Structured Nonsense: Automatic Imitation of Writing Style

Misha Denil
Department of Computer Science
University of British Columbia
201-2366 Main Mall
Vancouver BC
V6T 1Z4
mdenil@cs.ubc.ca

Abstract

In this report we present a system designed to generate nonsense sentences in a specified style. A style is defined implicitly by providing a set of training documents (e.g. books written by a single author) and the system extracts the necessary information to mimic structural properties of the text. We quantify our success in the style mimicry task by measuring the ability of our system to fool a standard stylometric evaluation tool.

1 Introduction

Most research related to stylometry focuses on either identifying the authorship of texts (the most famous example being the Federalist Papers) or on classifying texts by genre. There has been very little work on attempting to generate novel texts with a specified style.

In contrast, the majority of work on text generation focuses on transforming a knowledge representation into words, and these generation systems are typically produced for domain specific problems. By choosing a domain to operate in the researchers are essentially deciding on a predetermined style before the system is built. A text generation system might turn knowledge into text using a predefined set of rules; the style of the resulting output is then a product of the rules themselves, which have been crafted for the problem at hand.

In this report we take the view of (Argamon and Koppel, 1998) that the topic (i.e. semantic content) of a document is independent of the style and attempt to build a system which is capable of generating documents in a specified style while ignoring issues of semantics entirely. That is, our system produces nonsense sentences which are intended to be nonsense in the style of the documents it was trained on.

2 Related work

Our goal in this work is to build a generative model of language which mimics style. Although there is very little work which addresses this problem directly, there is a significant body of similar work which we can draw on for techniques and direction.

There has been some recent work which looks at how standard authorship attribution techniques preform when presented with maliciously crafted data (Brennan and Greenstadt, 2009). This work is useful to us since it examines the robustness of stylometric techniques to mimicry; we make use several of these standard techniques in our evaluation, and it is important to consider how robust they are in general. In a similar vein, (Somers and Tweedie, 2010) look at the ability of standard stylometric techniques to correctly identify a pastiche as being written by an author other than the one being imitated.

Kacmarcik and Gamon study a related problem where they look at the feasibility of automatically obfusticating the stylometry of a document (Kacmarcik and Gamon, 2006). Their success in this task suggests that our task may be quite difficult, since stylometric techniques can be sensitive to a small number of changes. Nonetheless we find that this is not the case in this report.

There is a large body of work in the NLP literature on classifying document style. Some representative works in this area are (Finn and Kushmerick, 2006) and (Argamon and Koppel, 1998) which both attempt to learn style classifiers in a variety of domains.

A 2005 study (Uzuner and Katz, 2005) investigates how different language models can be used to identify authorship and discuss which types of standard features are suitable to generalize from characteristics of a document to characteristics of the author.

3 Contribution

The work presented in this report differs from previous work in the sense that no one else has focused on gener-

ating work in a given style. Our work is also different in that we focus only on the stylistic aspects of generation and eschew the complications introduced by attempting to model semantics. This makes our models very simple compared to other text generation systems, and allows us to focus our efforts on style generation exclusively.

4 Corpus

The corpus we chose to use for this report is a collection of books[1] by 12 different authors, selected from Project Gutenberg. The books are provided as plaintext, so minimal pre-processing was required to put them in a suitable format. The text files provided by Project Gutenberg typically have a block of front and end matter that is added by the project administrators; since the exact format of this extra material can vary depending on the book, we removed it manually from each file.

For each of the 12 authors we selected between three and seven of their works for this project. The authors were chosen to represent a variety of styles of writing and contain works of non fiction as well as fiction from a variety of genres.

5 System

5.1 Architecture

The system in this report focuses primarily on sentence generation. We made the choice to focus on this level of detail because within a document, sentences are the most structurally rich elements. We also hypothesize that sentences are the most important structural element of a document to model well in order to produce a readable result. Although we will not be able to test this hypothesis directly, it has a significant amount of intuitive appeal.

The system we develop here has four phases:

1. **Annotation** In this phase, the raw data is annotated either by attaching POS tags to each word or by finding a parse tree for each sentence.

2. **Learning** In this phase, parameters for the different models are selected using the annotated data as input.

3. **Generation** In this phase, the trained models are used to produce novel sentences.

4. **Realization** In this phase, the sentences produced during generation are modified to improve readability and aesthetics.

The annotation phase is preformed using existing NLP software, and the realization phase is very simple. We will discuss the specifics of both of these phases in the implementation section, but for now we focus on the function of the modules used in the learning and generation phases.

In our system, learning and generation dual in the sense that during learning data is fed into a model which adjusts its parameters in response to this input, and generation involves running this process in reverse by using the existing model parameters to produce data which the model would expect to see as input. Reversibility is an important requirement of all the models in this system so we will discuss the models themselves and it should be understood that the learning phase involves selecting parameters for the models and the generation phase involves running these models in reverse to sample from the learned distributions.

The **content model** is responsible for modeling how words relate to one another within a single sentence. Proximity relationships between words can be very complex. One approach to deal with this complexity is to make the content model very sophisticated in order to capture these relationships. We attempt to manage this complexity by decomposing the problem into two parts. In our system the content model is responsible for selecting individual words but it does so under externally imposed structural constraints which depend on the context of the word being selected. We hypothesize this approach will allow us to make use of a very simple content model since the higher level structural constraints on content are modeled separately.

The **structure model** is responsible for modeling sentence structure. We use the output of the structure model to constrain the generation of content. The structure model imposes grammatical and syntactic constraints on the content model to ensure that, e.g. the resulting sequences of words are grammatical sentences.

Throughout this report it will be useful to refer to the structure model and the content model separately. We will refer to the complete system which combines the two models into a single system as the **combined model**.

5.2 Implementation

In the previous section we discussed the roles that different phases of generation play within our system in a general sense. In this section we describe specific modules which were implemented for each phase. Although the requirements of each phase general enough to be satisfied by many different implementations, we implement only one module for each phase in this report.

5.2.1 Annotation

The annotation phase augments the input data with additional structural information which is exploited during learning and generation. We used the Stanford

[1]The corpus is made available with the source code for our system. See Appendix A for instructions on how to obtain it.

Parser (Klein and Manning, 2003) and the Stanford POS Tagger (Toutanova and Manning, 2000) to produce structural annotations for our models. In principle the Stanford Parser would be sufficient since it produces POS tags for each word as part of the parse trees it produces; however, for practical reasons it is helpful to use both tools since the tagger is significantly faster when only POS tags are needed.

5.2.2 Learning and Generation

Content Model The content model used in this report is based on building empirical distributions of n-grams. The implementation is general, in the sense that an arbitrary n can be selected at training time.

Rather than recording n-grams of words directly, in this model we actually record n-grams of (word, POS tag) pairs. The reason for this is twofold: first, the attached POS tag serves as a primitive form of word sense disambiguation. The second and more important role of the POS tags is that they provide a mechanism for the content model to be constrained by the structure model. As discussed below, the structure model produces a sequence of POS tags which we call the sentence skeleton. The content model is used to select a word to replace each tag.

More formally, the content model encodes the joint likelihood $P(w_1, t_1, \ldots, w_n, t_n)$ where w_i is the ith word in the n-gram and t_i is the corresponding POS tag. When filling in a sentence skeleton provided by the structure model, we sample each word in the sentence from left to wright using $P(w_m|t_m, w_{m-1}, t_{m-1}, \ldots, w_{m-n+2}, t_{m-n+2})$ to sample the mth word in the sentence. If there is not enough history available (e.g. when sampling the first word of a sentence using a trigram model) we marginalize over the unavailable history and condition the resulting marginal distribution.

The content model can also be used to generate sentences in an unconstrained way by sampling the mth (word, tag) pair instead of conditioning on an externally provided POS tag. The only subtlety here is deciding when to end a sentence; however, we can take a very naïve approach and simply continue to sample new words until a sentence ending punctuation mark is generated. This technique is exploited to generate sentences directly from the content model during the evaluation.

Structure Model The structure model used in this report is based on building empirical distributions of parse tree productions. We assume that the probability of each production occurring is independent of where in the parse tree it occurs. This assumption is known to be false, but we hypothesize that this model will still be sufficient to provide useful constraints to the content model.

Formally, the structure model builds a collection of conditional distributions, $P(T|H)$ where each production is of the form $H \rightarrow T$.

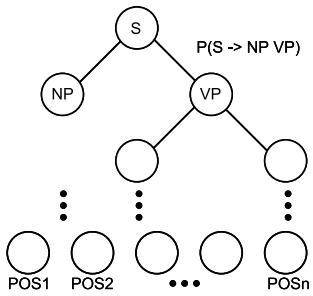

Figure 1: An illustration of the structure model.

The structure model is shown schematically in Figure 1. The bottom row of Figure 1 shows a sequence of POS tags which we call the sentence skeleton. This skeleton is the output of the structure model and is used as a constraint for the content model as discussed above.

If we do not stop the model at the POS tag level then we obtain the structure model we described above plus a unigram content model (i.e. it is constrained only to select a word from the correct POS category). This comes almost for free with our implementation because of the way the Stanford Parser produces parse trees. We use this approach to generate sentences from the structure model during the evaluation.

Combined Model We have described how each model is built from the input data and have discussed how to generate samples from each model. Algorithm 1 describes the process for producing a sentence from the combined model.

Algorithm 1: Generating from the combined model

```
Data: S – Structure Model
Data: C – Content Model
begin
    sentence ← [ ]
    skeleton ← S.generateSentenceSkeleton()
    for tag ∈ skeleton do
        next ← C.generateWord(t, tail(C))
        sentence.append((next, t))
    end
    return sentence
end
```

5.2.3 Realization

In our system the realization phase is a very simple post process, and is implemented as a series of regular expressions which are applied to the generated sentences. The realizer is responsible for ensuring that sentences are capitalized properly, that punctuation is positioned correctly with respect to the surrounding words

(e.g. `word ,  word` is transformed to `word,  word`), and a variety of other aesthetic transformations.

6 Evaluation

6.1 JGAAP

We evaluate our system using a stylometric tool known as JGAAP[2] (Juola, 2006). JGAAP implements a wide variety of standard stylometric tests for authorship attribution and provides a unified framework for running analyses. We first describe the general analysis workflow that JGAAP supports and then provide specific information on the various features and metrics used in our evaluation.

JGAAP is a tool designed for performing authorship attribution. This is an appropriate tool to use to evaluate our system since our goal is to mimic the style of a given author. We consider our mimicry to have been successful if JGAAP attributes our generated document to the author whose works were used to train the model which produced it.

To preform authorship attribution with JGAAP we must provide the system with a representative set of documents for each reference author, as well as a set of unlabeled documents. The result of running JGAAP is an assignment of each of the unlabeled documents to exactly one of the reference authors. The JGAAP workflow has three steps:

1. **Canonicalization** In this step the raw text of each document is processed into a standard form. JGAAP provides a variety of canonicalization filters, we chose to use the *normalize whitespace* and *unify case* filters throughout our analysis.

2. **Feature Extraction** In this step the canonicalized documents are turned into feature vectors, which JGAAP calls event sets. We evaluate our system using a variety of different features which are described in Section 6.2.

3. **Analysis** In this step the feature vectors for each unlabeled document are compared to each of the reference documents and the closest match is selected. We experimented with several different metrics for comparing feature sets and these metrics are described in Section 6.3.

6.2 Features

JGAAP supports a wide range of feature sets. We selected the following subset to use in our evaluation:

[2]JGAAP can be obtained at: `http://server8.mathcomp.duq.edu/jgaap/w/index.php/Main_Page`

words Each document is described by counts of individual word occurrences.

word bigrams Each document is described by counts of bigram occurrences.

word tetragrams Each document is described by counts of 4-gram occurrences.

hapax legomena Each document is described by the set of words which occur only once.

word lengths Each document is described by the distribution of word lengths.

common words Each document is described by the words which occur most frequently.

syllables per word Each document is described by a distribution over the number of syllables per word.

word stems Each document is described by the collection of word stems which it contains.

6.3 Analysis

In this section we describe the various metrics we used to compare document feature vectors. Some of these metrics are appropriate for comparing vectors of feature counts directly and others require distributions over these features. When distributions are required it should be assumed that we are referring to empirical distributions which are obtained by normalizing the corresponding feature count vectors. When referring to random variables we will use capital letters to denote distributions and lowercase letters to denote mass functions.

Kullback Leibler The KL divergence is used to quantify the similarity of two probability distributions. Given two distributions,

$$KL(P||Q) = \sum_x p(x) \log \left(\frac{p(x)}{q(x)} \right)$$

which can be interpreted as the expected difference in log likelihood of between $P(x)$ and $Q(x)$.

Manhattan The Manhattan distance between two vectors is the 1-norm of the difference between them. It is defined as

$$M(\mathbf{x}, \mathbf{y}) = \sum_i |x_i - y_i| \ .$$

KS The KS distance uses the test statistic from a two sample Kolmogrov-Smirnov test to compare two distributions. Given two discrete distributions this statistic is

$$K(P, Q) = \max_x |P(x) - Q(x)| \ .$$

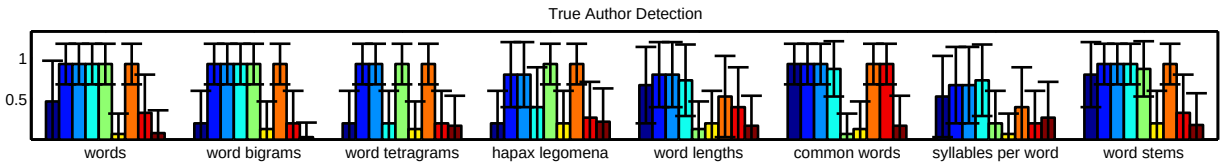

Figure 2: Authorship attribution performance when using no generated data. Each cluster of bars shows a different feature set and within each cluster individual bars represent different analysis methods. Error bars show one standard deviation above and below the mean. The bars for each analysis method are arranged from left to right in the same order they appear in Section 6.3.

K will be identically zero if the two distributions are the same.

Histogram This histogram distance between two vectors is the squared L_2 norm of the difference between them,

$$H(\mathbf{x}, \mathbf{y}) = \sum_i (x_i - y_i)^2 \quad .$$

Intersection The intersection distance treats the two feature count vectors as sets and computes

$$D = 1 - \frac{|A \cap B|}{|A \cup B|} \quad .$$

Naïve Bayes The naïve bays metric uses a naïve bayes classifier to assign authorship.

Cosine The cosine distance between two vectors is the their inner product normalized by their lengths,

$$C(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{||\mathbf{x}||||\mathbf{y}||} \quad .$$

Canberra The canberra distance is similar in flavor to the Manhattan distance but involves normalizing each term in the summation

$$D(\mathbf{x}, \mathbf{y}) = \sum_i \frac{|x_i - y_i|}{|x_i| + |y_i|} \quad .$$

Random Random distances are assigned between documents. This is useful only as a baseline metric.

7 Results

We preform the evaluation described in Section 6 on four different data sets. The first is a reference data set where we removed labels from some of the training documents and ran JGAAP to assign authorship to each of them. This evaluation provides an important baseline for assessing the usefulness of the different evaluation methods on our data set. The other data sets we used are data generated from the structure model, data generated from the content model, and data generated from the combined model. In all cases we used the entire training set as reference documents for JGAAP.

For each model type, we trained one model for each author using the available training data. We then use each of these models to generate 5000 sentences and use these generated documents as the unlabelled data in the input to JGAAP. The goal is to have each unlabelled document attributed to the author who was used to train the model which generated it. We report the average success on this task for each model averaged over 10 runs. In all cases when training the content model we used 3-grams.

The results from of baseline evaluation are shown in Figure 2. From this figure we can see that all of the feature sets preform similarly, with the notable exceptions of *word lengths* and *syllables per word* which appear to under preform the others. Much more striking in Figure 2 is that several of the analysis methods preform very poorly. Notably, naïve bayes appears among the poorest performers across all feature types, performing no better

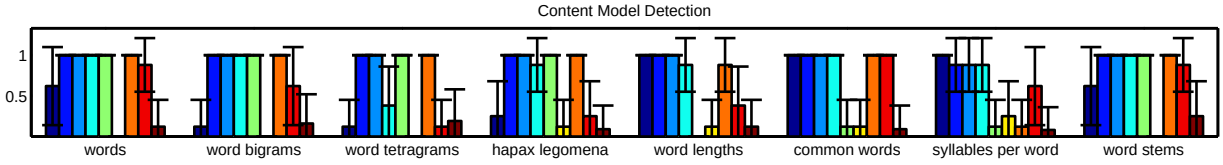

Figure 3: Authorship attribution performance when using only the content model. Each cluster of bars shows a different feature set and within each cluster individual bars represent different analysis methods. Error bars show one standard deviation above and below the mean. The bars for each analysis method are arranged from left to right in the same order they appear in Section 6.3.

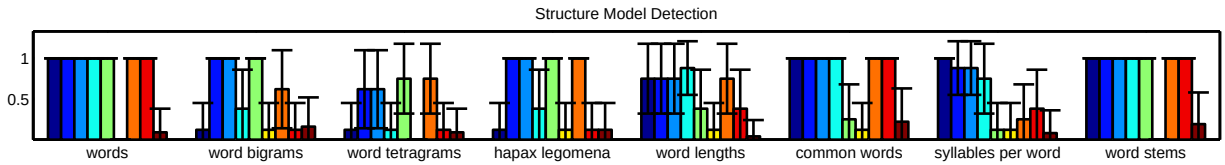

Figure 4: Authorship attribution performance when using only the structure model. Each cluster of bars shows a different feature set and within each cluster individual bars represent different analysis methods. Error bars show one standard deviation above and below the mean. The bars for each analysis method are arranged from left to right in the same order they appear in Section 6.3.

in general than random assignment. The best performing measures are *Manhattan*, *KS* and *histogram* with *Kullback Leibler* only marginally worse.

Figure 3 shows the results of our evaluation on data generated from the content model. In this case we see a pattern of high and low performing analysis methods similar to the baseline, which is encouraging because it indicates that much of the inter-analysis method variation in performance can be attributed to variation in performance on the source data and is thus not an artifact of our model. This pattern is quite consistent, with the notable exception of the *Kullback Leibler* measure when the *syllables per word* feature is used. Although this pair preforms relatively poorly in the baseline, documents generated by the content model are attributed to their progenitor with 100% accuracy in this case.

A feature of Figure 3 which differs markedly from the baseline is that there are several feature-analysis method pairs for which the attribution is successful 100% of the time. This phenomenon is present in the evaluation of all of our models; however, it never happens in the baseline evaluation. This phenomenon can likely be attributed to the relative richness in vocabulary available to an author compared to what is available to our system. All the models we use can only produce words (and word patterns) which they have seen before, while a human author is under no such restriction. This dearth of variation means that patterns in the training data reoccur more frequently in the generated data than between different works by the same author.

Figure 4 shows the results of our evaluation on data generated from the structure model. If we compare this performance to the performance of the content model alone we see a drop in performance on the *word bigrams* and *word tetragrams* features. This is not surprising, since the structure model on its own models only unigram distributions over parts of speech while the content model is able to select each word using information from the surrounding context. Performance using other feature sets is very similar to the performance shown by the content model.

Finally, Figure 5 shows the results of our evaluation on data generated by the combined model. Although the performance looks very similar to the previous two models we can notice some interesting points. If we look at the performance with the *word bigrams* and *word tetragrams* features we see that the performance of the combined model more closely matches the baseline evaluation than the structure model alone. This shows that adding content information to the structure model is able to better capture sequence information than the structure model alone. This is not surprising when we consider how the models work, but it is nice to see this phenomenon reflected in our results.

Unfortunately when we look from the other direction, that is we try to assess how the content model has benefited from the addition of structural information, the evidence for improvement is less apparent. In fact we find no

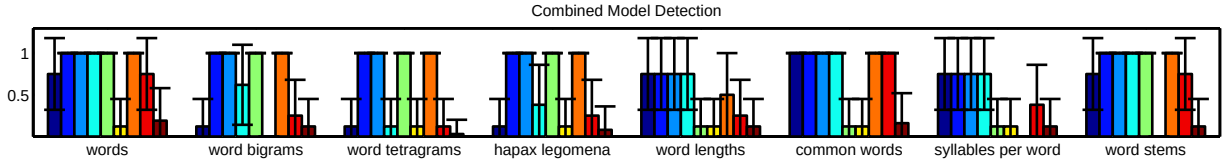

Figure 5: Authorship attribution performance when using the combined model. Each cluster of bars shows a different feature set and within each cluster individual bars represent different analysis methods. Error bars show one standard deviation above and below the mean. The bars for each analysis method are arranged from left to right in the same order they appear in Section 6.3.

evidence that the addition of the structure model provides any benefit beyond using the content model in isolation.

8 Lessons Learned

Two major lessons learned during the implementation of this report are:

Representation Matters The implementation that was built for this report stores all of its data in python dictionaries. While this is handy for fast development it makes the implementation difficult to extend because, e.g., the representation used in each of the content and structure models is very tailored for how data is used in those particular models in isolation. Building the combined model, which should have been a simple task of sticking the two existing models together, actually ended up requiring that we reimplement the functionality of the structure and content models to allow them to work together.

This difficulty with representation means that the code contains two implementations of the functionality in the structure and content models, one for using the models in isolation and another for the combined model. In retrospect, a more robust data storage management mechanism would have been very useful.

Parse Trees are Complicated When designing the structure model we made the assumption that the probability of a production is independent of its position in the parse tree. As we mentioned, this is known to be false but our expectation was that there would still be sufficient information in the structure model to benefit over a pure content model. Our analysis found that this is not the case for the low level features we examined. We also looked at the distribution of sentence lengths for some selected models[3] and found that the distribution of sentence lengths in our generated data does not match well with the sentence length distributions of the author used to train the model.

Our simple structure model appears to not only be incorrect (which was known from the beginning) but it is also insufficient to capture the type of information we intended it to encode.

9 Directions for Future Work

The evaluation in this report focused on fine grain features like the occurrence of individual words (unigrams, bigrams, etc). This analysis is not ideal since it is not clear that it properly tests the utility of the structure model. A more thorough evaluation would consider higher level features as well, such as the distribution of sentence lengths. The reason for not including this type of analysis in this report is purely pragmatic: JGAAP does not implement these type of analyses and we were not able to find any sufficiently powerful alternative tool with which to preform the evaluations. Future work on this type of problem should include analysis which integrates high level features. This is especially important since, as we mentioned in the previous section, our ad-hoc examination of sentence length distributions reveals that our models do not mimic this feature well. This weakness is not apparent in the analysis presented in Section 6.3.

An obvious step beyond what was covered in this report is the integration of semantics into the generated text. There are two ways to approach this problem, one is to try to learn the semantics of the training texts and create a model to mimic them, or another is to devise a method to specify the semantics through an external process. This second method has more potential for application, since it could, in principle, be attached to the output of any semantic engine in order to realize text in a given style; however, the problem of semantics is extremely difficult in general and nothing in this report suggests an obvious starting point for either approach.

[3]There results were not included in the analysis since the examination we preformed was very ad-hoc.

References

S Argamon and M Koppel. 1998. Routing documents according to style. *In Proceedings of First International*, 60(4):455–60, Oct.

Michael Brennan and Rachel Greenstadt. 2009. Practical attacks against authorship recognition techniques. *Innovative Applications of Artificial Intelligence*.

Aidan Finn and Nicholas Kushmerick. 2006. Learning to classify documents according to genre. *Journal of the American Society for Information Science and Technology*, 57(11):1506–1518, Sep.

P. Juola. 2006. Authorship attribution. *Foundations and Trends in information Retrieval*, 1(3):233–334.

Gary Kacmarcik and Michael Gamon. 2006. Obfuscating document stylometry to preserve author anonymity. *Proceedings of the COLING/ACL on Main conference poster sessions* -, pages 444–451.

D. Klein and C.D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 423–430. Association for Computational Linguistics.

Harold Somers and Fiona Tweedie. 2010. Authorhsip attribution and pastiche. *Computers and the Humanities*, 37(4):407–429.

K. Toutanova and C.D. Manning. 2000. Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In *Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora*, pages 63–70. Association for Computational Linguistics.

Özlem Uzuner and Boris Katz. 2005. A comparative study of language models for book and author recognition. *Natural Language Processing-IJCNLP 2005*, pages 969–980.

A Appendix: Source code and pointer to corpus

The best place to find the source code and corpus used in this report is to access the github repository directly at `https://github.com/mdenil/cpsc503_project`.