«Ciceroniana On Line» V, 2, 2021, 285-296

TODD COOK

WHAT WOULD CICERO WRITE? — EXAMINING CRITICAL TEXTUAL DECISIONS WITH A LANGUAGE MODEL

Textual corruptions, ambiguous variations, and lacunae in manuscripts have been traditionally dealt with by using expert insight and conjecture. Emending or refining a text is a challenge for budding and seasoned philologists alike, and most practitioners welcome a second opinion.

Recent developments in language modeling now allow users to predict the probability of different sentences and to predict missing words more accurately than before. This new information and perspective can be used to form judgments on novel textual emendations and to further quantify existing historical editorial judgments. Additionally, this technology may be pedagogically useful for generating cloze test completion exercises to calibrate a student's Latin word sense for critical judgments.

This paper shows how classical scholars can use a transformer language model to gain another perspective on critical textual decisions. We will explain the importance of analyzing an author's corpus, and the impact of the Good-Turing theory of frequency estimation when predicting missing words. These two components together will lead us to two novel extrinsic metrics that users should employ to evaluate the suitability of a language model for a particular author, and to understand how many predictions for each missing word should be considered. We will also explain some of the current limitation of transformer language models. The large corpus of Cicero provides us with an excellent baseline to evaluate the quality of the predictions provided by a general Latin language model.

1. *About Language Models*

A language model is a statistical learning model that has been trained on words and sentences so that it is able to provide probabilities of words in context, and by extension it can predict a missing or masked word. Language models have been around for many years in several forms (Markov Chains; Word Embeddings such as Word2Vec and GloVe and KenLM; N-gram probability tables, etc.) however, they all have had problems working with more than a relatively small window of context, usually 5 words. In contrast, transformer language model designs, have achieved state-of-the-art performance on language modeling and masked word prediction tasks by being able to use up to 509 tokens/words of a sentence's context. This paper will focus on practical applications of transformer models for classicists; for technical explanations of the architecture and theoretical explanations as to why transformer language models are able to make better predictions, see the select bibliography[1].

Recent work by Y. Assael, T. Sommerschield, and J. Prag[2] has shown the value of creating and applying a deep learning language model to correct characters in Greek epigraphy. Their model uses a bi-directional LSTM architecture to model and predict sequences of characters, whereas in this paper we will examine the use of the transformer model architecture with its focus on whole words in the context of a single sentence.

Transformer language models for Latin have been recently developed and released for general use, notably Latin-Bert[3], which was trained using more post-classical Latin than classical Latin (today's Latin Wikipedia, OCR scans of 19th century scholarship, etc.). It was evaluated by sampling similar texts, and it was published with general metrics.

While researching this paper, a transformer Latin language model was created, Cicero-similis, using the texts of Cicero and other classical authors, and we provide metrics specific to Cicero. We will now explain how we formulated the metrics for evaluating this language model specifically for readers of Cicero.

2. *Single Word Probabilities*

In Cic. *re p.* 6, 32, the word *animal* is found in A, Q, PhD manuscripts, and the word *anima* is found in the H^1 Macr. and codd. *Tusc.* manu-

[1] *BERT* 2018.
[2] Assael-Sommerschield-Prag 2019.
[3] Bamman-Burns 2020.

scripts[4]. When faced with variant readings it can be valuable information to know which phrasing is more likely to have been commonly used—and a language model can help determine this.

In figure 1, we show how one can obtain the probability values of two different words using the Python programming language. Although we don't want to burden readers with unnecessary technical details, this example demonstrates important points: the code is readily available; the commands are relatively straightforward; and the task can be easily done by most computers.

Here are the steps taken in figure 1: Import libraries; load a language model and its tokenizer; instantiate a FillMaskPipeline; mask the word we want to predict in our sentence (note the use of the special `[MASK]` token); predict; and find out where our two target words were predicted.

The FillMaskPipeline only provides single token predictions; to get the probabilities of longer words composed of multiple subtokens, the prediction process must be incrementally repeated.

```
    >>> from transformers import BertForMaskedLM, AutoTokenizer, FillMaskPipeline
    >>> tokenizer = AutoTokenizer.from_pretrained("cook/cicero-similis")
    >>> model = BertForMaskedLM.from_pretrained("cook/cicero-similis")
    >>> fill_mask = FillMaskPipeline(model=model, tokenizer=tokenizer)
    >>> fill_mask.top_k = 100
    >>> results = fill_mask("""inanimum est enim omne quod pulsu agitatur externo; quod autem est
[MASK], id motu cietur interiore et suo, nam haec est propria natura animi atque uis;
    quae si est una ex omnibus quae sese moueat, neque nata certe est et aeterna est.""")
    >>> predicted_token_ids = [datum['token'] for datum in results]
    >>> predicted_words = tokenizer.convert_ids_to_tokens(predicted_token_ids)
    >>> animal_idx, anima_idx = predicted_words.index('animal'), predicted_words.index('anima')
    >>> print(f"Using partial sentence, `animal` predicted at index: {animal_idx}; `anima` at {anima_idx}")
     Using full sentence, `animal` predicted at index: 8; `anima` at 27
```

Figure 1. Script showing language model predicting probabilities for different words in context

Now with just a little bit of code the transformer language model tells us how much more likely the word *animal* is than *anima*.

Often the likelihood given to a word can be correlated with how frequently it appears in the training data. However, this is not true in this case: in the training corpus the word *anima* appears with probability

[4] Powell 2006.

0.00016 and *animal* appears less often at 0.00005. A simpler language model (such as one constructed of N-gram probabilities) tends to predict words that are more common, whereas a good language model guesses missing words by using all of the surrounding words to build a context and form a prediction.

Every assessment by a language model needs to be assessed by a critic using the context of *many* sentences surrounding the text in question. When performing the fill-mask prediction task, while transformer models use all the tokens in a single sentence to build a context, they can only consider a single sentence at a time (extending the context used for predictions to multiple sentences is an area of ongoing research). Additionally, the value of a sentence's likelihood must be weighed by editor's view of how *lectio difficilior potior* applies to the particular situation.

3. *Lectio difficilior potior*

The principle of *lectio difficilior potior* ("the more difficult reading is the stronger one") is a central, guiding principle of textual criticism. Although language models can make impressive predictions, and at times they appear to display Natural Language Understanding (NLU), they cannot understand text; they can only suggest words that try to complete a grammatical thought. A good language model most often tends to predict words that are grammatical; this is because most of the training data does not contain errors, but rather contains grammatically correct statements. Informally, we might view a language model like a scribe's ear: during dictation a scribe may miss a beat and he may be tempted to fill in according to what he thinks he heard — and these second guesses are like the top predictions of a language model. Ultimately, a critical editor — not a language model — must decide, whether an easy word is appropriate given the surrounding context, or perhaps the author was trying to make a point with a more pithy, unusual word. Thus, when to apply *lectio difficilior potior* is a personal critical judgment, and using a language model may provide supplemental information that may be useful in categorizing which options are less likely.

4. *Sentence Probabilities*

Often critical textual decisions need to be examined beyond the single word level, and to help with these issues, a language model can be used to determine which of two whole sentences is more probable.

To find the probability of a sentence: Mask each word in turn; predict the missing token and accumulate the probabilities for each known token; multiple the probabilities to find the conditional probability of the whole sentence (see table 1).

Sentence with Masked Token	Probability of Missing Token
“[MASK] est ueritas ?”	0.00058710627490
“Quid [MASK] ueritas ?”	0.237660154700279
“Quid est [MASK] ?”	2.76185728580458E-05
“Quid est ueritas [MASK]”	0.00977017916738987
“Quid est veritas?”	3.7651029780095805e-11 or sum log probabilities: -24.002660810659727

Table 1. Probabilities of individual tokens in context

(Due to numerical precision issues with products of probabilities, it is common practice to use the summation of log probabilities when calculating sentence probabilities). Note: unlike previous language models, transformer and BERT models now typically include punctuation and can predict punctuation — since it helps to build the context of a sentence.

Calculating sentence probabilities allows us to move from individual word probabilities to groups of words. In this manner, given a sentence with variations due to word inversions (*e.g. potest malum* / *malum potest* [Cic. *Cato* 4])[5], it’s possible to determine which word inversion is more likely. Of course, it’s up to the critic to decide which is more appropriate given the author and the surrounding context. In fact, the critic must supply a perspective on the extended context; unfortunately, currently, BERT and transformer models can only predict masked words within a

[5] Powell 2006.

single sentence; providing the previous or following sentences do not improve masked word prediction accuracy.

5. *Evaluating a Language Model*

There are two types of metrics for evaluating a language model, intrinsic and extrinsic. An intrinsic metric is one that can be answered by the model and the training data. The most common intrinsic evaluation of a language model is perplexity, which measures how well a model learns to predict words in the training and test data. However, this value is only useful for machine learning engineers tuning a model or comparing different model architectures on the same dataset; perplexity scores don't offer any insights for classicists or lay users of a language model, so we will omit the technical details here, but for a deep explanation see Campagnola 2020.

An extrinsic evaluation of a model involves completing a task external to the model's training and test data, and most often we anticipate scholars will want to predict missing words for a particular author, and so we will now investigate and explain what data is necessary to support and develop these metrics.

The performance of any language model will vary for different authors, and it can even vary for different types of text from the same author. However, there are several measurements that can help provide objective metrics on a language model's performance. Let's frame the problem in practical terms:

If we found a new page of Cicero, how many new words should we expect to find?

This can encompass two types of new: as in never-seen-before in the author's corpus; and new as in new forms of inflections; words with previous lemma usage.

For Cicero, the answer is three words out of every hundred. We deduce this by applying a statistical technique to the author's corpus (see table 2). The Good-Turing frequency estimation technique states: «The probability of an unseen element is equivalent to the probability of elements seen only once»[6].

[6] https://en.wikipedia.org/wiki/Good%E2%80%93Turing_frequency_estimation.

Number of words seen once in Cicero:	34,608
Number of total words in Cicero corpus:	1,196,512
Probability of encountering a new word:	34608 / 1196512 = 0.0289

Table 2. Calculating Cicero's propensity for using a new word (Good-Turing frequency estimation)

Thus in the corpus of Cicero, there's a 3% chance of seeing a word that only occurs once. Find one hundred new words of Cicero? Expect to find three words never seen before. An alternative to the Good-Turing frequency estimation principle is to fit a curve on the nearby values and forecast a value for the next element, the count of unseen vocabulary words (see table 3 and figure 2).

Cicero's Unigram Frequencies

Number of Word Occurrences	Count
7	1,572
6	2,018
5	2,622
4	3,936
3	6,095
2	11,858
1	34,608
New words, previously unseen	(estimated by curve fitting) 88,198

Table 3. Cicero's unigram frequencies

Estimating Cicero's unseen words by curve fitting

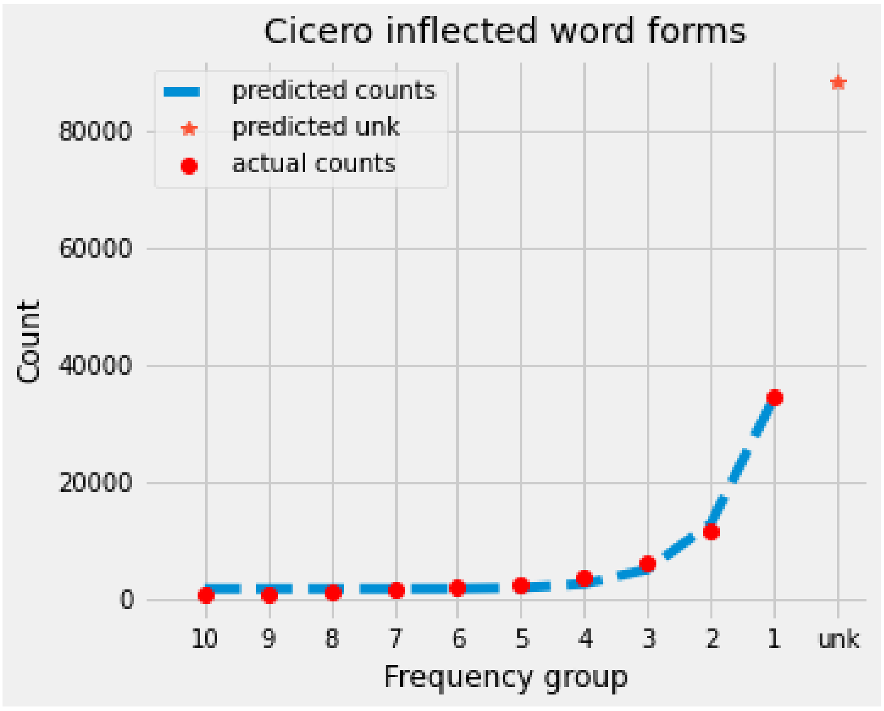

Figure 2. Visualization of a power law curve fitting a projection Cicero's unseen words

Using the tail of the distribution, we can fit a power law curve: Find 100 new words? Expect 7 to have never been seen. (We include power law curve fitting calculations here to show that even with liberal assumptions an upper-bound limit for new vocabulary words is quickly reached).

Regardless of which method one adopts to forecast the appearance of hitherto unseen words, either way, the important takeaway for scholars is that all models indicate we should expect to some percentage of previously unseen words to appear with any discovery of a new text, whether it is a new manuscript page or merely attempting to predict a missing word. By extension, when trying to predict missing words, or when trying to choose alternates among variations of an existing text, we would expect that some viable candidates are hitherto unseen in the author's corpus.

Thus, in practical terms, if you use a language model to predict a masked word for a text of Cicero, you should tolerate groups of results that stay within this expected range of new terms. A language model is performing well when it predicts words that are similar to the author's vocabulary, plus some margin for hitherto unseen words. A language model that performs poorly merely predicts too many words that we would not expect to see in the author *even if* a new page of work were discovered. The quality of the predictions must be evaluated on a case-by-case basis using the surrounding context and the editor's good sense.

6. *Model Training and Evaluations*

During research for this paper, a transformer language model was created, Cicero-similis, from Latin corpora (*Tessera, Phi5, Patrologia Latina*) consisting of 121.7 million tokens, and composed of 1,231,979 distinct tokens. The training texts were cleaned, regularized, lower cased, and enclitics were pre-tokenized. The model was trained for five epochs using hyperparameters tuned for the hardware used.

The model was evaluated using Cicero's sentences. First, the Tesserae corpus of Cicero was tokenized into 44,142 sentences, then 2,500 sentences were selected, with each sentence containing between 8 and 23 tokens. The reason for these size boundaries is that we wished that the model (and eventual human test subjects) would have a reasonable amount of context to use to make a prediction for the missing/masked token. For each test sentence, a whole word was randomly chosen and masked. Neither enclitics nor punctuation were allowed to be chosen as masked token values.

These chosen test sentences were then removed from the training corpus. Both datasets were inspected and no words became orphaned; all of the test sentences contained vocabulary that was present in the training data, but none of the test sentences appeared verbatim in the training data.

After many training runs and parameter adjustments, test results showed definitive trends: for 50% of the test sentences, the model was able to predict the missing word within its top 10 guesses, and 20% of the time, the correct word was the first prediction. Now, if these numbers seem low, remember, to predict the correct word the language model must choose from the 1,231,979 distinct words found in the training data.

As we have shown earlier, if we found a new page of Cicero (consisting of a hundred words), we would expect to find three words we had never seen before in his corpus; for the Cicero-similis language model, for 2,500 test sentences from the Cicero corpus, the model had a mean OOV (out of vocabulary) threshold of 66 (standard deviation 61), and the mean unseen word OOV threshold (3rd unseen word) occurred at the 138th prediction (standard deviation 90). In practical terms these metrics imply that if a Cicero scholar wishes to examine 95% of the reasonable predictions for a missing word, the scholar should consider the first 188 predictions of the model, and to view 99% of the reasonable predictions the scholar should consider the first 249 predictions. One can examine predictions beyond these thresholds, but the model becomes less and less confident and begins suggesting more words outside of Cicero's usage. The OOV threshold and the OOV unseen word threshold metrics need to be calculated for each author a language model is used on. The calculation of the metrics is straightforward, and they provide good guidelines on how many predictions to consider and how much confidence to have in those predictions.

All tests and tuning runs were done with the training and test data held out. Yet, in one isolated experiment, when the test data was added to the training data, performance only improved a little: the transformer model generalize well, it does not merely memorize answers, rather it gives us insight into what the word in the sentence should be, predicting as if it were, a collective unconsciousness of the Latin language.

7. *Conclusions*

When consulting a Latin language model about a textual decision, it's important to understand how well the model performs for that particular author. In this presentation, we have analyzed one language model (Cicero-similis) and seen that the infilling or fill-mask completion task returns the correct word 50% of the time in the top 10 for a test set of sentences from the corpus of Cicero. Additionally we have shown that this language model for Cicero has an OOV (out of vocabulary) threshold of 66 (mean) and the threshold for unseen vocabulary occurs after 138 predictions (mean). The first word OOV threshold value and the 3rd word OOV threshold value give users a representation of how fast the model's performance degrades.

A scholar should evaluate the use of a language model for a particular author using two metrics based on tests using sentences from the author's corpus: 1. How well does the model perform on the fill-mask/infilling task, 2. What is the OOV prediction threshold for those test sentences — that is, when the model provides a list of predictions for a missing word (in descending order of probability) how far down the list does the first word appear which has never appeared in the author's corpus usage. The first metric tells the scholar how often the model is right in the top ten predictions, and the second metric tells the scholar how many predictions the model provides for an author before it wanders into the territory of wholly unexpected words.

Areas of further research include establishing a human baseline for the Latin fill-mask/infilling prediction task; training a model with an optimal sub-tokenization scheme; better regularization of the training data; more data, etc. Further work is also planned to provide supporting code to make using language models easier for classicists, and to better handle multi-token predictions. The author of this paper is open to sharing the code used to generate this language model and he is committed to helping others be able to use and assess language models for scholarly and pedagogical purposes.

While looking forward here, the author would like to look back and thank Dr. Ermanno Malaspina for his comments and advice made during a presentation of a draft of this paper about pursuing the *lectio difficilior potior* aspect; and thanks to Dr. Kyle P. Johnson for his feedback and encouragement.

Overall, results will continue to improve with further research. Nonetheless it is clear that these newer language models now demonstrate the beginnings of Natural Language Understanding (NLU) enough that the classical community should assess their value, use and applications in earnest.

Bibliography

Assael-Sommerschield-Prag 2019: Y. Assael, T. Sommerschield, J. Prag, *Restoring ancient text using deep learning: a case study on Greek epigraphy,* October 2019, https://arxiv.org/abs/1910.06262.

Bamman-Burns 2020: D. Bamman, P.J. Burns, *Latin BERT: A Contextual Language Model for Classical Philology*, 21 September 2020, https://arxiv.org/abs/2009.10053.

BERT 2018: J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, *Pre-training of Deep Bidirectional Transformers for Language Understanding*, 11 October 2018, https://arxiv.org/abs/1810.04805.

Campagnola 2020: C. Campagnola, *Perplexity in Language Models*, 2020, https://chiaracampagnola.io/2020/05/17/perplexity-in-language-models

https://en.wikipedia.org/wiki/Good%E2%80%93Turing_frequency_estimation

https://en.wikipedia.org/wiki/Transformer_(machine_learning_model)

https://github.com/kpu/kenlm

Powell 2006: M. Tulli Ciceronis, *De Re Publica, De Legibus, Cato Maior de Senectute, Laelius de Amicitia*, recognovit brevique adnotatione critica instruxit J.G.F. Powell, Oxford 2006.

Resources

Cicero-similis Latin language model, https://huggingface.co/cook/cicero-similis.

Classical Language Toolkit, http://cltk.org.

Patrologia Latina Latin corpus, https://github.com/OpenGreekAndLatin/patrologia_latina-dev.

Supporting Notebooks, https://github.com/todd-cook/ML-You-Can-Use/tree/master/probabilistic_language_modeling.

Tesserae Latin corpus, https://github.com/cltk/lat_text_tesserae.