..."

were created during the path traversal process. These were then corrected during the labeling stage. We did not witness this phenomenon with NGDG, possibly because MCTS is directed by a reward function that penalizes sentences of low entropy. This allows MCTS to search through the space of possible sentence combinations more efficiently.

Concerning the LOC examples 8 and 9, the only 'Where' training question is "Where do hyenas live?". Yet, in our experiments, we were able to expand on this by generating additional 'Where' questions which are very different from the hyenas and different from each other: A fossil is something which a geologist might find, while work is something which an electrician might find. Both are meaningful, while the sense of 'find' in each is quite distinct. Finally, while the remaining examples in table 4 could not be directly linked to relevant examples in the training data, this only confirms our purpose of using a pretrained model like GPT-2 that can make use of its external knowledge while remaining relevant to the target task.

In summary, by integrating GPT-2 with our methods, we gained substantial improvements over the baseline classifier. This shows how text generation can improve performance for tasks with scarce data. Even when starting with just a few examples per label, we were able to generate informative data that boosted the accuracy on TREC-6 from 65% to 91% with MCTS and 88% with NGDG, on SST-2 from 73% to 83% and 78% respectively, after 8 AL runs. Even when reducing the number of examples for labeling from 50 to 20 in experiment 3, we were still able to achieve an improvement of 81% with MCTS and 80% with NGDG. This suggests the effectiveness of our approach in solving real-world classification tasks when minimal data is available. Moreover, with MCTS we witnessed improvements in performance compared to NGDG on both the TREC-6 and SST-2 datasets. MCTS guides the growth of the tree by visiting more relevant nodes more frequently. Hence, relevancy is increased by the paths that maximize the reward function, those that correspond to high entropy values in our setting.

However, searching only for high entropy is more likely to incur noise in the final output such as ill-formed sentences or content that does not fall under the labeling criteria. Since ill-formed sentences are likely to incur high entropy values, the lack of a sentence quality measure can make MCTS prone to output meaningless sentences. For instance, "What kind!!??", "Which is the abbrev?", and "What does IQ be?" were outputs of MCTS in the TREC-6 experiments. This point is reflected in the lower overall number of added examples when comparing MCTS to NGDG over the 8 AL runs. Moreover, when MCTS over-exploits visited paths, it can get stuck in certain sub-trees, leading it to output examples with a high level of similarity. For instance, "good movie" and "good movie!" are identical examples with the only difference being the exclamation mark '!'. This issue is especially noticeable in the MCTS Uncertainty-Based experiment in Table 3, where due to the number of closely similar examples in the output, a lower proportion of the top 20 examples could be labeled. Hence, to diversify the generated output, we introduced θ_{sim} in the MCTS Diversity-Based approach.

Overall, the success of our approach relies on the quality of the search space, which is determined by the language model; if it performs less well, this can result in a noisier space. Hence, in our previous work (Sankarpandi et al., 2019), our group could not achieve comparable results. Moreover, additional user involvement was needed to make sense of ill-formed outputs, making the whole approach laborious and more prone to the user's bias.

5 Conclusion

In this paper we proposed a framework for improving a classifier's performance with synthetic data. We have shown in our experiments that even when starting with just a few examples, we are able to achieve noticeable improvements. We believe this approach is likely to work for any domain or language so long as the language model is able to generate meaningful output. In this work for instance, we did not need more than 20 examples to fine-tune GPT-2 for the SST-2 experiments, or 30 for the TREC-6 experiments. We expect even better results when more examples are provided or with the application of an improved language model like GPT-3. In future work, we plan to extend our approach to further improve the reward and policy functions, and to reduce the human-labeling factor.

Acknowledgments

Husam Quteineh acknowledges the support of the Business and Local Government Data Research Centre BLG DRC (ES/S007156/1) funded by the Economic and Social Research Council (ESRC).

References

Milam Aiken and Mina Park. 2010. The efficacy of round-trip translation for mt evaluation. *Translation Journal*, 14(1):1–10.

Ateret Anaby-Tavor, Boaz Carmeli, Esther Goldbraich, Amir Kantor, George Kour, Segev Shlomov, Naama Tepper, and Naama Zwerdling. 2019. Not enough data? deep learning to the rescue! *arXiv preprint arXiv:1911.03118*.

Broderick Arneson, Ryan B Hayward, and Philip Henderson. 2010. Monte carlo tree search in hex. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):251–258.

Segun Taofeek Aroyehun and Alexander Gelbukh. 2018. Aggression detection in social media: Using deep neural networks, data augmentation, and pseudo labeling. In *Proceedings of the First Workshop on Trolling, Aggression and Cyberbullying (TRAC-2018)*, pages 90–97, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3(null):1137–1155.

C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43.

Hyeong Soo Chang, Michael C Fu, Jiaqiao Hu, and Steven I Marcus. 2016. Google deep mind's alphago. *OR/MS Today*, 43(5):24–29.

Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. 2008. Monte-carlo tree search: A new framework for game ai. In *AIIDE*.

Muhammad Umar Chaudhry and Jee-Hyong Lee. 2018. Feature selection for high dimensional data using monte carlo tree search. *IEEE Access*, 6:76036–76048.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Angela Fan, Mike Lewis, and Yann N. Dauphin. 2018. Hierarchical neural story generation. *CoRR*, abs/1805.04833.

Alexander Freytag, Erik Rodner, and Joachim Denzler. 2014. Selecting influential examples: Active learning with expected model output changes. In *European Conference on Computer Vision*, pages 562–577. Springer.

Jeremy Howard and Sebastian Ruder. 2018. Fine-tuned language models for text classification. *CoRR*, abs/1801.06146.

Rong Hu, Brian Mac Namee, and Sarah Jane Delany. 2016. Active learning for text classification with reusability. *Expert systems with applications*, 45:438–449.

Rebecca Hwa. 2004. Sample selection for statistical parsing. *Computational linguistics*, 30(3):253–276.

Ajay J Joshi, Fatih Porikli, and Nikolaos Papanikolopoulos. 2009. Multi-class active learning for image classification. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2372–2379. IEEE.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Sosuke Kobayashi. 2018. Contextual augmentation: Data augmentation by words with paradigmatic relations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 452–457, New Orleans, Louisiana. Association for Computational Linguistics.

Anastasia Krithara, Cyril Goutte, Jean-Michel Renders, and MR Amini. 2006. Reducing the annotation burden in text classification. In *Proceedings of the 1st International Conference on Multidisciplinary Information Sciences and Technologies (InSciT 2006), Merida, Spain.*

Varun Kumar, Ashutosh Choudhary, and Eunah Cho. 2020. Data augmentation using pre-trained transformer models. *arXiv preprint arXiv:2003.02245*.

David D Lewis and Jason Catlett. 1994. Heterogeneous uncertainty sampling for supervised learning. In *Machine learning proceedings 1994*, pages 148–156. Elsevier.

David D Lewis and William A Gale. 1994. A sequential algorithm for training text classifiers. In *SIGIR'94*, pages 3–12. Springer.

Xin Li and Dan Roth. 2002. Learning question classifiers. In *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, pages 1–7. Association for Computational Linguistics.

Kamal Nigam and Andrew McCallum. 1998. Pool-based active learning for text classification. In *Conference on Automated Learning and Discovery (CONALD)*.

Diego Perez, Julian Togelius, Spyridon Samothrakis, Philipp Rohlfshagen, and Simon M Lucas. 2013. Automated map generation for the physical traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 18(5):708–720.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *CoRR*, abs/1802.05365.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/researchcovers/languageunsupervised/language understanding paper. pdf*.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Alexander Ratner, Stephen H. Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. 2017. Snorkel: Rapid training data creation with weak supervision. *Proc. VLDB Endow.*, 11(3):269–282.

Nicholas Roy and Andrew McCallum. 2001. Toward optimal active learning through monte carlo estimation of error reduction. *ICML, Williamstown*, pages 441–448.

S. K. Sankarpandi, S. Samothrakis, L. Citi, and P. Brady. 2019. Active learning without unlabeled samples: generating questions and labels using monte carlo tree search. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 4628–4631.

Ozan Sener and Silvio Savarese. 2017. Active learning for convolutional neural networks: A core-set approach. *arXiv preprint arXiv:1708.00489*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Improving neural machine translation models with monolingual data. *CoRR*, abs/1511.06709.

Burr Settles. 2009. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences.

Burr Settles and Mark Craven. 2008. An analysis of active learning strategies for sequence labeling tasks. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 1070–1079.

Burr Settles, Mark Craven, and Soumya Ray. 2008. Multiple-instance active learning. In J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1289–1296. Curran Associates, Inc.

Connor Shorten and Taghi M Khoshgoftaar. 2019. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60.

Yawar Siddiqui, Julien Valentin, and Matthias Nießner. 2019. Viewal: Active learning with viewpoint entropy for semantic segmentation. *arXiv preprint arXiv:1911.11789*.

Samarth Sinha, Sayna Ebrahimi, and Trevor Darrell. 2019. Variational adversarial active learning. *CoRR*, abs/1904.00370.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

Simon Tong and Daphne Koller. 2001. Support vector machine active learning with applications to text classification. *Journal of machine learning research*, 2(Nov):45–66.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Alex Wang and Kyunghyun Cho. 2019. Bert has a mouth, and it must speak: Bert as a markov random field language model. *arXiv preprint arXiv:1902.04094*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018a. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Gaoang Wang, Jenq-Neng Hwang, Craig Rose, and Farron Wallace. 2017. Uncertainty sampling based active learning with diversity constraint by sparse selection. In *2017 IEEE 19th International Workshop on Multimedia Signal Processing (MMSP)*, pages 1–6. IEEE.

William Yang Wang and Diyi Yang. 2015. That's so annoying!!!: A lexical and frame-semantic embedding based data augmentation approach to automatic categorization of annoying behaviors using #petpeeve tweets. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2557–2563, Lisbon, Portugal. Association for Computational Linguistics.

Xinyi Wang, Hieu Pham, Zihang Dai, and Graham Neubig. 2018b. SwitchOut: an efficient data augmentation algorithm for neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 856–861, Brussels, Belgium. Association for Computational Linguistics.

Jason W. Wei and Kai Zou. 2019. EDA: easy data augmentation techniques for boosting performance on text classification tasks. *CoRR*, abs/1901.11196.

Donggeun Yoo and In So Kweon. 2019. Learning loss for active learning. *CoRR*, abs/1905.03677.