
Sentence Completion with HellaSwag Dataset

Author(s)

Vidur Goel
2021364
IIITD

Jyotirmaya Singh
2021055
IIITD

Aditya Ghosh
2020276
IIITD

Gunjan Dabas
2021253
IIITD

Aditya Daipuria
2021304
IIITD

Abstract

Commonsense reasoning is a fundamental yet challenging problem in Natural Language Processing (NLP). While novel deep learning techniques have made significant progress in this field, we take a different approach. In this paper we attempt to find the best model for a sentence autocomplete system using classical machine learning techniques only. We will be using the HellaSwag dataset for training, validation and testing. The HellaSwag dataset questions are found to be trivial for humans (with humans achieving more than 95% accuracy) while being challenging for machine learning models [1]. The complete study is yet to be conducted.

1 Introduction

Sentence completion is a fundamental task in natural language processing and understanding. Sentence completion requires predicting the most appropriate word or phrase to fill in the blank given some prior portion of the sentence. Since any given word in a sentence has a direct connection with its neighbouring words, sentence completion requires the understanding of grammar, semantics, and contextual information. The applications of a sentence completion model are numerous, from language modelling and text generation to content recommendation and sentiment analysis.

Our goal is to find the best performing classical machine learning model on the HellaSwag dataset by comparing various previously studied techniques and try to come up with our own novel approach.

2 Literature Survey

While a high majority of the current SOTA models on sentence completion (and for that matter, almost any NLP task) involve the usage of deep learning techniques, some progress had been made before the deep learning era.

We summarise our findings in Table 1.

MSR Dataset : MSR dataset is the Microsoft Research (MSR) Sentence Completion Challenge dataset. There are 1,040 test sentences which must be completed using one of five candidate words. Although it is not the exact same as HellaSwag, the techniques used can be useful to us since the core essence of the task is the same.

Table 1: Analysis of Sentence Completion Models

Paper	Dataset	Model	Result
A Challenge Set for Advancing Language Modeling [3]	MSR Sentence Completion Dataset	Smoothed 4-gram	39.0% accuracy
Efficient Estimation of Word Representations in Vector Space [5]	MSR Sentence Completion Dataset	Skip-gram	48.0% accuracy
Computational Approaches to Sentence Completion [4]	MSR Sentence Completion Dataset	LSA Total Similarity	49.0% accuracy
Dependency language models for sentence completion [7]	MSR Sentence Completion Dataset	Lab-SB	50.0% accuracy
Computational Approaches to Sentence Completion [4]	MSR Sentence Completion Dataset	Combination	52.0% accuracy
Efficient Estimation of Word Representations in Vector Space [5]	MSR Sentence Completion Dataset	Skip-gram + RNNLMs	58.9% accuracy
Exploiting Linguistic Features for Sentence Completion [6]	MSR Sentence Completion Dataset	Pointwise Mutual Information (PMI)	61.44% accuracy

[3] Zweig et al. used a simple 4-gram approach that yielded 34% accuracy on the test data. Using the CMU language modelling toolkit and Good-Turing Smoothing, the smoothened 4-gram method achieved an accuracy of 39% (5% more than the baseline). The vocabulary of words used in the smoothened model consisted of words with frequency greater than 5.

[4] Zweig et al. used LSA, which is a statistical technique used for identifying the underlying patterns or latent semantic structures in a corpus of text. LSA is combined with an n-grams approach by using linear interpolation between the n-grams and LSA probabilities. Out of all the models the authors tested, the LSA Total Similarity was the best performing individual model with an accuracy of 49%. Dietterich [8] says that we can obtain improved performance by using a combination of several methods for the task. Therefore, Zweig et al. used a linear combination of the outputs (probabilities of the sentence occurring) of the other models analysed - as inputs along with the LSA linear combination inputs. The LSA inputs included the total word similarity, the cosine similarity and the number of out-of-vocabulary terms in the answer. This ensemble method achieved an accuracy of 52%.

[7] Gubbins et al. propose probabilistic models for the lexicalisation of a dependency tree by generating words and paths in the dependency tree. The model is similar to the n-gram language model but allows for better generalisation. The model outperforms n-grams and achieves an accuracy of 50%.

[5] Mikolov et al. analysed skip-gram and RNNs which are neural network

based architectures. The above architectures achieved 58.9% accuracy when used together while skip-gram alone achieved only 48% accuracy.

[6] PMI is a statistical measure used to quantify the strength of the association between two terms in a dataset. It uses a series of formulae to calculate a score which measures the similarity between each possible answer and the words in the sentence. Wood created feature sets of bigrams and trigrams and incorporated these into semantic similarity assessment to achieve an accuracy of 61.44% - surpassing all methods discussed in this paper. For further validation, the model was tested on 285 sentence completion problems from SAT practice examinations where the model achieved 58.95% accuracy.

2.1 Inferences from literature

It is evident that the best direct approach available to us currently would be to use the Pointwise Mutual Index (PMI) method as proposed by Woods. As we can see in Table 1, it even outperforms several deep learning techniques, namely the combination of skip-gram with recurrent neural networks. Let's have a look at some other insights. The smoothed 4-gram model provides us with a base model upon which we could make improvements.

The concept of "memory" could be useful to us. Since using RNNs along with skip-gram boosted their accuracy by over 10% (observed in [5]), we could observe an improvement in our classical machine learning model by incorporating "memory" into it in some manner (keeping in mind that we don't end up using deep learning); via feature engineering for example.

Lexicalisation could also be used to capture and generate contextually appropriate text as was demonstrated by Gubbins et al [7].

Last but not least, we could also use ensemble techniques to improve the overall accuracy as was observed in [4] and [5]. A combination of statistical along with contextual approaches could possibly yield good results.

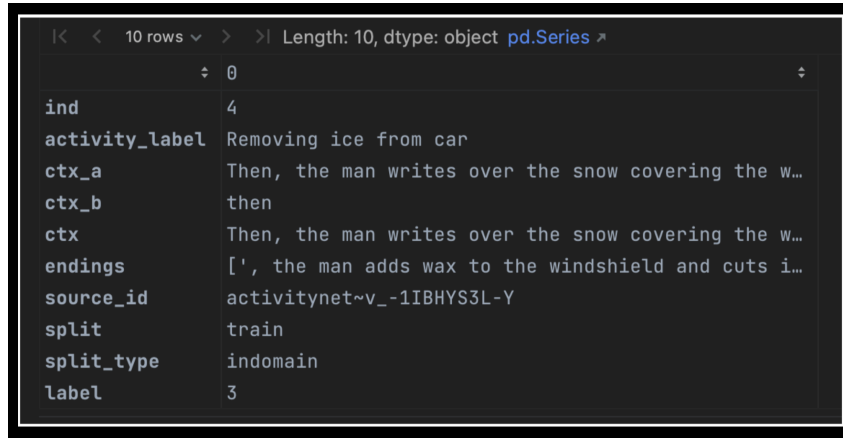
3 Dataset Review

The training dataset consists of 39905 rows and 10 columns. The rows are categorised by the kind of activity the sentence describes.

A sample row is shown in Figure 1. The column headers and their descriptions are summarised in Table 2 [2].

Table 2: Dataset Description

Column Name	Description	Model
ind	index of example	Integer
activity_label	The label of the activity being described by the sentence.	String
ctx_a	The context sentence A.	String
ctx_b	The context sentence B.	String
ctx	The entire context of string(ctx_a + ctx_b).	String
endings	The endings of the sentence. 4 possible endings are present for each example.	String
source_id	The ID of the source for the example.	String
split	The split of the dataset (train/test/val).	String
split_type	The type of split.	String
label	Index of the most appropriate ending from 'endings' column.	Integer



```

|< < 10 rows > >| Length: 10, dtype: object pd.Series ↗
┆
┆      0
┆
ind      4
activity_label Removing ice from car
ctx_a      Then, the man writes over the snow covering the w...
ctx_b      then
ctx      Then, the man writes over the snow covering the w...
endings    [' , the man adds wax to the windshield and cuts i...
source_id  activitynet~v_-1IBHYS3L-Y
split      train
split_type indomain
label      3

```

Figure 1: Sample entry of the Dataset

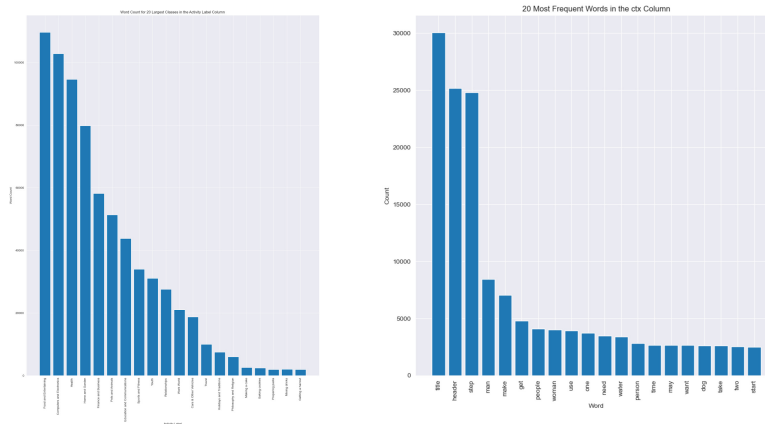
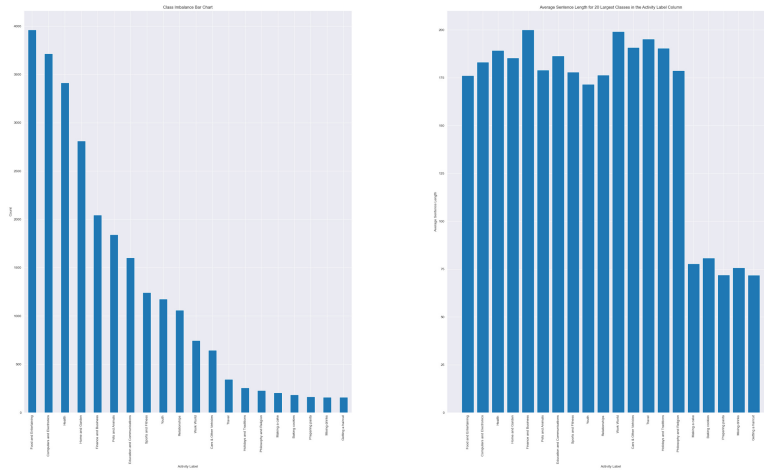
4 Exploratory Data Analysis and Preprocessing

4.1 Preprocessing

Initially, there were about 25,000 null values in the ctx_b column. This was due to the context sentence not having an appropriate starting word. Therefore, we inserted a placeholder '<start>' token in these columns for the time being as dropping these columns would lead to a significant loss of data. If we find a better strategy, we shall resort to that. There were also some sentences which consisted of only a period (.), these were replaced with a placeholder '<end>' token. We drop the irrelevant columns (such as ind, source_id, split and split_type) and split the 4 ending options into separate columns (labelled A0, A1, A2 and A3 respectively). Finally, we perform the traditional preprocessing steps for any NLP task, namely - converting the text to lowercase, removing punctuations and special characters, tokenization and removing stop words. We also performed lemmatization rather than stemming since our task is text completion and requires the use of meaningful words. We conduct further analyses on this dataset.

4.2 EDA

Since our data was quite noisy, we performed preprocessing before EDA to make the data more interpretable and yield more meaningful results. Since there are 178 different activities, for the sake of simplicity and clarity we plot the statistics of only the 20 most occurring classes here.



4.2.1 Class Imbalance Inference

We can observe from the generated bar plot that there is class imbalance present in the dataset. 'Food and Entertaining', 'Computer and Electronics', 'Health', etc., form a big component of the dataset. To mitigate this imbalance, we will use resampling techniques or cost sensitive learning.

4.2.2 Average Sentence Length Inference

We can observe from the generated bar plot that there is average sentence length imbalance present in the dataset. Some classes have a much higher average sentence length than others. This is important because imbalanced sentence lengths can affect the performance of models designed for sequence processing. We could perform padding and truncation to ensure that sentences have equal length.

4.2.3 Word Count Inference

We can observe from the bar plot that there is a big word count imbalance present in the dataset. This is a natural result of class imbalance as the classes with more samples will have a higher word count. We could pad shorter texts and truncate longer texts to a fixed or maximum word count.

4.2.4 Most Frequent Words/Unigrams Inference

We can observe from the given bar plot that words such as ‘title’, ‘header’ and ‘step’ are the most frequently occurring. This may seem natural but upon taking a look into the dataset, we realise that these are in-fact indicators of the ‘header’ of a paragraph or the ‘title’ of an article or the ‘step’ involved in a process. These indicator words have also been used in the ‘endings’ column. This is a double edged sword, as there might be correlations between the ‘ctx’ and ‘endings’ column through these words which can help in our task, however it may also be that the model runs better without such indicator words being present. Currently, I have left these as they are and I plan on running models with and without these variables to see how its accuracy changes. Other than that, the most frequently occurring words such as ‘man’ and ‘people’ tell us that the sentences present are mostly about human activities.

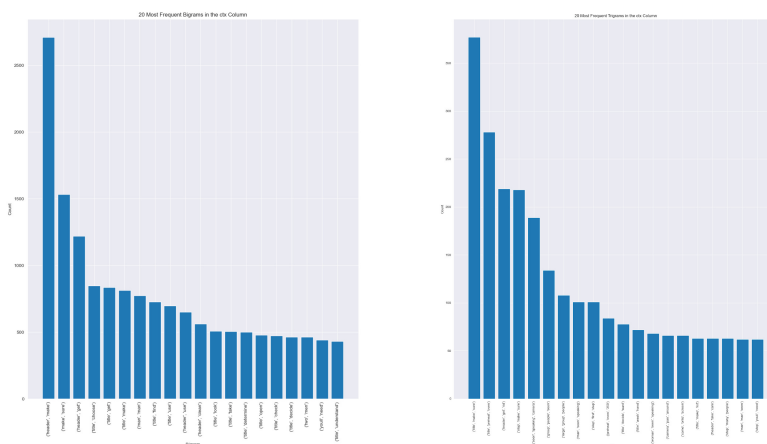


Figure 4: a)20 most frequent bigrams b)20 most frequent trigrams

4.2.5 Most Frequent Bigrams Inference

We can observe from the given bar plot that most frequently occurring bigrams start with the words ‘title’ and ‘header’. I will conduct the analysis as described in the case of ‘Most Frequent Words/Unigrams’ paragraph. Other than that, this plot can help us give an idea about what the text is about and understand its language style and cultural context.

4.2.6 Most Frequent Trigrams Inference

We can observe from the given bar plot that most frequently occurring trigrams start with the words ‘title’, ‘header’ and ‘step’. I will conduct the analysis as described in the case of ‘Most Frequent Words/Unigrams’ paragraph. Other than that, this plot can help us give an idea about what the text is about and understand its language style, cultural context and identify repetitive elements in the set.

5 Methodology

Amongst classical machine learning techniques, N-grams is the most popular option for text prediction/generation tasks. A thorough literature survey found out that using an N-grams model along with PMI yielded the highest accuracy on the MSR dataset challenge. Hence, we embark on implementing an N-grams model while enhancing its predictions using the PMI (Pointwise Mutual Information) metric. We tweak our model to be generative rather than picking one out of a few options available (as was the case with the MSR dataset and Hellaswag dataset).

An N-grams model is a type of probabilistic language model used in natural language processing (NLP) and computational linguistics. It is based on the assumption that the probability of a word in a text depends only on the previous n-1 words, also called the Markov assumption. The

Markov assumption simplifies the complex task of predicting the likelihood of a word sequence. An "n-gram" is defined as a contiguous sequence of n items from a given sample of text or speech. The items can be phonemes, syllables, letters, words, or base pairs, depending on the application. In the context of text, n-grams are typically sequences of words.

A 1-gram (or unigram) is a single word.

A 2-gram (or bigram) is a sequence of two words.

A 3-gram (or trigram) consists of three words, and so on.

N-grams are used for several purposes including Language Modeling, Text Prediction, Speech Recognition and Machine Translation. We use N-grams to implement our autocompletion system.

Definition of an N-gram: An n-gram is a sequence of n words: w_1, w_2, \dots, w_n . For example, in a trigram model (where $n = 3$), a sequence might be "the cat sat". Probability of an N-gram: The probability of an n-gram in a corpus is often estimated using the relative frequency of that n-gram in the training data. The probability of a word sequence $P(w_1, w_2, \dots, w_n)$ can be approximated as:

$$P(w_1, w_2, \dots, w_n) \approx \frac{\text{Count}(w_1, w_2, \dots, w_n)}{\text{Total Number of n-grams}}$$

Conditional Probability: N-grams models often use conditional probability to estimate the likelihood of a word given the previous $n - 1$ words. In a bigram model, for instance, the probability of a word w_n given the previous word w_{n-1} is:

$$P(w_n - 1 | w_n) = \frac{P(w_{n-1}, w_n)}{P(w_{n-1})} \approx \frac{\text{Count}(w_{n-1}, w_n)}{\text{Count}(w_{n-1})}$$

Chain rule of probability: The joint probability of a word sequence can be decomposed into a product of conditional probabilities using the chain rule:

$$P(w_1, w_2, \dots, w_n) = P(w_1) \cdot P(w_2 | w_1) \cdot P(w_3 | w_1, w_2) \dots P(w_n | w_1, w_2, \dots, w_{n-1})$$

Pointwise Mutual Information Pointwise Mutual Information (PMI) is a measure used in information theory and statistics to quantify the association between two events or variables. In the context of natural language processing (NLP), it's often used to discover collocations and associations between words in text data.

PMI between 2 event X and Y can be given as

$$\text{PMI}(X; Y) = \text{Log}\left(\frac{P(X, Y)}{P(X) \cdot P(Y)}\right)$$

where:

- $P(X, Y)$ is the conditional probability of X and occurring together.
- $P(X)$ and $P(Y)$ are the probabilities of X and Y respectively.

Calculation of PMI in Text Data In NLP, PMI is used to calculate the association between two words. For two words w_1 and w_2 , is calculated as:

$$\text{PMI}(w_1; w_2) = \text{Log}\left(\frac{P(w_1, w_2)}{P(w_1) \cdot P(w_2)}\right)$$

where:

- $P(w_1, w_2)$ is the probability of w_1 and w_2 co-occurring within a specified context (e.g., in the same sentence or within a certain window of words).
- $P(w_1)$ and $P(w_2)$ are the probabilities of encountering w_1 and w_2 independently in the text. These probabilities are usually estimated based on frequency counts from a large text corpus:

$$P(w_1, w_2) \approx \frac{\text{Count}(w_1, w_2)}{\text{Total Count}}$$

$$P(w_1) \approx \frac{\text{Count}(w_1)}{\text{Total Count}}$$

$$P(w_2) \approx \frac{\text{Count}(w_2)}{\text{Total Count}}$$

5.1 Interpretation of pmi

- **Positive PMI:** Indicates that the co-occurrence of X and Y (or w_1 and w_2) is more frequent than would be expected if they were independent. A higher PMI value suggests a stronger association.

- **PMI of 0:** Implies that X and Y occur together exactly as often as would be expected if they were independent.
- **Negative PMI:** Suggests that X and Y co-occur less frequently than expected under independence.

We use an n-grams model along with mean PMI of each sentence for ranking the sentences from most fitting to least fitting. The sentences are displayed in order of the highest average PMI values.

6 Models

Since we want to make an autocompletion model, we train the model on the “correct” endings provided by the Hellaswag training dataset. The initial preprocessing steps involve dropping the irrelevant columns (such as `ind`, `source_id`, `split` and `split_type`) and splitting the 4 ending options into separate columns (labelled A0, A1, A2 and A3 respectively). We then use the correctly provided label to find the “correct” ending and concatenate it with the context sentence provided to create the final sentence. All columns other than ‘activity_label’ and the ‘Sentence’ columns are dropped. The sentences are converted to lowercase and irrelevant words such as ‘[header]’, ‘[title]’, ‘[step]’ and ‘[substep]’ are dropped. The words are lemmatised and different n-gram (unigram, bigram, trigram, quadgram, pentagram, hexagram) tokens are created and their respective frequencies are calculated.

Input text when passed to a group of N-gram models which generate an array of predicted outputs. We then pass all these new predicted texts as input texts to the N-gram models to generate a concatenated output for each of the input texts. We repeat this process until the required length of the sentence is reached (can be changed in code). This is sort of a brute force approach where we exhaust all possible outputs that can be generated and use them further as inputs to find the best sentence possible. In case a word is not present in the vocabulary of the model, we continue prediction by randomly appending a word which does not change the context of the sentence. Since our task is of sentence completion, we separate out the sentences which have ended with a fullstop (or a ‘<e>’ character) from those that still have scope to be completed. It is important to note that the sentences without a fullstop will eventually terminate with one given we allow the length of the sentence to increase. However, due to lack of resources (time and computational) we run the model for only 5-7 words. We finally rank the appropriateness of each sentence by calculating the PMI (Pointwise Mutual Index) of each sentence.

7 Analysis

In most cases, the model provides us with a more or less sensible sentence that can be understood by most people. While the sentence may seem odd due to the nature of the corpus, the sentence structure remains intact according to grammatical rules and makes sense. We noticed however, that in some cases, sentences which are more appropriate had a lower PMI value as compared to those that didn’t (for example, a sentences which should have a period at the end had a lower PMI value than the same sentence without a period). We also performed Laplace smoothing (add-one smoothing) to counter situations where the individual words in a particular gram are present in the vocabulary but their counts cannot be used for the estimation of probabilities (since they may be 0). In encountering words that are completely unknown to the corpus, instead of using ‘<UNK>’ tokens to denote unknown words, we instead appended an unbiased known word which allowed the model to continue prediction while not changing the meaning of the original sentence. We chose this approach since this provided us with a clean and simple solution while not worrying about the issues of using <UNK> tokens such as loss of specificity and skewing the language model. There are however, several challenges associated with using an n-gram model for the autocompletion task:

1. **Storage and Computation:** Larger n values require significantly more storage and computational resources, as the model needs to store and process a higher number of possible n-grams. Moreover, the computational complexity of our model increases almost exponentially when increasing the number of words required in a sentence.

2. **Context Limitation:** N-grams consider a limited context. For instance, trigrams only look at the immediate two preceding words, which might not be sufficient for understanding the full context of a sentence.
3. **Markov Assumption:** N-grams rely on the Markov assumption, which posits that the probability of a word only depends on a limited history. This assumption simplifies computation but can limit the model's ability to capture longer dependencies in text.

8 Results

While there are no quantitative performance metrics for our task, we did conduct a small initial analysis of the performance of various n-gram models on the validation set of the HellaSwag dataset. The bigram model achieved an accuracy of 26.07% while trigrams, quadgrams, pentagrams and hexagrams achieved between 25.0%-25.2%. This is only slightly better than random guessing which is the baseline. In contrast, the various deep learning models achieved only around 30% accuracy. In fact, a model trained by University of South Florida and Oklahoma State University used GPT which achieved an accuracy of 25.8%. Our simple bigrams model, outperformed the same. This can be evidence of the fact that a simple n-grams model may be able to perform reasonably well on the given task. For our specific task, there is no quantitative metric for evaluating the generated outputs. We rely on the readers/evaluators to judge how well our model is able to generate text in this limited environment.

References

- [1] Rowan Z., Ari H., Yonatan B., Ali F., & Yejin C. (2019) HellaSwag: Can a Machine Really Finish Your Sentence? . ACL 2019
- [2] Rowan Z., Ari H., Yonatan B., Ali F., & Yejin C. (2019) HellaSwag Dataset . Kaggle.
- [3] Geoffrey Z. & Chris J.C.B. (2012) A Challenge Set for Advancing Language Modeling ACL/SIGPARSE
- [4] Geoffrey Zweig, John C. Platt, Christopher Meek, Christopher J.C. Burges, Ainur Yessenalina, and Qiang Liu. (2012). Computational Approaches to Sentence Completion. In Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 601–610, Jeju Island, Korea. Association for Computational Linguistics.
- [5] Tomáš Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean. (2013) Efficient Estimation of Word Representations in Vector Space. ICLR
- [6] Aubrie Woods. (2016). Exploiting Linguistic Features for Sentence Completion. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pages 438–442, Berlin, Germany. Association for Computational Linguistics.
- [7] Joseph Gubbins and Andreas Vlachos. (2013). Dependency Language Models for Sentence Completion. In Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, pages 1405–1410, Seattle, Washington, USA. Association for Computational Linguistics.
- [8] Thomas G. Dietterich (2000). Ensemble Methods in Machine Learning. International Workshop on Multiple Classifier Systems 2000, pages 1-15