My beautiful dissertation

Tobin South

October 28, 2020

Thesis submitted for the degree of
Masters of Philosophy
in
Applied Mathematics
at The University of Adelaide
Faculty of Engineering, Computer and Mathematical Sciences
School of Mathematical Sciences

Chapter 1

Background

[[TODO: Introduction to background]] Information theory, what is it, why do we use it Networks, why Natural language processing

1.0.1 Information Theory

Entropy

Entropy is a measure of the uncertainty of a random variable. In the context of information theory, this is defined by Equation 1.1, often refereed to as Shannon entropy, named after Claude Shannon for his work in 1948 studying the quantiles of information in transmitted messages [[TODO: cite Claude shannon 1948]]. The definitions hereafter are sourced from Elements of Information Theory by Thomas and Cover [[CITE: elements of information theory]]

Definition 1.0.1 (Shannon Entropy)**.** Let X be a discrete random variable with alphabet $\mathcal{X}$ and probability mass function $p(x) = P(X = x), x \in \mathcal{X}$. The entropy $H(X)$ of the discrete random variable X, measured in bits, is

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log_2 p(x) \tag{1.1}$$

The entropy of the random variable is measured in bits. A bit can have two states, typically 0 or 1. The entropy of a random variable is the number of bits on average that is required to describe the random variable in question. To measure the entropy in bits, we use a logarithm of base 2, and all logarithms throughout this work are assumed to be in based 2, unless otherwise specified.

To give a typical example of entropy, if a fair coin is tossed there are two equally probable outcomes, giving an entropy of 1 bit. Further, we use the convention of $0 \log 0 = 0$, which sensibly means that adding a state with 0 probably to the random variable does not change it's entropy.

Remark (Suprise). The entropy of the random variable X can also be described in terms of the expected surprise, where the surprise of a state is $\log \frac{1}{p(x)}$.

$$H(X) = \mathbb{E}\left[\frac{1}{p(x)}\right] \tag{1.2}$$

[[TODO: add some filler]]

Lemma 1.0.1. The entropy of a random variable is strictly non-negative, $H(X) \geq 0$.

Proof. $0 \leq p(x) \leq 1$ which implies that $log\frac{1}{p(x)} \geq 0$, hence the sum of products of strictly non-negative terms will always be non-negative. ■

[[TODO: add some filler]]

Joint Entropy and Conditional Entropy

Above we worked with a single random variable. To extend this we introduce a second discrete random variable Y. Using this, we extend the one dimensional entropy to joint entropy.

Definition 1.0.2 (Joint Entropy)**.** The joint entropy $H(X, Y)$ of a pair of discrete random variables (X, Y) with a joint distribution $p(x, y)$ and state spaces $(\mathcal{X}, \mathcal{Y})$

$$H(X, Y) = -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x, y) \log p(x, y) \tag{1.3}$$

From our definition of entropy and the law of total probability we can create a notion of conditional entropy.

Definition 1.0.3 (Conditional Entropy)**.** The conditional entropy $H(X|Y)$ of two discrete random variables X and Y is defined as,

$$\begin{aligned} H(X|Y) &= \sum_{y\in\mathcal{Y}} p(y) H(X|Y = y) && (1.4)\\ &= -\sum_{y\in\mathcal{Y}} p(y) \sum_{y\in\mathcal{Y}} p(x|y) \log p(x|y) && (1.5)\\ &= -\sum_{y\in\mathcal{Y}}\sum_{y\in\mathcal{Y}} p(x, y) \log p(x|y) && (1.6)\\ &= -E \log p(X|Y) && (1.7) \end{aligned}$$

Subtly different from the *conditional entropy* is the *cross entropy*. Whereas the *conditional entropy* is the amount of information needed to describe X given the knowledge of Y, the *cross entropy* is the amount of information needed to describe X given a optimal coding scheme built from Y.

Definition 1.0.4 (Cross Entropy)**.** The cross entropy $H_\times(q|p)$ between two probability distributions, defined over the same state space, p nd q is defined as,

$$H(q||p) = -\sum_x p(x) \log q(x) \tag{1.8}$$

define $H(X)$ vs $\hat{h}$ and $H(p||q)$ and $\hat{h}_\times$

Although cross entropy has the common notation $H(X, Y)$, in this thesis we will use an alternative $H(X||Y)$, reminiscent if the Kullback–Leibler divergence in Equation 1.9 below, so as to not confuse the cross entropy with the above join entropy Equation 1.3 of the same notation.

Remark. Importantly, note that $H(X|Y) \neq H(Y|X)$ and $H(q||p) \neq H(q||p)$, both properties we will exploit later.

Distances

We can extend these ideas to explore a notion of distance between probability distributions. Kullback–Leibler divergence is a measure of the inefficiency if one were to assume that a distribution is p when the true distribution is q.

Definition 1.0.5 (Kullback–Leibler divergence)**.** The Kullback–Leibler divergence (also called relative entropy), $D(p||q)$, between two probability distributions $p(x)$ and $q(x)$ is,

$$D(p||q) = \sum_{x \in \mathcal{X}} p(x) \log \frac{p(x)}{q(x)} \tag{1.9}$$

$$= E_p \log \frac{p(X)}{q(X)} \tag{1.10}$$

Again, we use the convention that $0 \log \frac{0}{0} = 0$ and $p \log \frac{p}{0} = \infty$.

Conveniently, we can also express the Kullback–Leibler divergence in terms of the cross entropy.

Lemma 1.0.2.

$$D(p||q) = H_p(q) - H(p) \tag{1.11}$$

Proof.

$$\begin{aligned} D(p\|q) &= \sum_{x\in\mathcal{X}} p(x)\log\frac{p(x)}{q(x)} && (1.12)\\ &= \sum_{x\in\mathcal{X}} p(x)\log p(x) - \sum_{x\in\mathcal{X}} p(x)\log q(x) && (1.13)\\ &= -H(p) + H(q||p) && (1.14)\\ & && (1.15) \end{aligned}$$

■

The Kullback–Leibler divergence has two difficulties; It's not symmetrical and it can return infinite values. Jensen–Shannon divergence builds from the Kullback–Leibler divergence to solve these problems to a symmetric, finite comparison between probability distributions.

Definition 1.0.6 (Jensen–Shannon divergence)**.** The Jensen–Shannon divergence between two probability distributions $p(x)$ and $q(x)$ is,

$$\mathrm{JSD}(p\|q) = \frac{1}{2}D(p\|m) + \frac{1}{2}D(q\|m) \tag{1.16}$$

using a mixture of the distributions, $m = \frac{1}{2}(p+q)$.

Remark. The square root of the Jensen–Shannon divergence provides a metric, often referred to as Jensen–Shannon distance.

[[TS: define metric?]]
[[TS: add mutual information]]
[[TS: add variation of information]]
[[TS: add diagram]]

Entropy Rates

Entropy rate of a stochastic process describes the amount of information required to describe the future state of a process, conditioned on the information in the history of the process.

"The entropy rate is almost surely an asymptotic lower bound on the per-symbol description length when the process is losslessly encoded" [[CITE: Some asymptotic properties of entropy of a stationary ergodic data source with applications to data compression,]]

Definition 1.0.7 (Entropy Rate)**.** Let $\mathcal{X} = \{X_i\}$ be a stochastic ergodic process with a finite alphabet, where X_i^j denotes a subsequence of the process $(X_i, X_{i+1}, \ldots, X_j)$. The entropy rate can be defined as,

$$H(\mathcal{X}) = \lim_{n\to\infty} H\left(X_n | X_{n-1}, X_{n-2}, \ldots, X_1\right) \tag{1.17}$$

Which, on the assumption of stationary, can be expressed as,

$$H(\mathcal{X}) = \lim_{n\to\infty} \frac{1}{n} H\left(X_1, X_2, \ldots, X_n\right) \tag{1.18}$$

While this notion of entropy rate provides a valuable theoretical tool, calculating it for real examples can prove difficult, and often impossible given data. In chapter 3 we will explore a method of estimating a similar quantity.

[[TODO: entropy rate of an i.i.d process proof]]

Predictability

Predictability is the probability π that an theoretical predictive algorithm could predict the next state of a process correctly, this often be difficult to obtain. However, an upper bound, $\pi \leq \pi^{max}(S, N)$, is possible through the use of Fano's inequality [6]. For a process with $\pi^{max} = 0.3$, at best we could hope to predict this process correctly 30% of the time, no matter how good our predicative algorithm [21].

Definition 1.0.8 (Maximal Predictability)**.** For a process X with entropy $H(X)$, Fano's inequality in the context of our maximal predictability gives,

$$H(X) = H(\pi^{max}) + (1 - \pi^{max}) \log(|\mathcal{X}| - 1) \tag{1.19}$$

The entropy of the maximal predictability $H(\pi^{max})$ is substituted with the binary entropy function [21],

$$H(\pi^{max}) = -\pi^{max} \log(\pi^{max}) - (1 - \pi^{max}) \log(1 - \pi^{max}). \tag{1.20}$$

Which finally gives us a form that can be solved numerically for the fundamental limit of the process' predictability, π^{max},

$$-H(X) = \pi^{max} \log(\pi^{max}) + (1 - \pi^{max}) \log(1 - \pi^{max}) - (1 - \pi^{max}) \log(|\mathcal{X}| - 1). \tag{1.21}$$

Throughout this thesis, maximal predictabilities will found by solving Equation 1.21 using the Powell's conjugate direction method, implemented in python using SciPy [22], with a starting estimate for the root at $\pi^{max} = 0.5$.

We extend this notion of maximal predictability of a process, to create a cross predictability using cross entropy Definition 1.0.4. [[TODO: more]]

1.0.2 Networks

[[TODO: go through Newman networks and state a bunch of definitions]]

1.0.3 Natural Language Processing

[[TS: introduce in here the notion that we're interested in text and NLP is a fundamental building block of understanding it]]

Natural language processing (NLP) is the area of study in which 'natural' human language is examined via machine. Natural language refers to either spoken or written language, designed to be understandable to a human listener or reader. This language is not explicitly designed to be machine understandable, and machine comprehension of this language is a challenging problem [[CITE: cite: the challenges of NLP]].

NLP is a broad term covering many models and techniques to computationally extracting meaningful information from text, ranging from the simple extraction of individual words, to the extraction of deeper semantic meaning.

Early work in NLP focused around simple grammatical rules and small vocabularies, such as the work of Georgetown-IBM [[CITE: cite: John Hutchins. From first conception to first demonstration: the nascent years of machine translation, 1947–1954. a chronology. Machine Translation, 12(3):195–252, 1997.]] to translate 60 sentences from Russian to English in 1954. With the rapid increase in computational power and digital text corpuses, modern NLP has focused or deeper challenges of extracting meaning from text with tools such as Word2Vec [[CITE: cite: word to vec]] or deep learning methods such as Google's BERT [[CITE: cite: BERT]].

These methods face a daunting challenge, language is not only complex and often duplicitous, but contextual and ever-changing. [[TODO: end better]]

Tokenization

Tokenization is one of the most fundamental building blocks of NLP and can often prove deceptively hard. In simple terms, tokenizing a piece of text is the process of breaking a long sequence of characters into smaller chucks of characters called tokens. In a simple context, this often means breaking a sentence into words, e.g. `'This thesis is great'` becomes `['This', 'thesis', 'is', 'great']`.

The task becomes more complex when we introduce contractions. In the case of the text, `"that's"`, should we introduce a new token to represent the compound, or break it into two new tokens, `['that', 'is']`? While this may seem trivial, a process of matching segments of text is highly dependent on choices such as these, and the more tokens that are introduced, the larger the vocabulary size.

The vocabulary size is simply the number of unique tokens in any corpus, but can have

compund words and unknown characters

Text Generation

drawn from any abritrayr distribution

Zipf law (note that s is between 1 and 2.)

drawing from real data

Chapter 2

Data Collection and Cleaning

In order to analyse information flow in news, we first need a dataset that contains it. This dataset needs to be comprehensive, containing most popular news sources; textual, as methods developed in this work are based on text data; and relevant, as the analysis of news should happen where people actually consume it.

In the 1950s, this source would be household television. The widespread popularity of which allowed TV broadcasting to become the primary tool for influencing public opinion in developed nations [5]. The rise of mobile internet and social media sites in the last two decades has reshaped this paradigm. No longer a population that watch news at fixed times, or read regularly scheduled newspapers; the conveniences of the modern developed world allow individuals to consume news any time, anywhere. As of 2019, 55% of US adults get their news from social media either 'often' or 'sometimes' and 88% state that 'social media companies have at least some control over the mix of news people see' [17].

Given the importance of a free press and the role of social media sites in the delivery of news, this work aims to study the news on social media. To begin this task, we first define some common terms for clarity.

Definition 2.0.1 (Social media)**.** The platforms used to consume and share information by individuals in the public. E.g. Twitter, Facebook, Reddit.

Definition 2.0.2 (News-media)**.** The organisations that are producing information about a broad range of current events and sharing that information with the public.

Definition 2.0.3 (News)**.** The *content* produced by news-media organisations.

Definition 2.0.4 (Consumers)**.** Individuals in the public that willingly seek out and consume news from news-media organisations.

2.1 Data

Using the media analysis source AllSides[1], a collection of news-media organisations was found. The purpose of AllSides is to provide an open analysis of political leanings of news sources [8], and to aggregate news allowing consumers to view articles from different sides of the political spectrum. Each news source is labelled into one of 5 categories, Left, Lean Left, Center, Lean Right, or Right. For each news source the ratings are determined internally using 'blind surveys of people across the political spectrum, multi-partisan analysis, editorial reviews, third party data, and tens of thousands of user feedback ratings' [8]. News sources are only assigned to a single category, but do have an attached confidence rating that is provided from users selecting if they agree or disagree with the rating. An example of the ratings can be seen in Figure 2.1.

From the website, a list of possible news sources was collected on February 1st, 2019. In this collection was organisation names, political bias', the number of user feedback ratings of the political bias, and, if available, the twitter handles associated with those sources. These collected news sources were broad, containing not just news-media organisations but authors, pundits and think tanks.

To select an appropriate set of news-media organisations, an examination and filtering process was undertaken. A source was only considered if it was a organisation (not an individual), that produced news content of a diverse range of topics. Many news sources were connected to think tanks or opinion groups, and only created news of a single topic or campaign. Further, if an news-media organisation has no twitter account or had less than 10,000 followers (a low bar in the social media world), then it was removed from the pool. This mainly removed inactive organisations and news organisation from very small rural towns. Finally, a single source was removed as it was not in English, and a single source was removed as it was the smaller sister site that had all content as a subset of it's larger site. The result of this filtering process is 170 news-media organisations with associated Twitter accounts and categorised political bias'. A list of all news-media organisations under analysis can be seen in **??** and all removed sources and the removal justification can be seen in **??**.

Collection

Using the Twitter user handles associated with each of the news-media organisations, the history of all tweets for each account was collected using the

[1]www.allsides.com

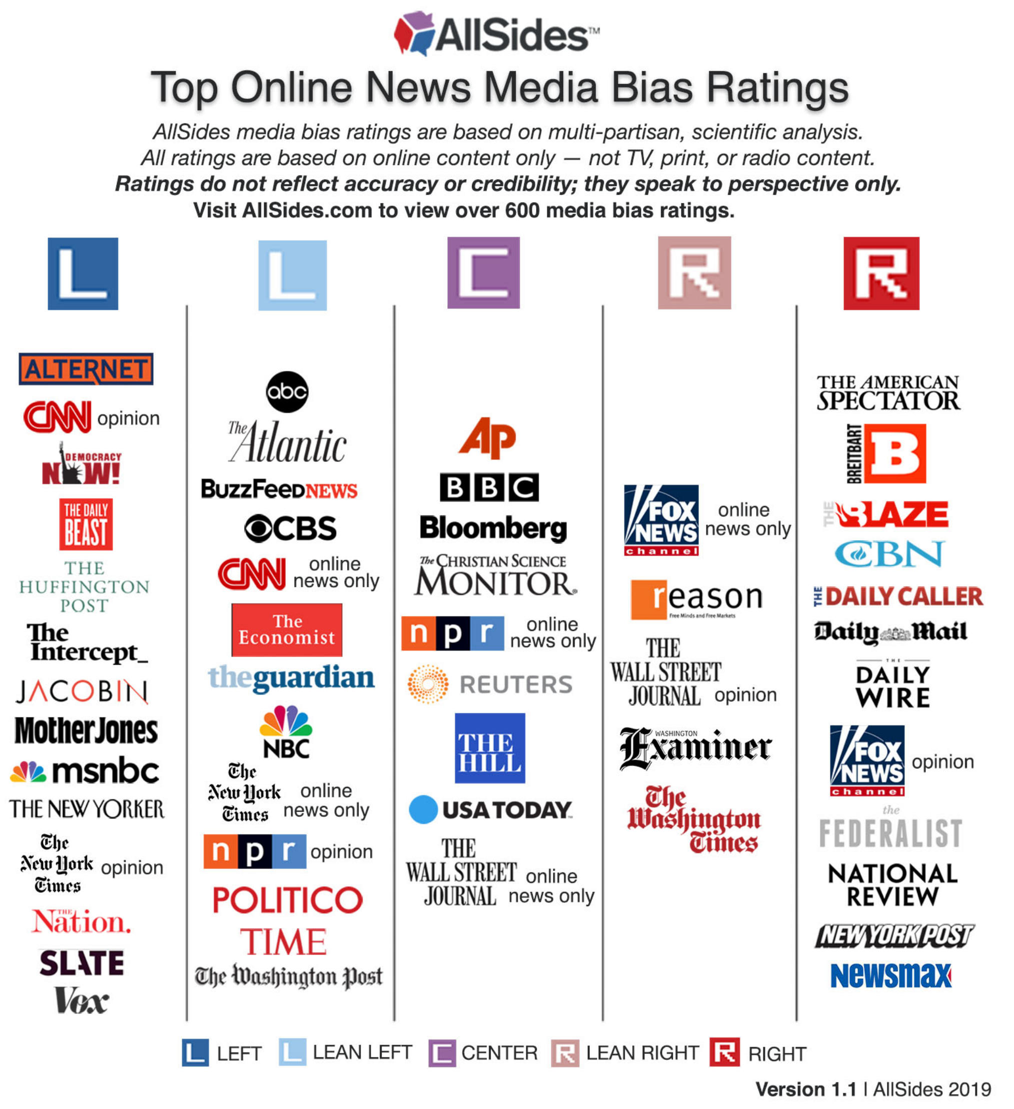

Figure 2.1: An example collection of News-Media sites that have been classified into biases; sourced from Allsides website [8]

Twitter application programming interface (API)[2] and web-scraping tools[3]. Of interest in this work are the tweets each news organisation tweeted between January 1st, 2019 to January 1st, 2020.

Each major news-media organisation will tweet pieces of news multiple times throughout the day. The manner in which each organisation does this

[2]https://developer.twitter.com/en/docs
[3]https://github.com/twintproject/twint

can differ and no standard format is used. The tweets often come in the form of single line description of articles, alongside a link to an article on the news-media organisation's website. The primary purpose of using social media sites to post these stories is to drive traffic to the organisation's website, wherein they can earn revenue from ad impressions. As such, the format of such news tweets is to extract core concepts from articles and frame them in their most essential and appealing way; in essence, they are trying to create so-called 'click-bait' [**?**]. This format is desirable for our work as we want to explore how the language we use in news to appeal to consumers differs between organisations. This simplified format presents a reduced essence of this notion.

Twitter also serves another purpose for news-media organisations; as a tool for breaking news. The modern 24 hour news cycle has had many effects on journalism, but chief among them is the need to produce breaking news at a lightning fast pace. The use of social media as a near instant tool for global public communication means that often no time can be wasted in publishing knowledge of a story, often while it is unfolding. Indeed, research has explored the role of Twitter for breaking news in the cases of the 2011 UK summer riots [23], through activity providing real time updates over the four days, and in the case of the death of Osama Bin Laden in 2011 [10], where the news was leaked and spread virally through social media before any news-media organisations could fully verify and publish stories on the claim.

Account Removal

Using this collection method a total of 3,221,769 tweets were collected from the 170 news-media organisation official Twitter accounts in the 2019 calendar year. This represents an average number of tweets per day of above 50 tweets for each news organisation. In total, this appears a large useful corpus of text data for analysis. However, the activity level and consistency of output variety greatly between organisations. As can be seen in Figure 2.2, some news organisations produce very little content on average. This can be explained through two mechanisms.

Firstly, some organisations are not very active on social media. In particular, smaller organisations, which are typically less well resourced, place a lower priority on social media posting. This lowered tweet volume, presents a challenge for this work. In particular the use of the non-parametric entropy estimator in chapter 3 require a substantial amount of text to reach meaningful results. As such, organisations that produced less that 1000 tweets in 2019 were removed from further consideration. This removed a total of 11

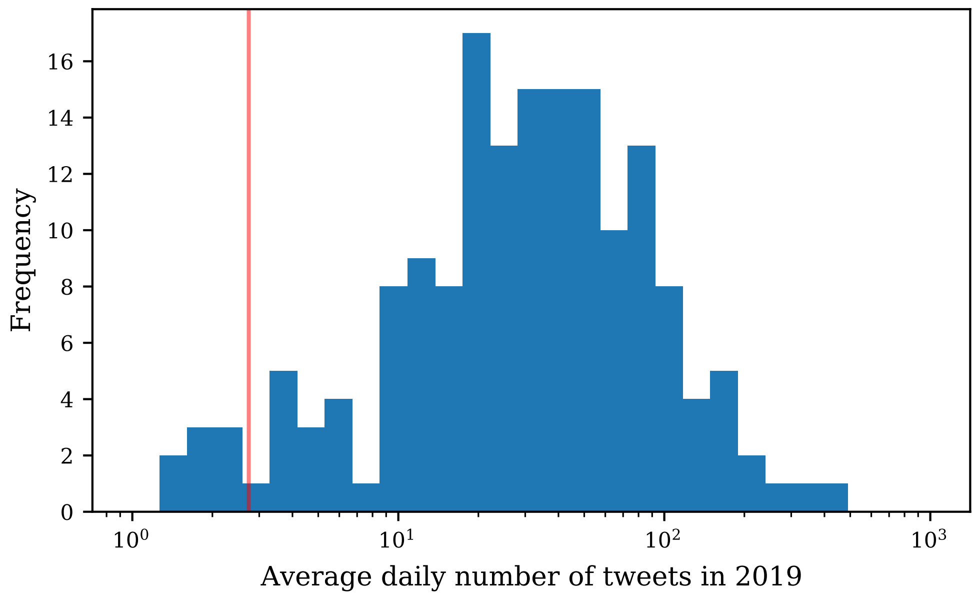

Figure 2.2: The average number of tweets produced each day during the 2019 calendar year for all 171 news-media organisations. The red line is the chosen threshold of 1000 tweets in the year, an average of 2.74 tweets per day. [[TODO: Change y-axis label to 'Number of news-media organisations'.]]

news-media organisations, listed in Table 2.1.

News-media Organisation	Bias	Number of tweets in 2019
RealClearPolitics	Center	532
IJR	Lean Right	777
WND News	Right	709
PRI	Center	346
EurekAlert!	Center	610
FAIR	Center	697
Crowdpac	Center	521
Inside Philanthropy	Center	781
Diplomatic Courier	Center	750
Peacock Panache	Left	198
Independent Voter	Center	303

Table 2.1: Table of news-media organisations that were removed from data due to a low number of tweets in the 2019 calendar year.

Secondly, an issue was identified with long periods of inactivity of a few organisations. Five news-media organisations, for reasons unknown, had large periods of time in which they did not post any tweets. These periods of time spanning a few months present key issues to our investigation. Moving forward we will consider time an important aspect of news, especially in the context of breaking news, and as such these organisations are not only at a disadvantage in this space, but present an anomaly in our data that hinders our ability to extract meaningful results from them. These five organisations, listed in Table 2.2 were removed from further consideration and analysis.

News-media Organisation	Bias
American Thinker	Right
Pacific Standard	Lean Left
Philly.com	Lean Left
Splinter	Left
ThinkProgress	Left

Table 2.2: Table of news-media organisations that were removed from data due to long periods of inactivity.

To further confirm the validity of the sources in terms of their activity level over time, we examine the isolated daily activity of each news-media organisation. An activity curve for the New York Times can be seen in Figure 2.3. A clear weekly trend, wherein tweet activity is decreased, but not zero, during the weekends can bee seen in the New York Time activity, but is emblematic of a general trend seen in most news-media organisation. Further, many news-media organisations have distinct spikes that occur a key points during the year. These spikes indicate an extreme news day, wherein an organisation is covering a rapidly evolving breaking news story, or responding to major changes in discourse through the day. These are interesting and important features in the data, an worth keeping in mind during further analysis. The full collection of figures containing the daily activity levels of all included organisations can be seen in **??**.

With this activity-level cleaned data, our news-media organisations have a slightly higher average number of tweets per day of 52.97. With the total number of remaining tweets at 2,977,980.

Account Analysis

As a result of this filtering we are left with 154 news-media organisations with complete data for the 2019 calendar year. In total there are 73 organi-

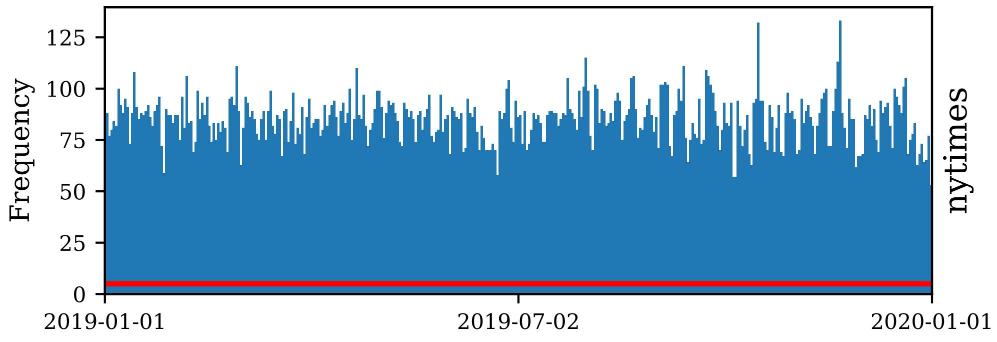

Figure 2.3: Twitter activity over 2019 for 'The New York Times'. Twitter handle is 'nytimes' with 44800317 followers and 31029 total tweets in 2019. A reference value of 5 tweets per day is shown in red. This is only one news-media organisation and all other activity figures for other organisations are available in ??, [[TODO: Change y-axis to number of tweets per day]]

sations in the left half of the bias spectrum, 44 in the centre and 37 on the right; expanded on in Table 2.3. This distribution, although shifted towards the left, still provides ample sources for the effect of bias to be explored further in this work, with keen attention to the potential impact of the skewed distribution.

Bias	Number of Organisations
Left	31
Lean Left	42
Center	44
Lean Right	16
Right	21

Table 2.3: The number of news-media organisations in each political bias classification in our data.

From the news-media Twitter accounts we can also access metadata about the organisation. Two useful such pieces of metadata are the geographic location, and the number of followers on twitter.

On the Twitter account of each news-media organisation they can elect to provide a text-based 'location'. In some cases this option can be used for other purposes, such for self promotion (e.g. the *New York Daily* states it's location as '`New York City / fb.com/nydailynews`') and many organisations have elected to leave the field blank. Of the organisation with text, these can be difficult to disambiguate and compare. Due to this complexity of possible locations, these classifications were done manually and are summarised in Table 2.4. In situations were multiple cities or locations are defined, the largest possible inclusion was taken. For example '`New York and the World`' would become 'Worldwide' in our classification, as would '`NYC, London, Paris, Hong Kong`'. There is a notable U.S focus to the data, as is expected using a U.S. based bias rating tool.

Location	Counts
New York	34
Washington, D.C.	20
Cafilforna	11
Other US City	44
General US	8
Worldwide	6
United Kingdom	5
Qatar	1
Pakistan	1
Korea	1
Unspecified or Unclear	43

Table 2.4: The aggregated self-defined locations of news-media organisations according to their Twitter account metadata.

The number of followers a news-media organisation has on Twitter is an important metric, as it underpins the default mechanism through which people consume the news content produced. Although other algorithm news-feed behaviours provide an important effect, the follower count plays a role in such a feed and is an effective metric of 'success' in the eyes of the news-media organisations.

The most followed organisation in the data is *The New York Times* with 44,800,317 accounts following it at the time of collection on the 13th January 2020. The least followed account included in the data is *CalMatters* with 15,069 followers. Interestingly, the follower counts are slightly higher for left biased organisations than for the right, in addition to being more numerous. This is shown via the followers distributions for each bias in Figure 2.4, which is indicative of the larger trend in social media of slightly left leaning demographics [?].

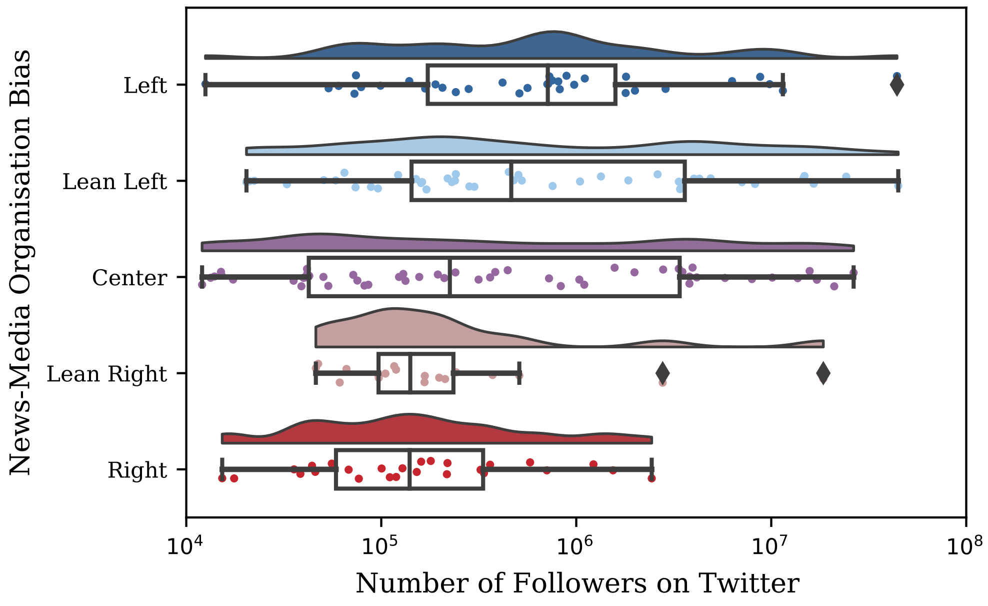

Figure 2.4: Number of followers on Twitter of news-media organisations included in the data. Grouping is according to Allsides bias.

2.1.1 Tokenization

With the 2,977,980 tweets from the remaining accounts, we process each tweet into an array of words. As discussed in section 1.0.3, the process of tokenization is applied to each string independently.

To perform the tokenization, the TweetTokenizer[4] from the Natural Language Toolkit (nltk) in Python is used. This built-in tokenizer is specially designed for tweet-style strings and bundles several useful features.

For each tweet, four steps are applied:

1. Twitter account handles, which appear in the form '@account_name', are removed.

2. Any sequence of a character that repeats more than three times is reduced to a maximum of 3. This has the effect of standardising text of similar meanings a common form, such as 'waaaaayyyy' and 'waaayyyyyy' to 'waaayyy'.

3. All URLs included in the text are removed.

[4] www.nltk.org/api/nltk.tokenize

4. The string is converted to lower-case and split at each white-space, giving an array of individual words.

These steps are applied uniformly across the corpus, and serve a core purpose. The underling task of this work is to identify the flow of linguistic structure within news. In order to achieve this, we need to standardise the text to the most common possible format, such that text of similar phrasing and meaning will be matched between sources. These tokenization and text cleaning steps allow for these similarities to be expressed.

Vocabulary Sizes

With the tweets of each Twitter account tokenized, we can begin to explore the vocabulary sizes. The vocabulary size is the number of unique tokens that exist in the collection of all tweets from a user. Figure 2.5 shows the strong relationship between the amount of tweets produced and the number of unique words, with diminishing increases to vocabulary as new tweets are added.

The ratio of vocabulary size to the number of tweets provides a first glimpse into the level of complexity in the language. Accounts that have a more specific domain, such as ‘thehill’, will increase their vocabulary at a slower rate as new tweets are added due to the consistently repeating domain specific words. In contrast, an account such as the ‘guardian’ produces a diverse array of content, and hence has an inflated vocabulary size in contrast to it’s output.

We see the extreme variance in tweet activity reflected in the distribution of vocabulary size. The vocabulary sizes span from 2976 unique tokens (sciencedaily) to 40824 (guardian).

This presents a key challenge for the task at hand. This work needs to find information flows between news-media organisations which can have content corpus’ that differ in size by two orders of magnitude, and vocabulary sizes that can differ by up to one. This can exacerbate the problems already presented by natural language, and informs the need to normalise flows in later sections.

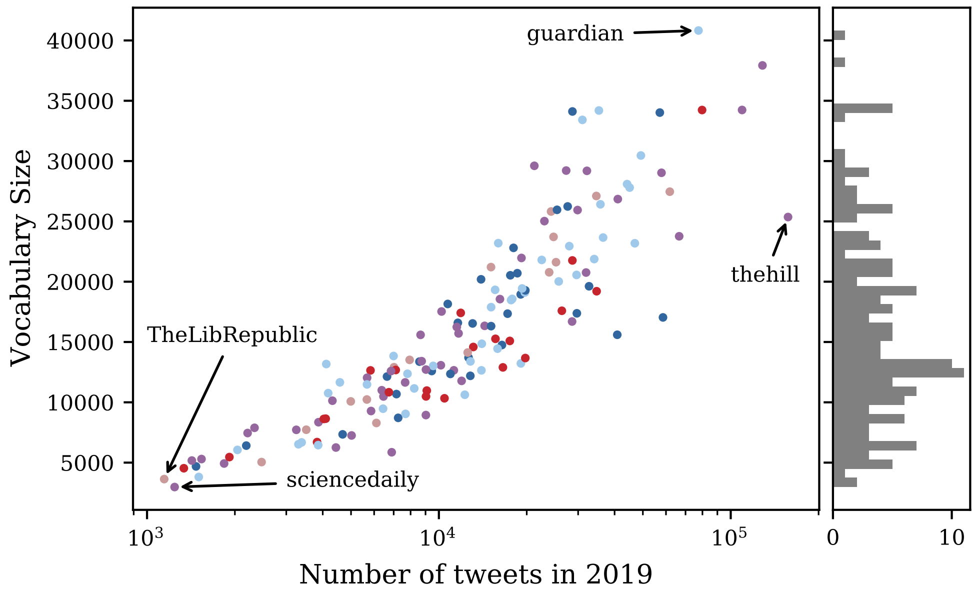

Figure 2.5: Vocabulary size from all tweets produced in 2019 for each news-media Twitter account and the total number of tweets produced.

Chapter 3

Entropy Rate Estimation

Extracting information flows is a problem deeply rooted in information theory. To examine these flows requires tools to quantify and measure this information in the form of natural language. As discussed in chapter 1, the words used to construct language have no qualitative meaning in the context of the numerical analysis. This means that the tools are comparative in nature. Indeed, information theory has been used extensively to compare properties of information in language[**? ?**].

In this chapter, we extend this philosophy in two key ways. We introduce a non-parametric entropy rate estimator and check it's assumptions using real data. We then generalise this entropy rate to a cross entropy rate, developing a tool for analysing information flows.

3.1 Entropy Rate Estimation

Recall Definition 1.0.7 of the entropy rate of a stochastic process. While a useful theoretical tool, this can be very difficult to compute, requiring knowledge of the joint entropy for a infinite set of realisations.

To overcome this, we seek a way to estimate the entropy of the process from a known sequence of data. In 1998 Kontoyianni et al. proved the convergence of a non-parametric entropy estimator in stationary processes [13].

Definition 3.1.1 (Kontoyianni Entropy Rate)**.** For a stochastic process $\mathcal{X} = \{X_i\}$, with n realisations, the entropy rate is given by,

$$H(\mathcal{X}) = \lim_{n\to\infty} \frac{n \log n}{\sum_{i=0}^{n} \Lambda_i}, \tag{3.1}$$

where Λ_i is the length of the shortest subsequence starting at position i that does not appear as a contiguous subsequence in the previous i symbols X_0^i.

This can also be obtained by adding 1 to the longest match-length,

$$\Lambda_i = 1 + \max \left\{ l : X_i^{i+l} = X_j^{j+l}, 0 \leq j \leq N - i, 0 \leq l \leq N - i - j \right\}. \quad (3.2)$$

This idea of using matched sub-sequences of text draws from the original work by Lempel and Ziv [26] in compression algorithms based on coding schemes. These algorithms attempt to compress a sequence down into the smallest possible representation, which at perfect efficiency would be the entropy, H. However these universal coding algorithms have no universal rate of convergence [20, 18] and in practice other approaches are often employed, tailored to the specific application at hand.

The idea of an entropy estimator based on match lengths was originally put forward by Grassberger [9] and proved consistent for independent and identically distributed (i.i.d.) processes and mixing Markov chains [19], stationary processes [12] and more generally to random fields [14].

Wyner and Ziv [25] showed that for every ergodic process the match length Λ_n grows like $\frac{\log n}{H}$ in probability. Extending from this notion Kontoyianni et al. showed the convergence of Equation 3.1 in stationary ergodic processes using the match-length Λ_i. This match-length in Equation 3.2 can be seen as the length of the next phrase to be encoded in the sliding-window Lempel–Ziv algorithm.

Conceptually, this match-length is simple. Figure 3.1 shows the calculation of two match-lengths at different time points of a line from Doctor Seuss. At each index i, the elements immediately proceeding $(i, i+1, i+2, \dots)$ are compared to the history of elements before i. The matches of length k are found such that the elements from j to $j+k$ perfectly match the elements from i to $i+k$, for any $j < i$ where k is then maximised. This search only considers the length of the match, regardless of it's location in the history.

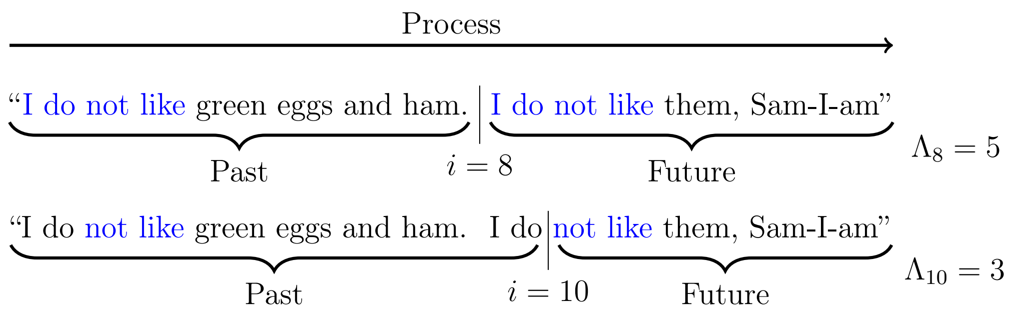
Process
"I do not like green eggs and ham. I do not like them, Sam-I-am"
Past
$i = 8$
Future
$\Lambda_8 = 5$
"I do not like green eggs and ham. I do not like them, Sam-I-am"
Past
$i = 10$
Future
$\Lambda_{10} = 3$

Figure 3.1: An example calculation of the match-length based Λ_i applied to a line from Green Eggs and Ham by Doctor Seuss. Blue text is that which has been matched from past to the future.

Even before it's formalisation by Kontoyianni et al., similar estimators had appeared in the literature applied to experimental data to determine the entropy rates of processes [1, 2, 7, 11].

Moving forward we will assume any any discussion of the *entropy rate* of a single process is assumed to be the *Kontoyianni entropy rate* of that process, unless otherwise stated.

3.2 Assumptions of Entropy Rate Estimation

The proof of convergence of this entropy rate places some limits on the process of investigation. In particular, three assumptions are made for convergence: ergodicity, stationarity and the Doeblin Condition (DC).

The Doeblin Condition is a reasonably weak condition, but is fundamental in the proof of the convergence. Simply put, the DC requires that after an arbitrary r time steps, every state is possible again with positive probability [12]. More formally, the definition is as follows.

Definition 3.2.1 (Doeblin Condition (DC))**.** There exists an integer $r \geq 1$ and a real number $\beta \in (0,1)$ such that, for all $x_0 \in \mathcal{A}$, $\quad P\left\{X_0 = x_0 \mid X_{-\infty}^{-r}\right\} \leq \beta$, with probability one.

Fortunately, as Kontoyianni et al. themselves state, the DC is "certainly satisfied by natural languages" [13]. Given the DC is not a fierce restriction, the condition is easy to satisfy.

In contrast, the assumptions of ergodicity and stationarity are harder to confirm. A long-standing assumption of information theory is that natural language can be modelled by a stationary process [15, 16, 3]. The assumption, while flawed, is well precedented and used again in this work.

While the much of the literature including the work of Kontoyianni assume ergodicity of natural language, some suggest that language should be modelled by a *strongly nonergodic* stationary process [4]. In brief, this contention is founded upon the idea that any given collection of text, such as a book, has a topic containing a small finite subset of words. Suggesting that it's text will inherently not explore the full state space of language. While well founded, our interest is not to look at the entropy rate of the English language as a whole, but rather to look at the entropy rate of individual text streams, which can explore the state space of news under consideration. As such, the assumptions of ergodicity and stationarity appear justified in the context of the problem.

Convergence

With the assumptions of the proof addressed, the challenges of entropy convergence needs to be examined. The entropy rate defined in Equation 3.1 is based upon an infinite set of data to calculate Λ_i's over. In reality, we have finite data, and need to examine the convergence of a modified estimator.

Definition 3.2.2 (Kontoyianni Entropy Rate Estimator)**.** The Kontoyianni Entropy Rate in Definition 3.2.2 can be estimated on a finite stochastic process $\mathcal{X} = \{X_i\}$, with N realisations, by

$$\hat{h} = \frac{N \log N}{\sum_{i=0}^{n} \Lambda_i}, \tag{3.3}$$

where Λ_i is, as earlier, the length of the shortest subsequence starting at position i that does not appear as a contiguous subsequence in the previous i symbols X_0^i.

$$\Lambda_i = 1 + \max\left\{l : X_i^{i+l} = X_j^{j+l}, 0 \le j \le N - i, 0 \le l \le N - i - j\right\}. \tag{3.4}$$

To examine the convergence of the estimator, a model of language can be used to generate sequences of text, upon which we can estimate the entropy rate.

Figure 3.2 shows the convergence of the estimator for a set of i.i.d realisations of a Zipf's law distribution. As discussed in section 1.0.3, Zipf's law is a common tool for generating simple text due to it's similarity to the power-law distributions of vocabulary seen in real corpora. The Zipf's law can be used with a number of exponents, s, where larger exponents tighten the distribution, reducing the observed vocabulary size of the sequence and hence the entropy.

Sequences are generated with 30,000 i.i.d realisations of the Zipf's law and the entropy rate of estimate of the process is calculated at each timestep between 1 and 30,000 using Equation 3.3 applied to only the realisations before that timestep. Estimates of entropy start very low when few realisations are available and rapidly rise as new realisations add complexity. Within the first few hundred timesteps entropy estimates can vary between timesteps as new realisations are added matching or not-matching previous realisations. This is reflected in the high variance between entropies estimates for Zipf process with the same exponent in these early stages. As the number of timesteps included reaches 5000 to 7500 the bias and variance of the estimate are significantly reduced and begin plateauing.

As more timesteps are added, the estimator continues to converge to the asymptotic entropy and the variance between estimates continues to reduce. High complexity sequences take longer to converge to this entropy,

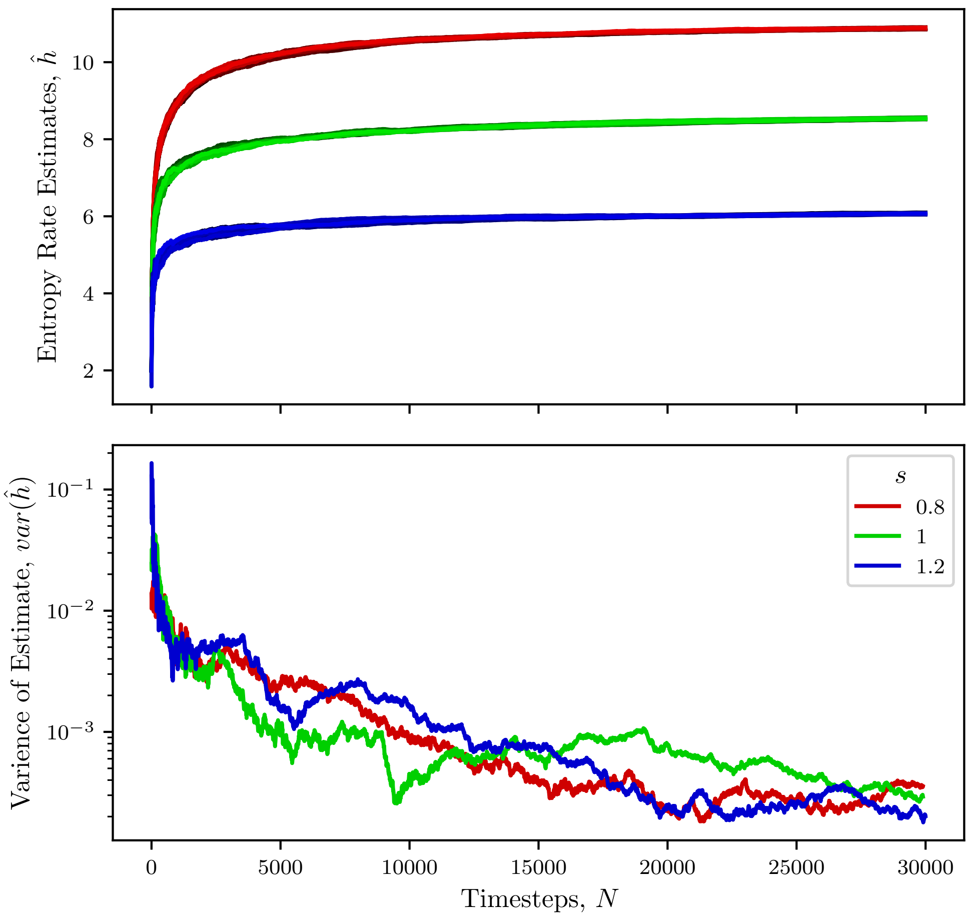

Figure 3.2: Convergence of the Kontoyianni entropy rate estimator on sequences of i.i.d Zipf law realisations with varying Zipf law rates, s.

but achieve suitably small levels of bias and variance within 15000 timesteps even for high entropy sequences. This is an important finding given the speed of calculating these estimates. The algorithm to calculated the match-lengths needed for estimating the entropy is $O(n^3)$ time complexity. As a result, speed takes a significant hit as the number of timesteps is increases. When performing simulations a parsimonious choice of simulation length is advantageous in allowing multiple simulations to be run. Hence, for Zipf law distributions a simulation length of 15000 is deemed sufficient for convergence to the entropy.

To confirm the validity of this approach, we can examine the known entropy rate of the Zipf processes. As shown in Equation 1.0.1, the entropy rate of an i.i.d process is simply the entropy of each element. In the case of a Zipf law the distribution has entropy[1],

$$\frac{s}{H_{N,s}}\sum_{k=1}^{N}\frac{\ln(k)}{k^s}+\ln\left(H_{N,s}\right) \tag{3.5}$$

where $H_{N,s}$ is the Nth generalized harmonic number defined by, $H_{N,s} = \sum_{k=1}^{N}\frac{1}{k^s}$. Many approaches to calculating this entropy use the asymptotic entropy as $N \to \infty$. This draws on the result that $\lim_{N\to\infty} H_{n,s} = \zeta(s)$, where ζ is the Riemann zeta function. While this asymptotic approach works well for numbers well above 1, the Riemann zeta function explodes to infinity at $s \to 1$. As such, the analytic entropy degenerates as $s \to 1$ and is poorly defined for $s < 1$. In contrast, using a finite choice of N results in well defined processes and entropies for all $s > 0$. A choice of $N = 199338$ is made to match the observed total vocabulary size seen in the news-media Twitter data as explored in section 2.1.1. For high values of s, the two asymptotic and finite entropies are very close, 3.18158 and 3.18368 respectively, and only differ significantly as s approaches 1. This finite entropy calculation allows the model to more accurately match the Zipf exponents fitted to real text data, which are often closer to or below 1 [24].

Using simulations of the Zipf process for a variety of values of the exponent s, we compare how the estimated entropy rate compares to the 'true' entropy rate as calculated above. In Figure 3.3 simulations are run for values of s in the range [0.01, 2] with increments of 0.01. High values of s in this range have an increasingly lower entropy, as the skewness of the distribution becomes more extreme. This distribution results in a high numbers of realisations of low rank words creating repeated sequences which lower both the analytic entropy rate and the entropy rate estimate. Values above 2 reduce the entropy rate in vanishingly smaller increments.

For values of s between 2 and 0.5, the entropy rate estimate appears to be a rough upper bound on the true entropy rate of the process. This upper bound is only achieved with sufficient lengths of sequences such that the estimator can converge to this upper bound. Indeed, given sufficient length the variance of the estimates on sequences drawn from the same distribution is very low. This is in contract to the bias of the estimate, which varies with the changing exponent.

When s becomes lower than 0.5, the Zipf law distribution becomes more evenly distributed with reduced skew. This results in a larger probability of

[1]Proof: The probability of a word of rank k being selected from a pool of N elements using exponent s is $(k^s H_{N,s})^{-1}$. Hence, the entropy of each individual element is $\sum_{k=1}^{N}(k^s N_{N,s})^{-1}\ln\left((k^s N_{N,s})^{-1}\right)$. Using $H_{N,s} = \sum_{k=1}^{N}\frac{1}{k^s}$, this can be rearranged to $\frac{s}{H_{N,s}}\sum_{k=1}^{N}\frac{\ln(k)}{k^s}+\ln\left(H_{N,s}\right)$.

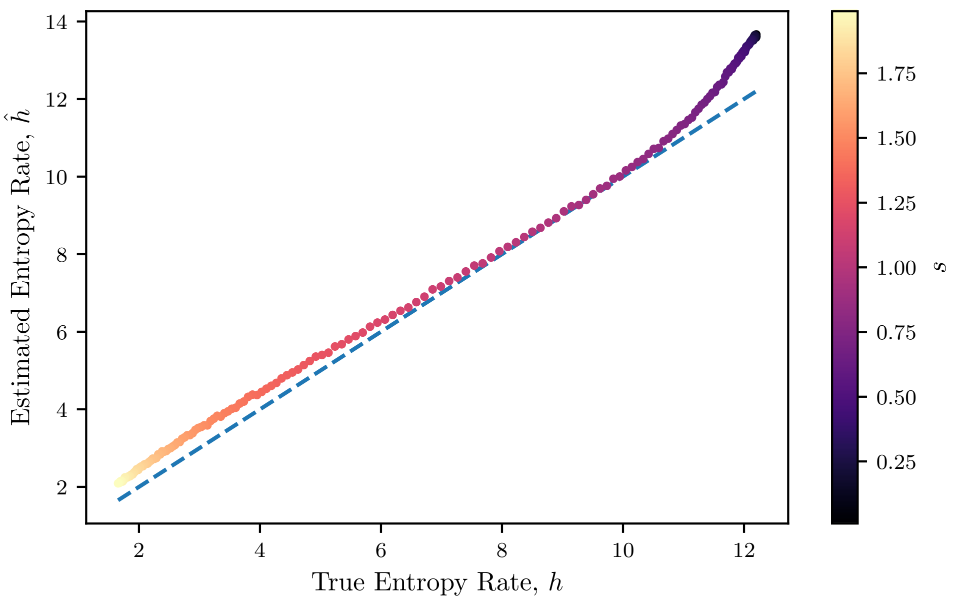

Figure 3.3: Estimated entropy rates and analytic entropy rates of sequences of 20,000 i.i.d Zipf law random variables with exponent s. Dashed line represents the true entropy rate equalling the entropy rate estimate. As values for s approach 0 the high variance of the distributions results in poor estimates due to the finite sample of the Zipf law.

low rank word occurrences, producing a process where many words appear very few times. As a result, the finite nature of the sequence results in a entropy rate estimate that grows faster than the true entropy, increasing the bias for these high entropy sequences.

In general, the convergence of the estimator is sufficient, with a slight caveat. While the estimator convergences to an estimate tightly with very little variance, the bias of the estimate is not constant and varies with the complexity of the sequences. While this finding is itself interesting and warrants future work, the estimator is both consistent and it's estimates appear monotonic with the true entropy rate. As such, we will use this estimator to approximate the true entropy rate moving forward, with a cautious eye to the possible effects of this inconsistent bias.

To extend from this result using the Zipf law process, we apply the same approach using the news-source Twitter data. 5000 tweets are drawn uniformly from the collection of all tweets from all news-media outlets. These tweets are tokenized and concatenated into a single sequence of natural language text ranging from 85000 tokens to 90000.

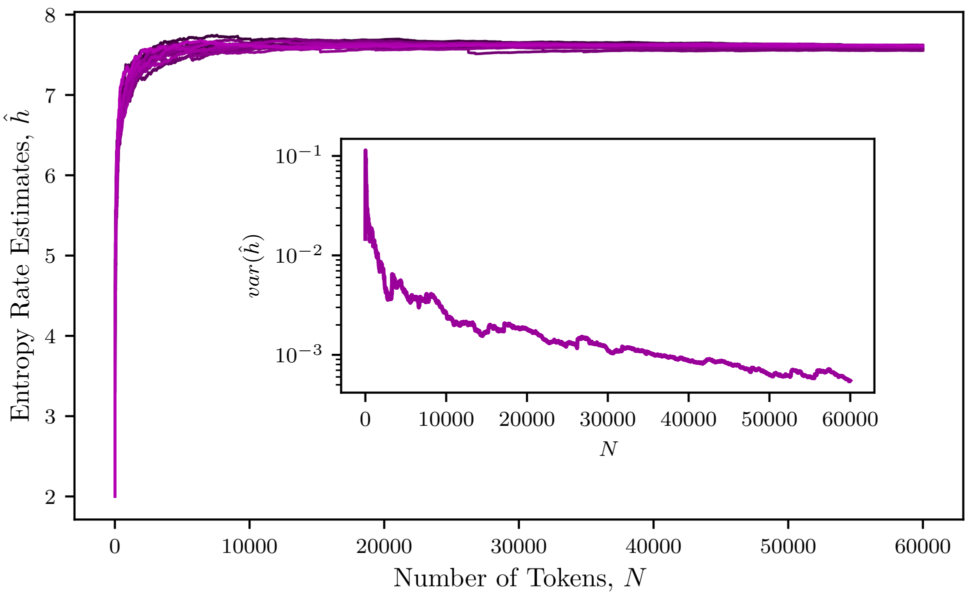

Figure 3.4: Convergence of the Kontoyianni entropy rate estimator on sequences words generated by drawing tweets uniformly without replacement from the pool of all tweets produced by all news-media organisations.

Unlike the case of Zipf, we cannot show a true entropy rate for this distribution, but can demonstrate it's convergence. Each sample has the entropy rate calculated using only the first N tokens, up to the length of 60,000 tokens. Figure 3.4 shows the clear convergence of the estimator of the real data, approaching a entropy rate of 7.53. As with the Zipf simulations, the variance of the estimated entropy rates of the samples reduces as more tokens are included in the estimate. The real data takes longer to converge, needing up to 50,000 tokens to achieve a variance of less that 10^{-3}.

Self Entropy Rate Summary

Before moving away from the self entropy rate, let's take one more opportunity to visualise the how this Kontoyianni entropy rate estimator works in the real data. Figure 3.5 shows a simplified version of the conceptual self entropy rate calculation. For a given Twitter user, tweets appear sequentially, separated in time. At any given time, the content of the immediate future of that user's tweets is compared to the entire history of the tweets before that time. This process is then repeated for all possible times in the data. In essence, the calculation of the Λ_i's is a repeated examination of how much of

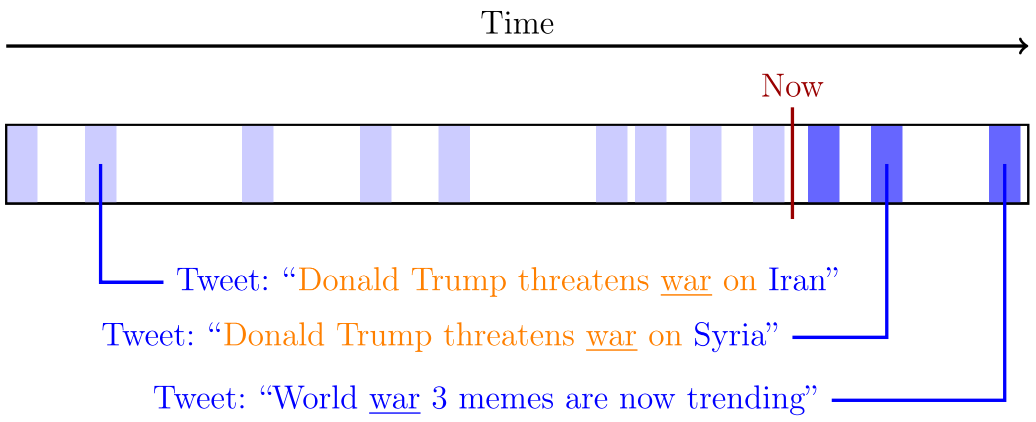
Time
Now
Tweet: “Donald Trump threatens war on Iran”
Tweet: “Donald Trump threatens war on Syria”
Tweet: “World war 3 memes are now trending”

Figure 3.5: A conceptual diagram of self entropy rate estimation using the Kontoyianni entropy rate estimation. Tweets shown as blue rectangles are positioned in time and contain textual content. Content proceeding the position *Now* will have snippets of text matched with text from the history of the process, denoted by orange and unlined text. These text matches are used to calculate the Λ_i's which inform the entropy estimate.

the immediate complexity at timestep i can be described using the history of the sequence. In total, this estimator provides an average of how many bits are needed to describe the future of this process given it's past at any point.

3.3 Cross Entropy Rate

To create a notion of information flow, we need to move beyond looking at individual sources in isolation. To do so, we need a tool of comparison between sources rooted in our tools from information theory. We find such a tool in a generalisation to a Kontoyianni cross entropy rate.

Similar to the extension of entropy,

$$H(X) = \sum_{x \in \mathcal{X}} p(x) \log p(x) = -\mathbb{E}[\log P(X)]$$

to cross entropy,

$$H(p, q) = \sum_{x \in \mathcal{X}} p(x) \log q(x) = -\mathbb{E}_p[\log q(X)],$$

in Definition 1.0.4, we can generalise our notion of Kontoyianni entropy rate from Definition 3.2.2 to a *cross* entropy rate which we will call the Kontoyianni cross entropy rate.

Definition 3.3.1 (Kontoyianni Full Cross Entropy Rate)**.** The cross entropy rate of a target process $\mathcal{T}$ coded from a source process $\mathcal{S}$ can be estimated via,

$$H(\mathcal{T}||\mathcal{S}) = \frac{N_{\mathcal{T}} \log_2 N_{\mathcal{S}}}{\sum_{i=1}^{N_{\mathcal{T}}} \Lambda_i(\mathcal{T}|\mathcal{S})} \tag{3.6}$$

Where $N_{\mathcal{X}}$ is the length of process $\mathcal{X}$ and $\Lambda_i(\mathcal{T}|\mathcal{S})$ is given by the shortest subsequence starting at position i in the target $\mathcal{T}$ that does not appear as a contiguous subsequence anywhere in the source $\mathcal{S}$.

$$\Lambda_i(\mathcal{T}|\mathcal{S}) = \max\left\{l : T_i^{i+l} = S_j^{j+l}, 0 \leq j \leq N_{\mathcal{S}}, 0 \leq l \leq \min(N_{\mathcal{S}} - j, N_{\mathcal{T}} - i)\right\}, \tag{3.7}$$

where T_a^b and S_a^b are continuous subsequences starting from index a to index b of the target, $\mathcal{T}$, and source, $\mathcal{S}$, processes respectively.

This approach to a cross entropy matches segments of text in the target to segments of text anywhere in the source in the same manner that the Kontoyianni entropy rate matched segments of text in the future of a process from a index, i, to the history before the index. In contrast to the entropy rate estimate, which asking how much information was needed on average to *describe the future of a source from it's past*, this cross entropy rate estimate is asking how much information is needed on average to *describe the target given full knowledge of the source.*

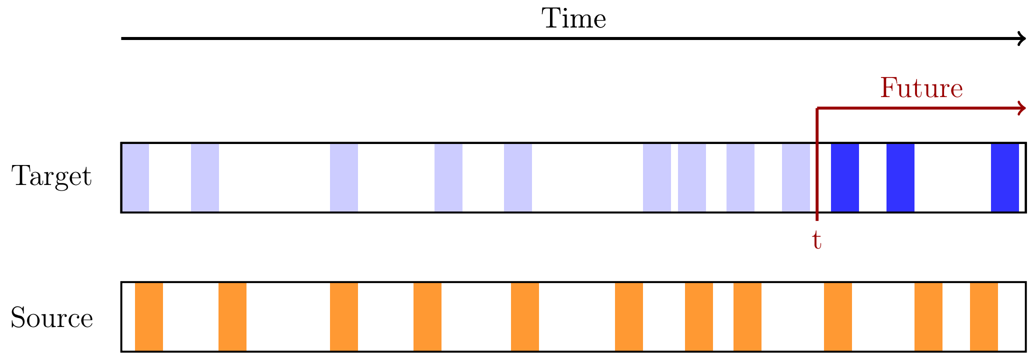

Figure 3.6: A conceptual diagram of **Kontoyianni full cross entropy rate** estimation. Tweets shown as rectangles are positioned in time for both a target and source, containing textual information. Content in the source is matched with content in the immediate future of the target for a given time point, t, to calculate match-lengths. This time point is shifted along the target timeline to average match-lengths and calculate the full cross entropy rate.

This full knowledge over all of the process gives the estimator the 'Full' in it's title, but presents a impropriety. In the context of our problem, this estimator is cheating by viewing the future of news through the lens of the source process.

As Figure 3.6 illustrates, the text subsequence match in the target could be drawn from future time points in the source. Restated, the cross entropy rate will, in-part, be describing how much information you need to encode the future of a piece of target news already knowing the future of the news from another source. While this may be an interesting insight in itself, it doesn't probe the underlying process of *information flow* with which this thesis focuses.

Rather than looking at the entire lifetime of the source during the matching calculations, we can reduce our search space to the text that occurred in the *past* of the source. To achieve this we use an important piece of our data, the time that tweets occurred. For each word in the target process, T_i has an associated time with it, $t(T_i)$. When matching the future of $\mathcal{T}$, starting from an index i, we can reduce the source process, $\mathcal{S}$ to only the words that were themselves tweeted before time $t(T_i)$.

Put simply, we can alter the Kontoyianni full cross entropy rate to a time-synced cross entropy rate by replacing the full process, $\mathcal{S}$, with a time reduce source process $\mathcal{S}_{\leq t(T_i)}$. This can be seen visually in Figure 3.7 and is formally defined as follows.

Definition 3.3.2 (Kontoyianni Time-synced Cross Entropy Rate)**.** The time-synced cross entropy rate of a target process $\mathcal{T}$ coded from a source process $\mathcal{S}$ can be estimated via,

$$H(\mathcal{T}||\mathcal{S}) = \frac{N_{\mathcal{T}} \log_2 N_{\mathcal{S}}}{\sum_{i=1}^{N_{\mathcal{T}}} \Lambda_i(\mathcal{T}|\mathcal{S}_{\leq t(T_i)})} \tag{3.8}$$

Where $\Lambda_i(\mathcal{T}|\mathcal{S}_{\leq t(T_i)})$ is given by the shortest subsequence starting at position i in target $\mathcal{T}$ that does not appear as a contiguous subsequence in the time reduced source $\mathcal{S}_{\leq t(T_i)}$ where,

$$\mathcal{S}_{\leq t(T_i)} = \{S_j | t(S_j) \leq t(T_i) \forall i\}. \tag{3.9}$$

Which gives,

$$\begin{aligned}\Lambda_i(\mathcal{T}|\mathcal{S}_{\leq t(T_i)}) = \max\{l : T_i^{i+l} = S_j^{j+l}, 0 \leq j \leq N_{\mathcal{S}},\\ 0 \leq l \leq \min(N_{\mathcal{S}} - j, N_{\mathcal{T}} - i)\},\end{aligned}$$

where T_a^b and S_a^b are continuous subsequences starting from index a to index b of the target, $\mathcal{T}$ process, and the time reduced source, $\mathcal{S}_{\leq t(T_i)}$, respectively.

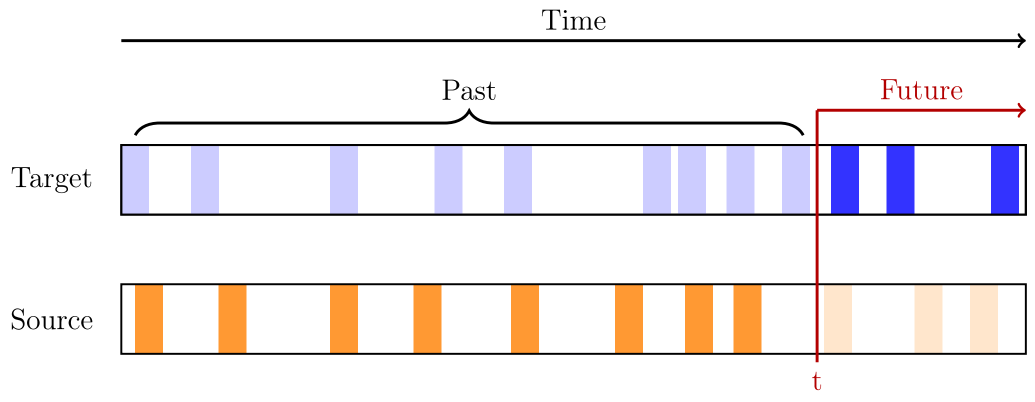

Figure 3.7: A conceptual diagram of **Kontoyianni time-synced cross entropy rate** estimation. Tweets shown as rectangles are positioned in time for both a target and source, containing textual information. Content in the source that occurs before time t is matched with content in the immediate future of the target from t to calculate match-lengths. This time point is shifted along the target timeline to average match-lengths and calculate the time-synced cross entropy rate.

This time-synced entropy rate is testing not just the differences in the language processes of the source and target, but also measuring what information in the target is present in the source's history. This is an important distinction, as it allows us to probe a very important aspect of our data, namely, the time in which news is created.

If a piece of information appears earlier in the source than in the target, it will be detected during the match length search, resulting in a lower entropy. This is to say, in the context of news, if the source breaks a story first, *less* information is required to describe the subsequent news output from the target.

Conversely, if a target produces a piece of information before the source, then that information will not appear in the history of the time-synced source during the match-length search. This will result in lower values of Λ_i for that piece of information, which raises the cross entropy rate.

From this, we can find that, on average, if a source produces information earlier than a target, the cross entropy rate, $\hat{h}(\mathcal{T}|\mathcal{S})$, will be lower than if the target produces information earlier than the source. This method of examining who produces information first can be extended into a notion of *information flow*, a discussion we will leave for chapter 4.

3.3.1 Validating the Assumptions Cross Entropy Estimation

To validate these new cross entropy rate estimators a similar process is performed as with the entropy rate. Fortunately, many of the assumptions transfer over neatly.

In the case of ergodicity, stationarity and the Doeblin Condition, all three are properties of the process under investigation, which we argued above are well founded. We introduce a additional condition on the processes for the sake of cross entropy rates, namely that the processes have the same state space. This additional condition extends the earlier conditions to apply jointly between both the source and target processes. While not all news-media Twitters will have the same realisations of word in the corpus, the processes could be reasonably thought to have the same state space, as all process are using English and discussing similar topics.

This then leads naturally to the next question of convergence. Following a sample process of uniform withdrawals from a distribution will result in functionally similar convergence to the entropy rate above. Indeed, this can be seen in Figure 3.8, where tweets are sequences are draw uniformly from the pool of all tweets and cross entropy rates, denoted $\hat{h}_\times$ to distinguish from the entropy rate estimates $\hat{h}$, are calculated. This figure is, as expected, functionally similar to Figure 3.4 from the entropy rate section above. The cross entropy rate calculation is fundamentally no different from the entropy rate calculation as a randomly selection history of another process is just a useful and a randomly selected history of another process when both draw from the same distribution.

A more nuanced investigation of the cross entropy convergence emerges when we utilise processes with *different* distributions. Figure 3.9 does exactly this. Using pairs of exponents for different Zipf distributions simulations of 30,000 long i.i.d processes are made with separate target and source processes. The cross entropies rates are estimated across these pairs. When the exponents are equal, $s_{source} = s_{target}$, we observe the same entropy and convergence pattern as we would for a single i.i.d process with that exponent.

When $s_{source} \neq s_{target}$ the entropy rates vary, and are not always similar to the entropy rate of the target or the source.

3.3.2 Predictability

Perhaps this would be a good place to generalise the notion of cross maximal predictability.

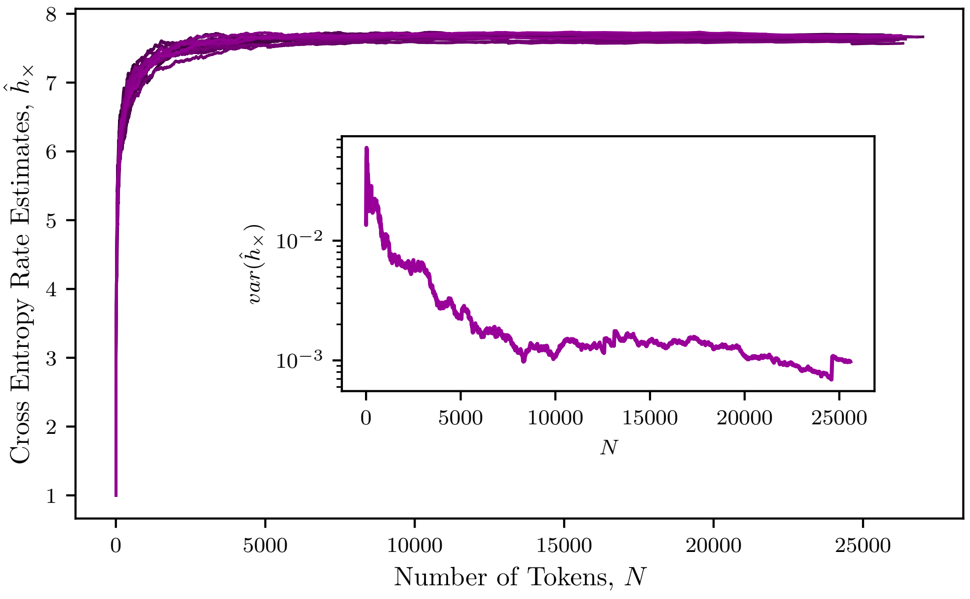

Figure 3.8: Convergence of the Kontoyianni time-synced cross entropy rate estimator on pairs of sequences independently generated by drawing tweets uniformly without replacement from the pool of all tweets produced by all news-media organisations.

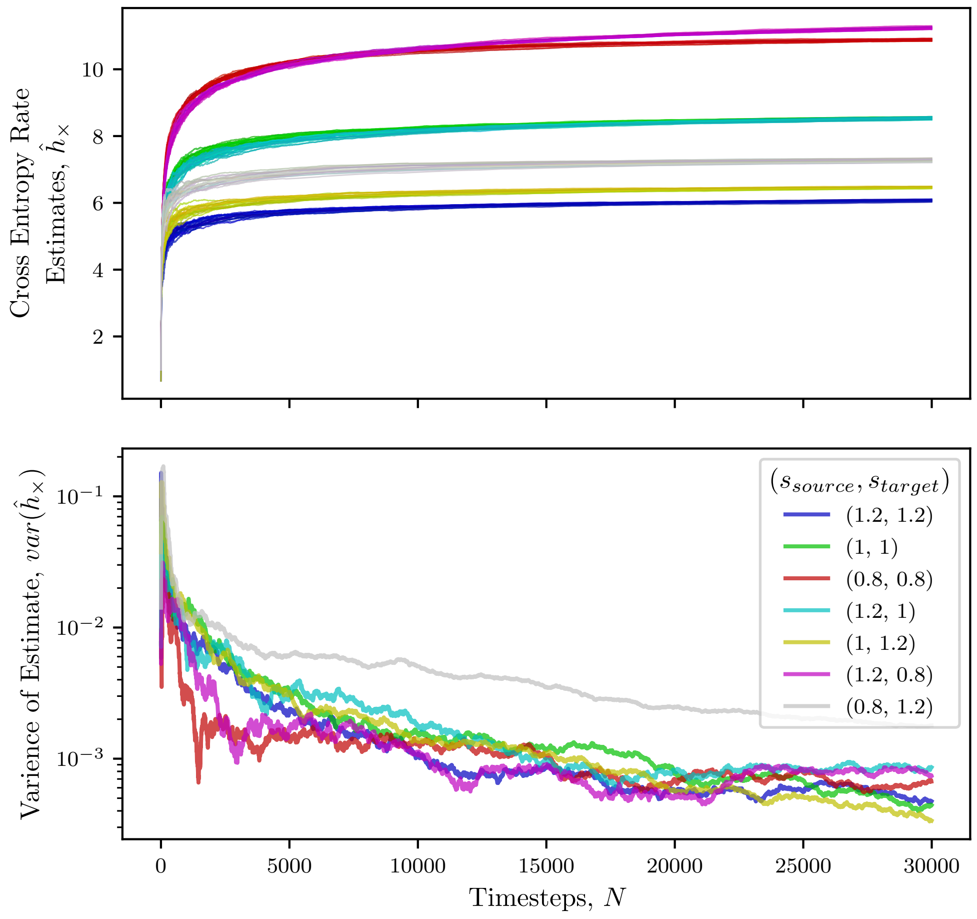

Figure 3.9: Convergence of the Kontoyianni time-synced cross entropy rate estimator on pairs sequences of i.i.d Zipf law realisations with varying pairs of Zipf law rates, s_{source} and s_{target}.

3.3.3 A Note on Package Development

As stated earlier, a key challenge in estimating these Kontoyianni cross entropy rates is calculation speed. With time complexities of $O(n^3)$ on the number of input tokens, fast code is necessary to allow for estimation using long sequences. To achieve this speed and contribute to this field of work more broadly, a speed-focused open source package was developed to help researchers efficiently and easily calculate entropy rates such as those discussed above. The important snippets from code contributions of the package are available in Appendix A and this section will outline two key tools used in speeding the code up.

The fundamental complexity limitations of the algorithm mean that speed improvements need to come from smart implementation.

I'm thinking of putting a section here that just talks about the need for speed of computation and highlights some of the tricks used to speed it up. E.g. Fowler–Noll–Vo hashing, JIT compiling. Do you think this is appropriate to include and if so at what level of detail.

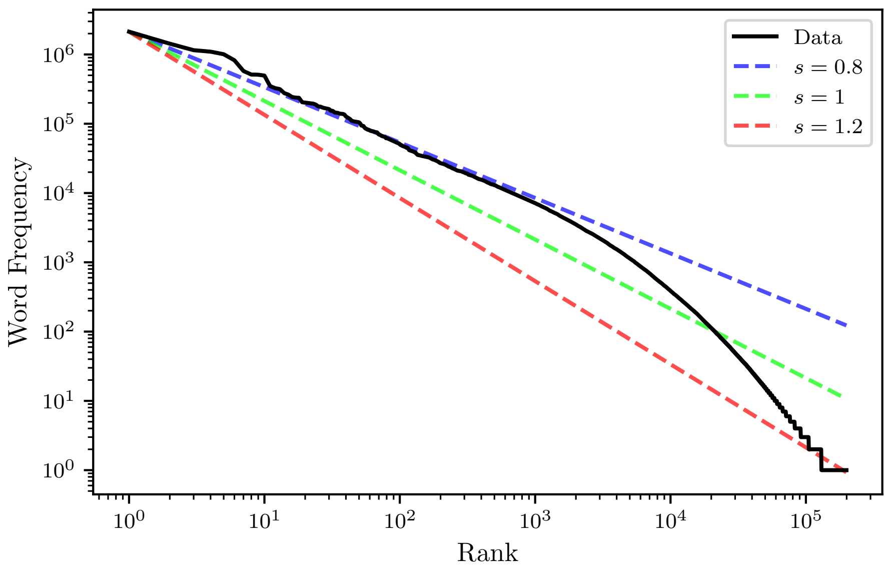

Figure 3.10: [[TS: Bonus: Included for your reviewing, this is a figure that will soon be added to the data chapter.]] Word frequency of tokens in the corpus of all tweets produced by all news sources compared the rank of the token by frequency. Zipf law distributions are also shown for varying exponents, s.

Chapter 4

Fuck I really need to do more writing

Appendix A

ProcessEntropy: Open source high-speed entropy calculation package.

Listing A.1: Kontoyianni cross entropy rate estimation code from the package 'ProcessEntropy' available on Github.

```
import numba
from numba import jit, prange
import numpy as np
import math
import nltk

from ProcessEntropy.Preprocessing import *


@jit(nopython=True, fastmath=True)
def find_lambda_jit(target, source):
    """
    Finds the longest subsequence of the target array,
    starting from index 0, that is contained in the source array.
    Returns the length of that subsequence + 1.

    i.e. returns the length of the shortest subsequence starting at 0
    that has not previously appeared.

    Args:
        target: NumPy array, preferable of type int.
        source: NumPy array, preferable of type int.

    Returns:
        Integer of the length.

    """

    source_size = source.shape[0]-1
    target_size = target.shape[0]-1
    t_max = 0
    c_max = 0

    for si in range(0, source_size+1):
        if source[si] == target[0]:
            c_max = 1
            for ei in range(1,min(target_size, source_size - si+1)):
                if(source[si+ei] != target[ei]):
                    break
                else:
                    c_max = c_max+1

            if c_max > t_max:
                t_max = c_max

    return t_max+1




@jit(parallel=True)
def get_all_lambdas(target, source, relative_pos, lambdas):
    """
    Finds all the the longest subsequences of the target,
    that are contained in the sequence of the source,
    with the source cut-off at the location set in relative_pos.

    See function find_lambda_jit for description of
        Lambda_i(target|source)

    Args:
        target: Array of ints, usually corresponding to hashed words.

        source: Array of ints, usually corresponding to hashed words.

        relative_pos: list of integers with the same length as target ↘
            denoting the
            relative time ordering of target vs. source. These integers ↘
                tell us the
            position relative_pos[x] = i in source such that all symbols↘
                 in source[:i]
            occurred before the x-th word in target.

        lambdas: A pre-made array of length(target), usually filled with↘
             zeros.
            Used for efficiency reasons.

    Return:
        A list of ints, denoting the value for Lambda for each index in ↘
            the target.

    """
    i = 0
    while relative_pos[i] == 0: # Preassign first values to avoid check
        lambdas[i] = 1
        i+=1

    # Calculate lambdas
    for i in prange(i, len(target)):
        lambdas[i] = find_lambda_jit(target[i:], source[:relative_pos[i↘
            ]])

    return lambdas

```

```python

def timeseries_cross_entropy(time_tweets_target, time_tweets_source, please_sanitize = True, get_lambdas = False):
    """
    Finds the cross entropy H_cross(target|source) where the processes are embedded in time.

    i.e. How many bits we would need to encode the data in target
    using information in the source that is before the current time.


    This is described mathematically in [1] as,

    T = target
    S = source

    $$
    \hat{h}_{ \times}(T | S)=\frac{N_{T} \log _{2} N_{S}}{\sum_{i=1}^{N_{T}} \Lambda_{i}(T | S)}
    $$


    Args:
        time_tweets_target: A list of tuples with (time, tweet_content).
            This is the stream of new information that we can testing the ability to encode.
            If please_sanitize = True (default) then tweet_content can be a string.

        time_tweets_source: A list of tuples with (time, tweet_content).
            This is the stream of previous information from which we try to encode the target.

        please_sanitize: Option to have the tweet string converted to numpy int arrays for speed.
            If False, please sanitize tweet into a list of tokenized words, ideally converting these to ints,
            via a hash.

        get_lambdas: Boolean choice to return the list of all calculated Lambda values for each point
            in target. Usually used for debugging.

    Return:
        The cross entropy as a float


    [1] I. Kontoyiannis, P.H. Algoet, Yu.M. Suhov, and A.J. Wyner. Nonparametric entropy
    estimation for stationary processes and random fields, with applications to English text.
    IEEE Transactions on Information Theory, May 1998.

    Credit to Bagrow and Mitchell for code ideas that I've stolen for this function.

    """

    # Decorate tweets (so we can distinguish the users), before sorting in time:

    if please_sanitize: # Option to have the tweet string converted to
```

```
          numpy int arrays for speed.
          # tweet_to_hash_array fucntion can be found in package.
          decorated_target = [ (time,"target",tweet_to_hash_array(tweet)) ↘
              for time,tweet in time_tweets_target ]
          decorated_source = [ (time,"source",tweet_to_hash_array(tweet)) ↘
              for time,tweet in time_tweets_source ]
      else:
          decorated_target = [ (time,"target",tweet) for time,tweet in ↘
              time_tweets_target ]
          decorated_source = [ (time,"source",tweet) for time,tweet in ↘
              time_tweets_source ]

      # Join time series:
      time_tweets = decorated_target + decorated_source

      # Sort in place by time:
      time_tweets.sort()

      # Loop over combined tweets and build word vectors and target->↘
          source relative_pos:
      target, source, relative_pos = [], [], []
      for time,user,tweet in time_tweets:
          words = tweet
          if user == "target":
              target.extend(words)
              relative_pos.extend( [len(source)]*len(words) )
          else:
              source.extend(words)


      target = np.array(target, dtype = np.uint32)
      source = np.array(source, dtype = np.uint32)
      relative_pos = np.array(relative_pos, dtype = np.uint32)
      lambdas = np.zeros(len(target), dtype = np.uint32) # Premake for ↘
          efficiency

      lambdas = get_all_lambdas(target ,source, relative_pos, lambdas)

      if get_lambdas:
          return lambdas
      return  len(target)*math.log(len(source),2) / np.sum(lambdas)


@jit(parallel=True)
def conditional_entropy(target, source,  get_lambdas = False):
      """
      Finds the simple conditional entropy as a process.

      Entropy of target process conditional on full knowledge of states of↘
           source process.

      Args:
      target: A 1-D numpy array of integers.

          This is the stream of new information that we can testing the ↘
              ability to encode.

      source: A 1-D numpy array of integers.
              This is the stream of previous information from which we try↘
                   to encode the target.

         get_lambdas: Boolean choice to return the list of all calculated ↘
```

```
                Lambda values for each point
                 in target. Usually used for debugging.

        Return:
            The conditional entropy as a float
        """
        lambdas = np.zeros((1,len(target)))
        for i in prange(0, len(target)):
            lambdas[i] = find_lambda_jit(target[i:], source)

        if get_lambdas:
            return lambdas
        else:
            return  len(target)*math.log(len(source),2) / np.sum(lambdas)
```

Listing A.2: Kontoyianni entropy rate estimation code from the package 'ProcessEntropy' available on Github.

```
import numba
from numba import jit, prange
import numpy as np
import math
import nltk

from ProcessEntropy.Preprocessing import *

@jit(nopython = True)
def get_all_self_lambdas(source, lambdas):
    """
        Internal function.

        Finds the Lambda value for each index in the source.

        Lambda value denotes the longest subsequence of the source,
        starting from the index, that in contained contigiously in the ↘
            source,
        before the index.

    Args:
        source: Arry of ints, usually corresponding to hashed words.

        lambdas: A premade array of length(target), usually filled with ↘
            zeros.
            Used for efficiency reasons.

    Return:
        A list of ints, denoting the value for Lambda for each index in ↘
            the target.

    """

    N = len(source)

    for i in prange(1, N):

        # The target process is everything ahead of i.
        t_max = 0
        c_max = 0

        for j in range(0, i): # Look back at the past
            if source[j] == source[i]: # Check if matches future's next ↘
```

```python
                element
                c_max = 1
                for k in range(1,min(N-i, i-j)): # Look through more of future
                    if source[j+k] != source[i+k]:
                        break
                    else:
                        c_max = c_max+1

                if c_max > t_max:
                    t_max = c_max

        lambdas[i] = t_max+1

    return lambdas



def self_entropy_rate(source, get_lambdas = False):
    """
    Args:
        source: The source is an array of ints.
    Returns:
        The non-parametric estimate of the entropy rate based on match lengths.


    $$
    \hat{h}(S)=\frac{N \log _{2} N{\sum_{i=1}^{N \Lambda_{i}(S)}
    $$

    This is described mathematically in [1] as,

    [1] I. Kontoyiannis, P.H. Algoet, Yu.M. Suhov, and A.J. Wyner. Nonparametric entropy
    esti mation for stationary processes and random fields, with applications to English text.
    IEEE Transactions on Information Theory, May 1998.

    """

    N = len(source)
    source = np.array(source)
    lambdas = np.zeros(N)
    lambdas = get_all_self_lambdas(source, lambdas)

    if get_lambdas:
        return lambdas
    else:
        return N*math.log(N,2) / np.sum(lambdas)


def text_array_self_entropy(token_source):
        """
        This is a wrapper for `self_entropy_rate' to allow for raw text to be used.

        Args:
                token_source: A list of token strings (hint: a list of words).

        Returns:
```

```
    The non-parametric estimate of the entropy rate based on match lengths.

    """
    return self_entropy_rate(np.array([fnv(word)  for word in token_source]))

def tweet_self_entropy(tweets_source):
    """
    This is a wrapper for `self_entropy_rate' to allow for raw tweets to be used.

    Args:
        tweets_source: A list of long strings (hint: a list of tweets).
            If it detects that you have added a list of time, tweet pairs
            (as in timeseries_cross_entropy) it will recover.

    Returns:
    The non-parametric estimate of the entropy rate based on match lengths.

    """
    source = []

    if type(tweets_source[0]) == tuple:
        # This is for the case of a
        for time, text in tweets_source:
            source.extend(tweet_to_hash_array(text))
    else:
        for text in tweets_source:
            source.extend(tweet_to_hash_array(text))

    return self_entropy_rate(source)
```

Listing A.3: Code for calculating predictability from the package 'ProcessEntropy' available on Github.

```
# This is a bonus file to help convert to predictabilties.

from scipy.optimize import fsolve
import numpy as np
import math

from ProcessEntropy.SelfEntropy import *
from ProcessEntropy.CrossEntropy import *

def predictability(S,N, inital_guess = 0.5):
    """Finds the value of the predicatbility for a process with an entropy rate S and a vocabular size N."""
    # explodes for small values of N or large values of S :(
    try:
        f = lambda Pi : S + Pi*math.log(Pi,2) + (1 - Pi)*math.log(1 - Pi,2) - (1 - Pi)*math.log(N-1,2)
        PiMax = fsolve(f,inital_guess)
    except:
        PiMax = 0
    return float(PiMax)

```

```

def process_predictability(process):
    """Calculates the predictability of the process. """
    entropy = nonparametric_entropy_estimate(process)
    N = len(set(process))
    return calc_predictability(entropy,N)



def cross_predictability(target,source):
    """Calculates the predictability of the target given the information↘
        in the source."""
    cross_entropy = timeseries_cross_entropy(target,source)
    N = len(set(target)) # THIS IS WHERE I"M NOT SURE WHAT N TO USE
    return predictability(entropy,N)


def surprise(probability):
    """Returns surprise value for given probability"""
    return log(1/probability,2)
```

Listing A.4: Code for preprocessing tweet and text data from the package ‘ProcessEntropy’ available on Github.

```
# This is a bonus file to help convert to predictabilties.

from scipy.optimize import fsolve
import numpy as np
import math

from ProcessEntropy.SelfEntropy import *
from ProcessEntropy.CrossEntropy import *

def predictability(S,N, inital_guess = 0.5):
    """Finds the value of the predicatbility for a process with an ↘
        entropy rate S and a vocabular size N."""
    # explodes for small values of N or large values of S :(
    try:
        f = lambda Pi : S + Pi*math.log(Pi,2) + (1 - Pi)*math.log(1 - Pi↘
            ,2) - (1 - Pi)*math.log(N-1,2)
        PiMax = fsolve(f,inital_guess)
    except:
        PiMax = 0
    return float(PiMax)


def process_predictability(process):
    """Calculates the predictability of the process. """
    entropy = nonparametric_entropy_estimate(process)
    N = len(set(process))
    return calc_predictability(entropy,N)



def cross_predictability(target,source):
    """Calculates the predictability of the target given the information↘
        in the source."""
    cross_entropy = timeseries_cross_entropy(target,source)
    N = len(set(target)) # THIS IS WHERE I"M NOT SURE WHAT N TO USE
    return predictability(entropy,N)

```

```

def surprise(probability):
    """Returns surprise value for given probability"""
    return log(1/probability,2)
```

Bibliography

[1] S. Chen and J. H. Reif. Using difficulty of prediction to decrease computation: Fast sort, priority queue and convex hull on entropy bounded inputs. In *Proceedings of 1993 IEEE 34th Annual Foundations of Computer Science*, pages 104–112. IEEE, 1993.

[2] S. Chen and J. H. Reif. Fast pattern matching for entropy bounded text. In *Proceedings DCC'95 Data Compression Conference*, pages 282–291. IEEE, 1995.

[3] T. M. Cover and J. A. Thomas. *Elements of Information Theory.* John Wiley & Sons, Nov. 2012. ISBN 978-1-118-58577-1.

[4] Ł. Dębowski. Is Natural Language a Perigraphic Process? The Theorem about Facts and Words Revisited. *Entropy*, 20(2):85, Feb. 2018. doi: 10.3390/e20020085.

[5] B. Diggs-Brown. *Strategic Public Relations: An Audience-Focused Approach.* Cengage Learning, p. 48, 2011.

[6] R. M. Fano and D. Hawkins. Transmission of Information: A Statistical Theory of Communications. *American Journal of Physics*, 29:793–794, Nov. 1961. ISSN 0002-9505. doi: 10.1119/1.1937609.

[7] M. Farach, M. Noordewier, S. Savari, L. Shepp, A. Wyner, and J. Ziv. On the entropy of DNA: Algorithms and measurements based on memory and rapid convergence. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 48–57, 1995.

[8] J. Gable, S. McDonald, and J. Blades. Media Bias Ratings AllSides. https://www.allsides.com/media-bias/media-bias-ratings, 2019.

[9] P. Grassberger. Estimating the information content of symbol sequences and efficient codes. *IEEE Transactions on Information Theory*, 35(3): 669–675, May 1989. ISSN 00189448. doi: 10.1109/18.30993.

[10] M. Hu, S. Liu, F. Wei, Y. Wu, J. Stasko, and K.-L. Ma. Breaking news on twitter. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 2751–2754, Austin, Texas, USA, May 2012. Association for Computing Machinery. ISBN 978-1-4503-1015-4. doi: 10.1145/2207676.2208672.

[11] P. Juola. What can we do with small corpora? Document categorization via cross-entropy. In *Proceedings of an Interdisciplinary Workshop on Similarity and Categorization, Department of Artificial Intelligence, University of Edinburgh, Edinburgh, UK*, 1997.

[12] I. Kontoyiannis and Y. M. Suhov. Prefixes and the entropy rate for long-range sources. In *IEEE International Symposium on Information Theory*, pages 194–194. INSTITUTE OF ELECTRICAL ENGINEERS INC (IEEE), 1994.

[13] I. Kontoyiannis, P. Algoet, Y. Suhov, and A. Wyner. Nonparametric entropy estimation for stationary processes and random fields, with applications to English text. *IEEE Transactions on Information Theory*, 44(3):1319–1327, May 1998. ISSN 00189448. doi: 10.1109/18.669425.

[14] A. N. Quas. An entropy estimator for a class of infinite alphabet processes. *Theory of Probability & Its Applications*, 43(3):496–507, 1999.

[15] C. E. Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[16] C. E. Shannon. Prediction and entropy of printed English. *Bell system technical journal*, 30(1):50–64, 1951.

[17] E. Shearer and E. Grieco. Americans Are Wary of the Role Social Media Sites Play in Delivering the News, Oct. 2019.

[18] P. Shields and B. Weiss. Universal redundancy rates for the class of B-processes do not exist. *IEEE transactions on information theory*, 41 (2):508–512, 1995.

[19] P. C. Shields. Entropy and prefixes. *The Annals of Probability*, pages 403–409, 1992.

[20] P. C. Shields. Universal redundancy rates do not exist. *IEEE transactions on information theory*, 39(2):520–524, 1993.

[21] C. Song, Z. Qu, N. Blumm, and A.-L. Barabasi. Limits of Predictability in Human Mobility. *Science*, 327(5968):1018–1021, Feb. 2010. ISSN 0036-8075, 1095-9203. doi: 10.1126/science.1177170.

[22] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and S. . . Contributors. SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python. *arXiv:1907.10121 [physics]*, July 2019.

[23] F. Vis. Twitter as a Reporting Tool for Breaking News. *Digital Journalism*, 1(1):27–47, Feb. 2013. ISSN 2167-0811. doi: 10.1080/21670811.2012.741316.

[24] J. R. Williams, J. P. Bagrow, C. M. Danforth, and P. S. Dodds. Text mixing shapes the anatomy of rank-frequency distributions. *Physical Review E*, 91(5):052811, May 2015. doi: 10.1103/PhysRevE.91.052811.

[25] A. Wyner and J. Ziv. Some asymptotic properties of the entropy of a stationary ergodic data source with applications to data compression. *IEEE Transactions on Information Theory*, 35(6):1250–1258, Nov./1989. ISSN 00189448. doi: 10.1109/18.45281.

[26] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, May 1977. ISSN 0018-9448. doi: 10.1109/TIT.1977.1055714.