Disambiguating Voynich Manuscript transliterations with word embeddings

Jirka Lhotka, Francesco Salvi, Liudvikas Lazauskas
Department of Computer Science, EPFL, Switzerland

Supervised by Lonneke van der Plas
Idiap Research Institute, Switzerland

***Abstract*—Voynich manuscript is a 15th century document written in unknown language and script. Consequently, reading and transliterating it to electronic form is equivocal and leads to ambiguities. We present a way to disambiguate said uncertainties using word embeddings models, which achieves accuracies up to 86% on artificially corrupted texts of the same size.**

I. Introduction

A. Voynich manuscript

The Voynich manuscript is a mysterious medieval document containing illustrations of mostly unidentifiable plants, astronomical or astrological diagrams, as well as over 170,000 characters over about 37,000 words written in an unknown script [1]. Historically, many have argued that the text was a forgery intended as a hoax and that it did not in fact carry any meaning; however, recent studies of Voynich found it to have many properties characteristic of real languages (such as the token distribution satisfying Zipf's law), suggesting the "hoax hypothesis" is rather unlikely [2]. Using radiocarbon dating, the manuscript was dated to sometime between 1404 and 1438, with high accuracy [3].

B. Transliteration ambiguities

Despite the script of Voynich being unknown, several ad-hoc alphabets have been designed to represent it in digital form. The most widely used of those is the "Extensible Voynich Alphabet" (EVA) [4], which includes 26 basic characters and numerous rare extended characters.

One problem of studying Voynich is that the process of transforming the text from its handwritten original form to a digital machine-readable format (*transliterating*) is equivocal. Since the script and the language are both completely unknown, we do not have any information about the alphabet or the semantics of the text that could help us distinguish whether two similar signs are distinct characters or not. Consequently, the existing transliterations include unresolved ambiguities, which come in four main forms:

i) *Uncertain spaces* (8.14% of all spaces) – Denoted with as ",", it indicates an unclear word separator, e.g. "qo,kain" signifies it is unclear whether the script reads "qo kain" or "qokain".
ii) *Alternate readings* (0.413% of all characters) – Denoted as "[X:Y]" where X and Y are sequences of characters of possibly different length, it indicates 2 possible readings of the text, e.g. "qo[ka-h]in" signifies it is unclear whether the script reads "qokain" or "qohin".
iii) *Single uncertainty* (0.112% of all characters) – Denoted as "?", it indicates a character that we cannot read, e.g. "qo?ain" signifies any character in the alphabet could replace "?", leading to a wide range of possible readings.
iv) *Uncertain sequence* (0.016% of all characters) – Denoted as "???", it indicates a sequence of characters that we cannot read of uncertain length, e.g. "qo???ain" could mean "qoaain" as well as "qoshshkaain" and infinitely many other possible readings.

The goal of our work is to resolve these ambiguities by picking the best allowed alternative for each ambiguous word. For example, given the ambiguous word "vir,t[a:u]tem", our task is to pick the best among the allowed alternatives "vir tatem", "vir tutem", virtatem", "virtutem".

II. Methods

A. Word embeddings

Word embeddings are numerical (vector) representations of words based on the idea that the context a word appears in encodes its meaning [5]. They provide a way to quantify word similarity as well as to predict the most probable word to appear in a given context.

There are multiple techniques to calculate word embeddings. Due to the scarcity of our data, the newest Transformer-based models are not suitable for our task, and we instead consider two other options – *Word2vec* and *FastText* [6]. Both of those further provide two options for creating embeddings called *SkipGram* and *CBOW* (Continuous Bag of Words).

1) ***SkipGram and CBOW:*** are two tasks commonly used to learn word embeddings. In CBOW, the task the model is trained on is to predict a word based on its context (the words that surround it in a sentence); while in SkipGram, the task is essentially reversed and the goal is to predict the context given the middle word. Using the phrase ”Cornelius sub arborem sedet” as an example, a CBOW task would be to predict the word ”arborem” given the context ”Cornelius”, ”sub” and ”sedet”, while the respective SkipGram task would be to predict words ”Cornelius”, ”sub” and ”sedet” given the word ”arborem”. The number of words considered from each side of the target word defines the *window size*, where a size of 1 means that the context will contain at most 2 words – the direct left and right neighbours of the target word.

2) ***Word2vec:*** is the earliest successful word embedding architecture. It works by first encoding the entire vocabulary (all existing words) into one-hot vectors, and then training a neural network with 1 hidden layer to perform either the SkipGram or the CBOW tasks. After training, the vector of hidden layer weights corresponding to each word in the vocabulary constitutes its embedding [7].

3) ***FastText:*** can be seen as an extension of Word2vec, including also subword information [8]. It performs the same tasks using character n-grams of different lengths instead of full words, which are then combined to create word embeddings. This can lead to better performances, especially for rare words, as morphosyntactic information is also incorporated into the vectors [5]. Moreover, using n-grams enables FastText to compute embeddings for out-of-vocabulary words, which were not encountered during training.

B. Text corruption

As the correct transliteration for Voynich is unknown, we cannot evaluate our models on it directly. To circumvent this limitation, we train them on two medieval texts of similar size from the same period: the Italian text Inferno, part of the Divine Comedy by Dante Alighieri [9], and the Latin text Historia Hierosolymitanæ Expeditionis, by Albert of Aix [10], which we reduce to one fourth of its original size to make its length analogous to the Voynich Manuscript.

To build models with conditions comparable to Voynich, we corrupt the two texts adding artificial uncertainties, reproducing those of the manuscript. For practical purposes[1], we ignore uncertain sequences (’???’), removing also words with such uncertainty from the Voynich text. After corruption, we obtain two corrupted texts containing ambiguities whose correct solutions are known, allowing us to quantitatively evaluate our models. To corrupt the text, we randomly pick letters to corrupt and change them to ambiguities found in Voynich with a probability that preserves the distribution of said ambiguities in the manuscript. This corruption is done under the simplifying assumptions that (1) ambiguities are distributed uniformly, i.e. all letters are equally likely to become an ambiguity (2) uncertain spaces are an actual space in 50% of the cases (3) alternate readings always include the correct option. Applying this to our Latin and Italian text then yields Voynich-like ambiguities, e.g. ”virtutem habet siccatoriam” gets corrupted to ”virt,utem ha[u:b]et sicca?oriam”.

C. Synonym selection

To obtain a first intuition of the quality of the embeddings, we analyze how well they perform in the task of predicting synonyms of target words.

1) Italian: For Italian, we empirically analyse nearest neighbours of medium-frequency words seen in training. Specifically, we train Word2vec and FastText models with standard parameters, both for CBOW and SkipGram, on the corrupted Inferno with uncertain words removed. We then randomly select 10 words with frequencies between 5 and 8, which corresponds roughly to a sample between the 88th and the 93rd percentile of the word distribution. For each such word, we manually inspect words that the models output as its closest neighbours (by cosine similarity) to get an initial understanding of the models’ performance.

2) Latin: For Latin, we can rely on a benchmark specifically built for the task of synonym selection [11], giving a more accurate quantitative evaluation. The benchmark consists of 2759 TOEFL-like sets of 5 words: a target word and 4 candidate synonyms, among which one is an actual synonym and three are decoys. For each set, the cosine similarity between the embedding of the target word and each of the embeddings of the candidate synonyms are computed, picking then the

[1]Uncertain sequences are very rare, but at the same time most difficult for our approach to resolve as they vastly enlarge the size of our vocabulary, which would make our models perform poorly.

candidate with the highest similarity. Again, we train our models with standard parameters and removing all the uncertain words from the corrupted Latin text. For this task, the models need to compute the embeddings of words in the benchmark, possibly never encountered during training. Since Word2vec models cannot compute out-of-vocabulary embeddings, only FastText models can be used with this benchmark.

III. Resolving ambiguities

A. Baseline

In order to evaluate our models, we first create a simple baseline that randomly replaces each uncertainty according to the character distribution in the text, where each character occurring in an alternative reading counts as half an occurrence. For instance, given the uncertain word "virt[a:u]tem", if the text had a letter frequency of 10% for "a" and 5% for "u", the prediction would be with probability 66.6% "virtatem" and with 33.3% "virtutem". Uncertain spaces are replaced by an actual space or by an empty string with equal probability.

B. Prediction models

When defining our models, we consider two ways to deal with uncertainties in training and in context individuation, as well as two prediction-generating techniques, leading to four possible set-ups. Taking the sentence "c?rnelius sol,us sedet" and the task of resolving "sol,us" as an example, we first consider two options for dealing with uncertainties:

1) **Removing all uncertain words**, training the model and individuating contexts only using words without ambiguities. In the example sentence, the model would be trained only with "sedet", and the uncertain word sol,us would also only have "sedet" as context.
2) **Replacing all uncertainties** with a single special character "£", that does not otherwise appear in the texts, and use all the resulting text both for training and for context individuation. In the above example, the model would be trained on "c£rnelius", "sol£us" and "sedet", and the context of "sol,us" would be "c£rnelius" and "sedet". This has the added benefit of not throwing away valid character n-grams of ambiguous words, which are potentially useful for FastText.

Additionally, there are two ways of generating a prediction for each uncertain word. In both cases, we start by summing the embeddings of all the words in the context to obtain a centre *guess embedding*, corresponding to the missing word. Then, we compare the alternatives with the guess embedding, and pick the best one. To rank them, we can either:

A) Use **cosine similarity**, computing the similarity between the centre embedding and the embeddings of all the alternatives. In this case, embeddings corresponding to words out-of-vocabulary are computed, so only FastText models are viable.
B) Use **softmax probabilities**, propagating the centre embedding to the output layer using the learned weights to get the probabilities of all the alternatives directly from the model. For this, it is strictly necessary that all the alternatives appear in the vocabulary during training, so that a neuron can be created for each of them and its corresponding weights updated. Hence, for this type of model, a list including all the alternatives of all the uncertain words is created and added to the vocabulary before training.

Note that for alternatives composed of a multi-word sequence, the final similarity/probability is the average of the similarities/probabilities of each word in the sequence, where the similarity/probability of each word in the sequence is computed with other words from the sequence included in the context. For example, we compute the probability of an alternative "sol us" by averaging the probability of "sol" with "us" in the context and vice versa.

Combining these two choices results in 4 possible types of models: 1A, 1B, 2A and 2B. Even though models of type B could also allow the use of Word2vec, we only use FastText for this task, as initial tests showed it to have vastly superior performance on data of this size (cf. Section IV-A).

C. Hyperparameter search

For each of the models described in Section III-B, the following hyperparameters must be chosen: minimum word frequency df, embeddings vector size v, window size w, number of noise words for negative sampling n (cf. [12]), and learning rate α. All the models were trained on 20 epochs with char n-grams between 2 and 6 for FastText. For each type of model, the choice between CBOW and SkipGram has been treated as an additional hyperparameter sg, corresponding to 0 for CBOW and to 1 for SkipGram. For each model, we choose the best hyperparameters by running grid-search optimization, computing the accuracy over the different types of alternatives on the artificially-corrupted Inferno, and picking the set of hyperparameters that achieve the best overall accuracy.

IV. Results

A. Synonym selection

1) Italian: The results of the analysis described in Section II-C1 show that Word2vec performs very poorly with data of this size in both CBOW and SkipGram, not being able to learn meaningful synonyms for any of the selected words. FastText CBOW performs only slightly better, learning meaningful neighbours for only a handful of words, but FastText SkipGram achieves far superior performance, learning at least one synonym for almost all the words. All the synonyms learned, though, have a similarity in form, which FastText can recognize since it accounts for character n-grams. In fact, we observe that a high number of nearest neighbors for any word are variations of the same syntagma, morphologically but not semantically related. For example, the 5 nearest neighbors of 'regno' are 'tegno', 'ngegno', 'indegno', 'ingegno', 'stagno', all building on 'egno'/'gno' but with completely different meanings.

2) Latin: Following the procedure outlined in Section II-C2, we obtain an accuracy of 45.7% for CBOW and 57.4% for SkipGram, confirming both our results from the Inferno, i.e. that SkipGram performs better than CBOW in learning synonyms and that, overall, the performances are still rather poor with this data size. As detailed in Section II-C2, Word2vec cannot be evaluated with this method.

B. Resolving ambiguities

	Base	1A	1B	2A	2B
df	-	5	2	1	5
v	-	300	300	20	100
w	-	8	5	5	5
n	-	20	20	20	20
α	-	0.05	0.05	0.05	0.05
sg	-	0	0	0	0
Accuracy	0.495	0.466	0.859	0.461	0.856
Acc. AR	0.743	0.564	0.848	0.558	0.827
Acc. SU	0.069	0.081	0.579	0.132	0.579
Acc. US	0.458	0.455	0.872	0.45	0.874

TABLE I

Best hyperparameters and accuracies for the Baseline (Base) and the 4 types of models. AR stands for Alternate Reading, SU for Single Uncertainty and US for Uncertain Space.

The results of hyperparameters optimization, as described in Section III-C, are shown in Table I. The results clearly demonstrate that models using softmax probabilities (type B) perform strongly better, achieving an overall accuracy of about 86%, while alternative models of type A underperform the Baseline with an accuracy of about 46%. The choice between type 1 or 2, instead, does not seem to have a particular impact, with very similar results for both 1A/1B and 2A/2B. When looking at the individual types of ambiguities, SU achieves the lowest accuracy, which is expected as these have the most alternatives to take into account. Models of type B perform slightly better at disambiguating US than AR, while the opposite is true for models of type A. Finally, the Baseline is particularly accurate in resolving AR, which is not surprising since the way the texts are corrupted (cf. II-B) is such that letters with higher frequencies will be corrupted more frequently.

With respect to hyperparameters, CBOW models come on top, which is unsurprising as they are trained on the very same task. In general, despite the shortness of the text, a high minimum word frequency seems to be preferred, avoiding frequencies of 1 which would include all kinds of rare words.

C. Voynich ambiguities

Ambiguous word	Prediction (lines)	Prediction (paragraphs)
[cth:oto]res	otores	otores
or,y	ory	ory
cthar,dan	cthardan	cthardan
oteo[s:r],roloty	oteorroloty	oteorroloty
cthiar,daiin	cthiardaiin	cthiardaiin
sair,y	sair y	sairy

TABLE II

Sample of predictions for Voynich ambiguities.

Using model 1B with the hyperparameters reported in Table I, we apply our pipeline to the Voynich manuscript, picking an alternative for each uncertain word. Since the Voynich lacks any form of punctuation, it is not clear what constitutes a sentence. Therefore, we applied our model in two different ways: considering lines or paragraphs as sentences. Full results are accessible online, and a sample is reported in Table II.

V. Conclusion

Using FastText with softmax probabilities, our model achieves an accuracy of 86% on the artificially corrupted Inferno, outperforming the baseline by 36 percentage points. While this result is promising, not all the simplifying assumptions employed (cf. Section II-B) might hold for the Voynich manuscript. In particular, Voynich ambiguities tend to to be clustered in specific portions of the text, with certain characters more prone to uncertainties than others. Examining how our model performs when these assumptions are relaxed would be thus an interesting possibility for future work.

References

[1] Andreas Schinner. “The Voynich manuscript: Evidence of the hoax hypothesis”. In: *Cryptologia* 31.2 (2007), pp. 95–107 (cit. on p. 1).

[2] Marcelo A. Montemurro and Damián H. Zanette. “Keywords and Co-Occurrence Patterns in the Voynich Manuscript: An Information-Theoretic Analysis”. In: *PLOS ONE* 8.6 (June 2013), pp. 1–9. DOI: 10.1371/journal.pone.0066344. URL: https://doi.org/10.1371/journal.pone.0066344 (cit. on p. 1).

[3] *Radio-carbon dating of the Voynich MS*. URL: http://www.voynich.nu/extra/carbon.html (cit. on p. 1).

[4] *EVA alphabet*. URL: http://www.voynich.nu/transcr.html#Eva (cit. on p. 1).

[5] William Merrill and Eli Baum. “Voynich2Vec: Using FastText Word Embeddings for Voynich Decipherment”. In: () (cit. on pp. 1, 2).

[6] Ashish Vaswani et al. “Attention Is All You Need”. en. In: (June 2017). URL: https://arxiv.org/abs/1706.03762v5 (cit. on p. 2).

[7] Tomas Mikolov et al. “Efficient Estimation of Word Representations in Vector Space”. In: *arXiv:1301.3781 [cs]* (Sept. 2013). arXiv: 1301.3781. URL: http://arxiv.org/abs/1301.3781 (cit. on p. 2).

[8] Piotr Bojanowski et al. “Enriching Word Vectors with Subword Information”. In: *arXiv:1607.04606 [cs]* (June 2017). arXiv: 1607.04606. URL: http://arxiv.org/abs/1607.04606 (cit. on p. 2).

[9] Dante Alighieri , 1265-1321. *Divina Commedia di Dante: Inferno*. Italian. Project Gutenberg, Aug. 1997. URL: https://www.gutenberg.org/ebooks/997 (cit. on p. 2).

[10] Albert of Aix , 1095-1125. *Historia Hierosolymitanae Expeditionis*. URL: https://www.thelatinlibrary.com/albertofaix.html (cit. on p. 2).

[11] Sprugnoli Rachele, Marco Passarotti, and Giovanni Moretti. “Vir is to Moderatus as Mulier is to Intemperans. Lemma Embeddings for Latin”. en. In: (2019). DOI: 10.5281/ZENODO.3565572 (cit. on p. 2).

[12] Tomas Mikolov et al. “Distributed Representations of Words and Phrases and their Compositionality”. In: *Advances in Neural Information Processing Systems*. Ed. by C. J. C. Burges et al. Vol. 26. Curran Associates, Inc., 2013 (cit. on p. 3).