SAND2017-7685C

Compression-Based Algorithms for Deception Detection

Christina L. Ting, Andrew N. Fisher, and Travis L. Bauer

Sandia National Laboratories, Albuquerque, NM 87123
{clting, anfishe, tlbauer}@sandia.gov

Abstract. In this work we extend compression-based algorithms for deception detection in text. In contrast to approaches that rely on theories for deception to identify feature sets, compression automatically identifies the most significant features. We consider two datasets that allow us to explore deception in opinion (content) and deception in identity (stylometry). Our first approach is to use unsupervised clustering based on a normalized compression distance (NCD) between documents. Our second approach is to use Prediction by Partial Matching (PPM) to train a classifier with conditional probabilities from labeled documents, followed by arithmetic coding (AC) to classify an unknown document based on which label gives the best compression. We find a significant dependence of the classifier on the *relative* volume of training data used to build the conditional probability distributions of the different labels. Methods are demonstrated to overcome the data size-dependence when analytics, not information transfer, is the goal. Our results indicate that deceptive text contains structure statistically distinct from truthful text, and that this structure can be automatically detected using compression-based algorithms.

1 Introduction

Deception, whether in content or style, is a common element across all forms of communication. However, human judgement performs roughly at chance at identifying deception [5]. Furthermore, with the massive amounts of textual information produced online, analysts need an automatic method for identifying features that are indicative of deception.

Previous work in deception detection has primarily relied on manual feature selection based on, for example, psycholinguistic theories of deception and/or computational linguistics, followed by supervised machine learning to build a classifier [1, 6, 17, 21, 22, 27]. In this work, we explore an alternative to explicit feature selection based on compression. In particular, we use compression to automatically identify the most significant structural and statistical elements common among deceptive documents in order to distinguish them from truthful documents. The approach can be generalized to other applications in which the goal is to identify similarities among common entities, *e.g.* authorship detection or user categorization.

In Section 2, we review related work on deception detection. In Section 3 we briefly discuss the two compression methods used to identify deceptive text; additional details can be found in the Appendix. In Section 4, we present results on two separate datasets for deception: a hotel dataset [20,21] containing truthful and deceptive hotel reviews (deception in opinion), and the *(Extended) Brennan-Greenstadt* [6] corpus on authentic, obfuscated, and imitated writing samples (deception in identity). We also discuss modifications to improve the performance of PPM/AC on unbalanced training data when the lossless information transfer component of compression does not need to be preserved. Finally, in Section 5, we conclude with the main findings of this work.

2 Related Work

2.1 Deception detection

As discussed in the Introduction, most work on deception detection in text has focused on identifying features that are indicative of deception. In this section, we provide a brief review of these studies, followed by an alternative, statistical-based approach to motivate our current work.

Pennebaker et al. [17] approached deception detection by developing a linguistic profile of deception based on the notion that *how* people express themselves may be more informative than *what* they express. The authors were able to apply their Linguistic Inquiry and Word Count (LIWC) text analysis program [23] to correctly identify deception at a rate of 67% when given a singular topic and 61% overall. Their results indicate liars show lower cognitive complexity, fewer self-references and other-references, and more negative emotion words. Building off Pennebaker's earlier work, Ott *et al.* [21] viewed the problem of deception detection in three ways: a standard text categorization task based on n-gram classifiers; an application of psychological effects of lying emerging in text based on LIWC; and an example of genre identification based on distributions of part-of-speech tags [4, 24], where deceptive and truthful writing fit into sub-genres of imaginative and informative writing, respectively. The authors found that the n-gram classifier outperforms both the psycholinguistically-motivated features and genre identification, but that a combined classifier is able to achieve nearly 90% accuracy on their dataset of truthful and deceptive hotel reviews.

In a similar approach to the above two groups, Brennan *et al.* [1,6] proposed to look for subtle changes in human behavior due to the additional cognitive effort required for deception. According to the authors, these subtle changes are manifested in stylistic differences between truthful and deceptive text. The authors explored three feature sets: a *Writeprints feature set* containing lexical, syntactic, and content specific features [25]; a *lying-detection feature set* containing features of lying in computer mediated communications and typed documents [8,13]; and a *9-feature set* containing features from the neural network experiments by Brennan and Greenstadt [7]. As with Pennebaker, the authors found that non-specific features (*e.g.*, function words) are as effective as content-specific features for identifying deception.

In addition to the dictionary bag-of-words and parts-of-speech features discussed above, Probabilistic Context Free Grammars (PCFGs) have been applied to extract deeper syntactic stylometry features with improved results [11]. Finally, a statistical language modeling (SLM) approach has also been used for deception detection [26]. In their work, Zhou *et al.* were motivated by a general technique that does not require explicit feature selection or extraction. Their SLM approach captures dependecies between words in n-grams and considers all words as potential features. Similarly, compression algorithms for text have been developed to automatically identify the most significant structural and statistical features in a document, and have found previous success in authorship identification and classification in gnereral [2, 9, 12, 15, 18].

3 Methods

Compression algorithms for text attempt to take a document and reduce the number of bits used to encode the document. The reason it is possible to reduce the number of required bits for a document is that similar structures in a document allow one to assign fewer bits for more common characters. For example, if one sees a q in a document, the likelihood that the next character is a u is so high that it is almost not necessary to include the u. Compression exploits these types of facts by using fewer bits to represent qu than to represent something like q-. Thus, if a document has a lot of similar structure, then usually a compression algorithm can compress it into fewer bits.

The basic assumption behind our use of compression algorithms for classifying truthful versus deceptive writings is that truthful documents should share a similar structure and deceptivedocuments should share a similar structure distinct from truthful documents. This structure can then be automatically detected by using a compression algorithm resulting in a compression to a fewer number of bits. The two methods that we use exploit this feature of compression in two different ways. With NCD [14], compression is used as a method of determining the similarity between two documents; with PPM/AC [10], compression is used to determine how similar a document is to a training set. Details of the two methods are provided in the Appendix; here we focus our discussion on modifications made to the standard implementation of PPM.

3.1 Modifications to PPM

As mentioned in the beginning of Section 3 and in the Appendix, we want to use PPM to create models for truthful and deceptive documents. When applying PPM for compression, the algorithm builds a set of tables that predicts the probability for the next character, given a finite set of previous characters. The finite set of previous characters is called the context. The model for truthful and deceptive documents is then the final state of the PPM tables after processing every document in the training set. Once we have the PPM tables for the truthful and deceptive models, we can use PPM with AC to compress a test document and

classify the document according to which model compresses the test document best (i.e. with the fewest number of bits). For this compression, we use a modified, *static* version of the PPM algorithm. In particular, we use the PPM tables after building the models, and do not update the PPM tables while compressing the test document. We avoid the updating step to avoid biasing the models with information from the test document.

The modifications described above are enough to adapt the PPM algorithm to a classifier scheme. However, we also explore the inclusion or exclusion of two non-character symbols: the *eof* and the *esc*. In the implementation of the PPM algorithm that we use, both the *eof* and the *esc* are given a static count of 1 in *every* PPM table, while the characters are given a count equal to the number of times that they have been seen for a given context. The prediction for a character in a table is then given by the ratio of its count to the total counts in the table; see Appendix for details. So, if the total counts in a given table are large, then the additional count of 1 for the *eof* and 1 for the *esc* is negligible. However, if the total counts in a given table are small, then including *eof* and *esc* significantly changes the prediction for all the characters in that table.

In addition to modifying the prediction of the characters, the rational for excluding the *eof* and the *esc* is further motivated by the following. The purpose of the *eof* is to mark the end of the file; unless some particular regular structure is expected at the end of the document, the *eof* is not likely to provide much information. The purpose for the *esc* symbol in the usual implementation of the PPM compression scheme is to signal to the decompression algorithm that the current table does not provide a prediction for the next character, and that the next table will be considered; see Appendix for details. Since we are using compression only as a means for comparing similarity, we do not need to decompress the documents. Hence, we do not need to include symbols, such as the *esc*, that are necessary only for decompression. In what follows, we explore the effects of including or excluding the *eof* and the *esc*.

4 Results

4.1 Data

We present results on two datasets. The first dataset is the *Brennan-Greenstadt* (*BG*) corpus of truthful and deceptive writing samples from 12 authors. The truthful writing samples are pre-existing samples written for an academic or business purpose. Each author provided 7–10 samples, for a total of approximately 5000 words per author. The deceptive writing samples are written for the purpose of two types of adversarial attacks: obfuscation and imitation. For the obfuscated samples, authors are asked to write a 500-word article describing their neighborhood while hiding their identity. For the imitated samples, authors are asked to write a 500-word article describing a day in their life in the third person in the style of Cormac McCarthy's "The Road". An extended version of the *BG* corpus [6] was collected from Amazon Mechanical Turk (AMT), where

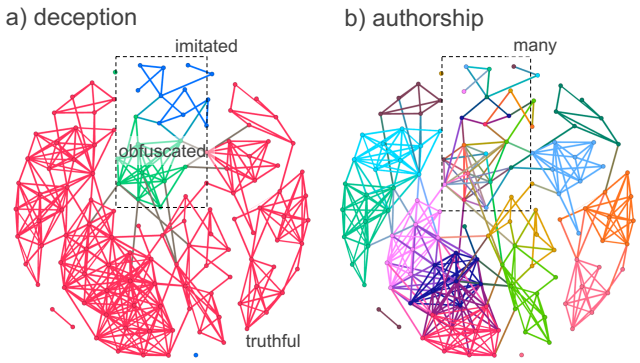

Fig. 1. Graphs showing the lowest 5% of the NCD scores for the *BG* dataset.

the participants were asked to adhere to a set of submission guidelines. Out of 100 submissions, 45 were accepted.

The second dataset consists of truthful and deceptive hotel reviews collected by Ott et al. in separate studies on positive [21] and negative [20] opinion spam. The authors mined various online travel review communities for truthful reviews. Although not a *gold standard* for truthful, recent studies suggest that deception rates in these online travel review communities is small [16, 19]. Deceptive reviews were then obtained through AMT, where each participant was given 30 minutes to complete the review and only one review per participant was allowed. 400 of each of the following categories were collected: truthful positive, truthful negative, deceptive positive, and deceptive negative.

4.2 NCD

We begin by computing the NCD between all pairs of documents within the *BG* corpus and visualizing the result as a graph, where the edge connecting documents d_i and d_j is assigned a weight corresponding to $NCD(d_i, d_j)$. To identify clusters with higher intracluster similarity, our approach is to remove edges above a maximum $NCD_{\max}$, where smaller values correspond to higher similarity. In Fig. 1, we visualize [3] the result of this approach using the Fruchterman-Reingold layout, where nodes correspond to documents and we keep only the lowest 5% of the NCD scores (*i.e.* the top 5% in terms of similarity). The two graphs are color-coded according to: a) deception and b) authorship. From the deception graph, it is apparent that the NCD score is able to correctly distinguish truthful documents (pink) from deceptive documents, where the latter can be further separated into "imitation" (blue) and "obfuscation" (green) categories. Within the truthful region of the deception graph, there appears to be several different clusters, which can be attributed from the authorship graph as belonging to different authors. In contrast, the two deception clusters, which are highlighted by the dashed box, can be seen to contain several different color nodes, indicating documents from several different authors. This result suggests

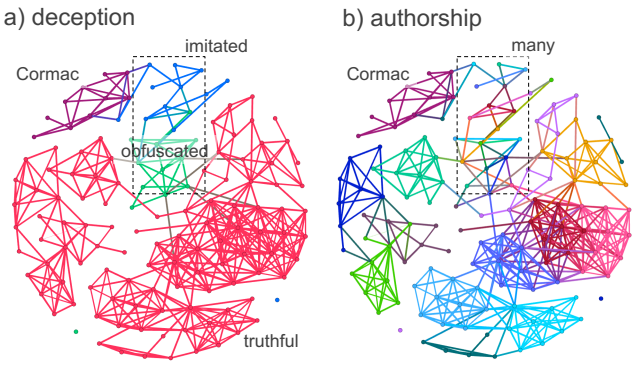

Fig. 2. Graphs showing the lowest 5% NCD scores for the *modified BG* dataset, where we have removed the word "neighborhood" from obfuscated text, and added authentic writing samples from Cormac McCarthy.

that deceptive documents contain features distinct from truthful documents and automatically discernible without global state.

Recall that deceptive documents in the *BG* corpus are collected by asking the authors to provide text that imitates Cormac McCarthy, and text that describes a neighborhood while obscuring the author's identity. One could then propose that the compression algorithm is picking up on these two features, rather than features of deception. To test this hypothesis, we have modified the BG corpus in two ways. First, to test whether NCD identifies Cormac McCarthy rather than imitation, we added writing samples of Cormac McCarthy taken from "The Road", "Blood Meridian", and "All the Pretty Horses". Second, to test whether NCD identifies "neighborhood" rather than obfuscation, we removed this word from the corpus. In Fig. 2 we show results for our *modified BG* corpus. Even without the "neighborhood" feature, the NCD algorithm is still able to correctly identify obfuscated (green) from truthful (pink) documents. In addition, there is a new cluster corresponding to the authentic Cormac McCarthy (purple) that is separate from imitated Cormac McCarthy (blue). These results suggests that, while it is possible to hide one's identity (*i.e.* separate from your own authorship cluster), it is less easy to imitate another's identity (*i.e.* participate in an authorship cluster not your own). Furthermore, we have identified two ways to "obscure" identity: intentional obfuscation and attempting to imitate another identity.

We also applied the NCD algorithm to the corpus of truthful and deceptive hotel reviews. In this case, we are unable to identify an $NCD_{\max}$ that separates truthful from deceptive documents. We hypothesize that the failure to identify deception in the hotel corpus as opposed to the *BG* corpus may be due to the difference in "deception" defined in the two datasets. For the hotel reviews, "deception" is in terms of lying about content; for the *BG* writing samples, "deception" is in terms of lying about identity. The results here suggest that lying about content is more difficult to identify using the NCD compression

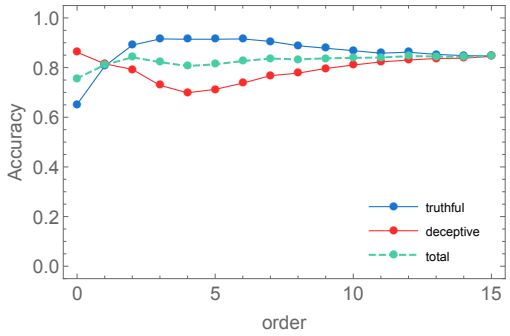

Fig. 3. PPMA/AC on hotel dataset: accuracy as a function of context order for classifying truthful (blue), deceptive (red), and total (green) documents.

algorithm. We have also tested whether the NCD algorithm is identifying hotels rather than deception, and were unable to obtain clusters of reviews written about the same hotel.

4.3 PPMA/AC

Here we present results using PPMA, followed by AC to build a classifier from a background collection of labeled documents. The category that compresses the unknown document with fewer bits is selected as the category. For categories, we consider "truthful" (T) and "deceptive" (D).

We begin with the hotel dataset, for which the NCD algorithm was unable to identify truthful and deceptive clusters. Fig. 3 shows accuracy as a function of context order d, averaged across a 4-fold cross validation scheme when a test document is compressed to a set of static PPMA tables. When looking at the entire dataset, it can be seen that the accuracy quickly reaches a maximum of 0.85 at order 2, without significant gains at higher orders. We note that an optimal accuracy at order 2 is somewhat unexpected since the best compression for the English language (and verified in this dataset) is usually around order 4 [10]. However, this result suggests that, in contrast to NCD, PPMA/AC successfully distinguishes deceptive from truthful hotel reviews. Furthermore, Table 4 in the Appendix shows that PPMA/AC outperforms standard machine learning with stylometry features on the same dataset by at least 20%. Similar stylometry features have been applied for deception detection, where it has been proposed that deceptive text contains simpler words and shorter sentences than truthful text [8].

Although we could already be satisfied with the high accuracies obtained in our classification of truthful and deceptive hotel reviews using PPMA/AC, it is still useful to understand why the algorithm achieves such high accuracy so that we can understand how to modify the algorithm when this is not the case. Therefore, we take a closer look at the statistics of the PPMA/AC algorithm on the hotel dataset. In Fig. 4, we show expected values for different statistics

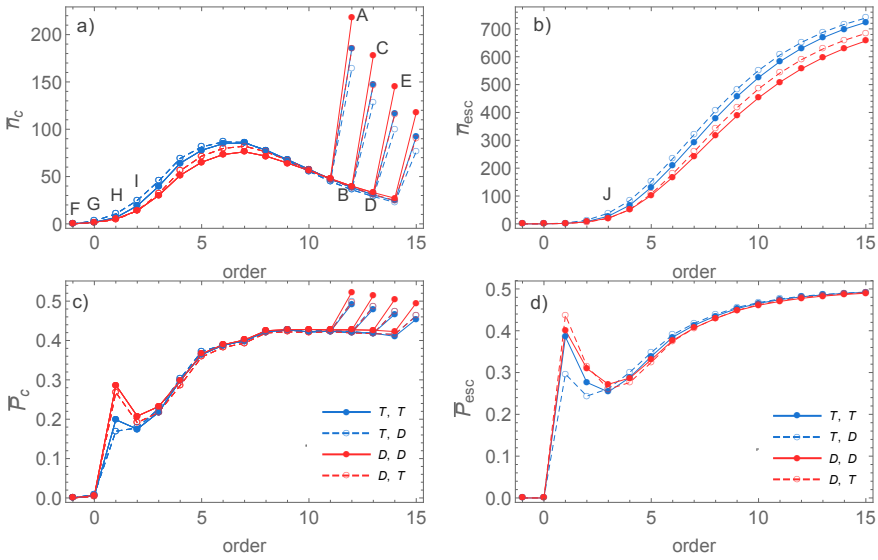

Fig. 4. PPMA/AC on hotel dataset: expected values for the counts $\bar{n}$ and probabilities $\bar{P}$ as a function of context order for max orders $d_{\max} \geq 12$. Panels a) and c) are for observed characters; b) and d) are for escapes. Figure label indicates test and model category, respectively.

as a function of order for different max orders. Note that *max order* here refers to the same *order* used in Fig. 3. We need to distinguish max order here because each time the max order context has not been observed by the model, the PPMA algorithm emits an escape and drops to lower order contexts; see the Appendix for details. Thus, for a given $d_{\max}$ there will be relevant statistics for all context orders $d \leq d_{\max}$. Note that each curve corresponds to a different test-to-model pairing, where correct test-to-model pairings (T, T and D, D) are indicated by the solid curves and incorrect test-to-model pairings (T, D and D, T) are indicated by the dashed curves.

We begin with Fig. 4a, which shows the expected value for the observed character counts $\bar{n}_c(d; d_{\max})$ as a function of the context order and maximum allowed order for different max orders $d_{\max}$. For clarity, we show only $d_{\max} > 12$, but note that results can be generalized to all $d_{\max}$. In terms of the implications of $\bar{n}_c(d; d_{\max})$ to the accuracy of a classifier, it can be seen that for both truthful (blue) and deceptive (red) documents, the correct test-to-model pairings (solid) have higher expected values for higher orders (including $d_{\max}$) than the incorrect test-to-model pairings (dashed), where the latter also have higher expected values at the lower orders. In other words, a given test document is able to match higher order contexts of its correct model, which is intuitively what we would expect. Furthermore, Fig. 4c shows that the expected probabilities of the observed contexts $\bar{P}_c(d; d_{\max})$ decrease as the context order decreases (except

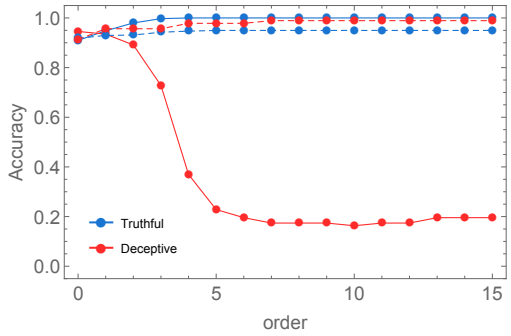

Fig. 5. PPMA/AC on *extended BG* dataset: accuracy as a function of context order for classifying truthful (blue) and deceptive (red) documents using PPMA for the *extended BG* dataset (solid) and for the *normalized BG* dataset (dashed).

for order 1). Since lower probabilities require more bits to encode, the incorrect model will compress the test document worse than the correct model. These are precisely the desired properties for a compression algorithm used as a classifier.

Interestingly, the average counts per order overlap for all max orders except for $\bar{n}_c(d_{\max}; d_{\max})$. In fact, $\bar{n}_c(d_{\max}; d_{\max})$ can be shown to be the sum of all higher order counts from larger $d_{\max}$. This is equivalent to saying that anything observed at contexts larger than the maximum that is requested by $d_{\max}$ gets lumped into $\bar{n}_c(d_{\max}; d_{\max})$. In the figure, this fact is demonstrated by the following combinations: $A = C + B$, $C = E + D$, and $A = E + B + D$.

In addition to accounting for contexts previously observed in the model, PPM is required to account for unobserved contexts by emitting escapes. In Fig. 4b we plot the expected value for the number of escapes $\bar{n}_{\text{esc}}(d)$ as a function of context order d. Note that the results for different $\bar{n}_{\text{esc}}(d)$ are independent of $d_{\max}$, so that we only show $d_{\max} = 15$. It can be seen that for both truthful (blue) and deceptive (red) test documents, the incorrect model (dashed) emits more escapes across all orders than the corresponding correct model (solid). Therefore, for all context orders the test document more often misses the tables of the incorrect model, emits an escape, and drops to a lower order table. This is shown directly in Fig. 4 by noting that the number of escapes at any given order is directly related to the sum of the number of characters emitted by all the lower order tables: $J = F + G + H + I$. Thus, the more escapes emitted at a given order, the more characters must also be emitted at lower order contexts. This has two effects. First, the cost of emitting an escape and its corresponding probability (see Fig. 4d) are accounted for. Second, as discussed above, the cost of dropping to lower orders is accounted for through lower probabilities, on average, of the lower order contexts (see Fig. 4c). These are the two primary reasons why the PPMA/AC algorithm is able to correctly classify deceptive and truthful documents in the hotel dataset with such high accuracy.

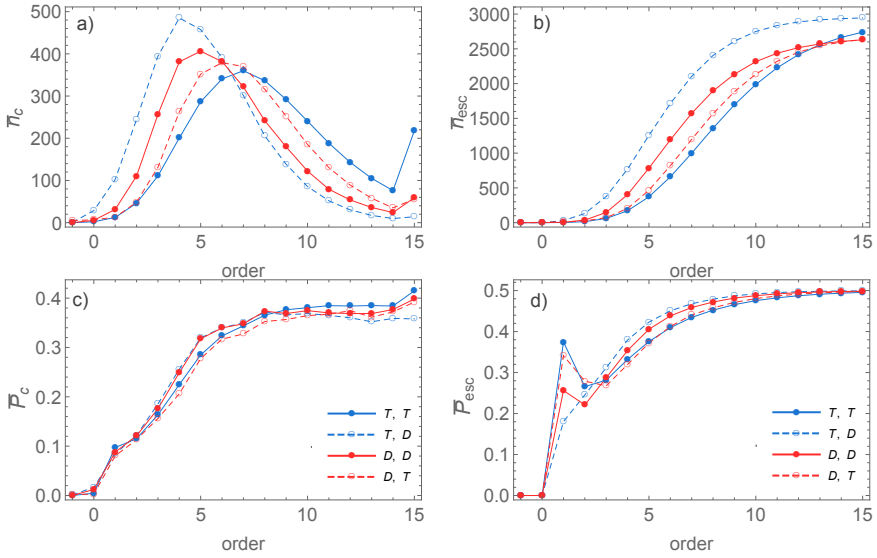

Fig. 6. PPMA/AC on *extended BG* dataset: expected values for the counts $\bar{n}$ and probabilities $\bar{P}$ as a function of context order for max order $d_{\max} = 15$. Panels a) and c) are for characters; b) and d) are for escapes. Figure label indicates test and model category, respectively.

Next, we apply the PPMA/AC algorithm to the *extended BG* corpus. Fig. 5 shows the accuracy of the truthful (blue solid) and deceptive (red solid) documents, where all deceptive (obfuscated and imitated) documents are grouped together. It can be seen that, while the truthful category reaches an accuracy near 1.0 by order 3, the deceptive category drops to 0.2 after order 5. We hypothesize that this result is due to an imbalance in training data. In particular, because the *extended BG* dataset contains nearly an order of magnitude more samples of truthful than deceptive documents, we obtain significantly more examples of truthful than deceptive contexts, which significantly alters the expected values of $\bar{n}_c(d; d_{\max})$ and $\bar{n}_{\text{esc}}(d)$ to favor the truthful model. To test this hypothesis, we investigate the output of the PPMA/AC algorithm applied to the *extended BG* corpus; see Fig. 6.

There are some notable differences between Fig. 4 and Fig. 6 that explain the low accuracy of the deceptive documents of the *extended BG* dataset, compared to the hotel dataset. First, recall that accurate classification by the PPMA/AC algorithm requires the expected value of the character count $\bar{n}_c(d; d_{\max})$ to peak at higher orders for the correct model (solid) than for the incorrect model (dashed). This is expected, since the test document should, on average, match longer strings of the correct model. Whereas the truthful tests (blue) show this pattern, the deceptive tests (red) do not; see Fig. 6a. Second, an accurate classifier requires the correct model (solid) to emit fewer escapes $\bar{n}_{\text{esc}}(d)$ than the

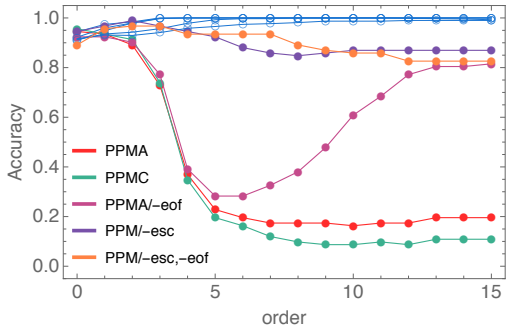

Fig. 7. Modified PPM on *extended BG* dataset: accuracy as a function of context order for classifying truthful and deceptive documents. Different colors correspond to different modifications to the PPM algorithm, where accuracy for all truthful documents are indicated by the empty circles.

incorrect model (dashed). The truthful test documents (blue) show this property across all context orders d, whereas the deceptive test documents (red) do not; see Fig. 6b. Taken together, the truthful model compresses *both* the truthful and deceptive test documents with fewer bits by matching higher order contexts and emitting fewer escapes. These results are consistent with our hypothesis that because there are so many more examples of observed contexts in the training data for the truthful model, a given test (truthful or deceptive) will match higher order contexts of the truthful model. Therefore, we obtain low accuracies for deceptive documents in Fig. 5. Based on these observations, two modifications to improve the accuracy are proposed: 1) a modification of the PPMA/AC algorithm to ignore the *esc* and 2) a normalization of the size of the training data, followed by a voting scheme to select the best model. We note that in reality, situations where there are far more examples of "regular" than "anomalous" training data is normal, and so it is important to modify the compression algorithms to handle these situations.

4.4 Modifications to the implementation of PPM

In Fig. 7 we show the accuracy of truthful and deceptive documents from the *extended BG* dataset, where different colors correspond to accuracies of the deceptive documents with different modifications to the PPMA/AC algorithm (see Section 3.1 for a discussion). However, we still show results for the accuracy of the truthful documents (empty markers) to demonstrate that none of the modifications we have considered reduce the accuracy of the truthful documents. The accuracy of identifying deceptive documents using PPMA is reproduced from Fig. 5 in red. It can be seen that ignoring the *esc* symbol (purple) increases the accuracy of the deceptive documents to nearly 100% for order 2, before dropping off again for higher orders. Interestingly, ignoring the *eof* symbol (magenta) eventually achieves an accuracy comparable to ignoring the *esc* symbol, but re-

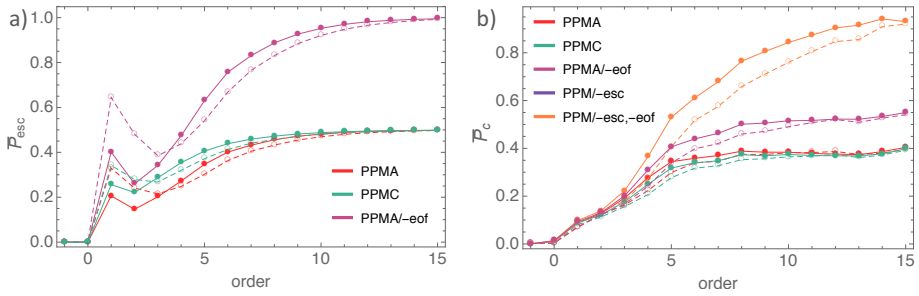

Fig. 8. Modified PPM on *extended BG* dataset: expected values for a) the escape probabilities and b) the character probabilities, for max order $d_{\max} = 15$. Different colors correspond to different PPM algorithms; D, D and D, T are given by solid and dashed lines, respectively.

quires higher context orders: $d_{\max} > 12$. The combination of ignoring both the *esc* and the *eof* symbols significantly improves the accuracy across all contexts in a similar manner as ignoring only the *esc* symbol. In contrast to ignoring the *esc* and/or the *eof*, PPMC (green) decreases the accuracy. To understand how these modifications affect the accuracy, in Fig. 8a) we take a closer look at the expected values for the escape probabilities $\bar{P}_{\text{esc}}(d)$. Here, the curves are distinguished by color according to the modification made to the PPMA/AC algorithm in Fig. 7 and we focus on deceptive test documents, where solid corresponds to D, D and dashed corresponds to D, T. The trivial result of ignoring the *esc* symbol is not shown.

PPMC (green) slightly increases $\bar{P}_{\text{esc}}(d)$, relative to PPMA, particularly at the lower context orders. This result can be expected by recalling that the PPMC algorithm assigns the escape a count equivalent to the number of distinct entries in a given table at a given context; see Appendix. PPMA, in contrast, assigns all escapes a count of 1. Both PPMA and PPMC converge to $\bar{P}_{\text{esc}}(d = 15) = 0.5$, since these higher order contexts have never been seen before and contain only the *eof* and the *esc* symbols, each with probability 1/2. In contrast, PPMA without the *eof* symbol (magenta) significantly increases $\bar{P}_{\text{esc}}(d)$ across all orders, before eventually converging to $\bar{P}_{\text{esc}}(d = 15) = 1.0$. In other words, by ignoring the *eof*, the higher order tables now contain only a single symbol: the *esc*. Because a probability of 1.0 requires zero bits to encode, the *esc* symbol is essentially ignored for the higher order contexts, which are where the vast majority of the escapes are actually issued; recall Fig. 6b. Therefore, ignoring the *eof* symbol has a very similar effect as ignoring the higher order escapes.

In cases where the expected value of escapes $\bar{n}_{\text{esc}}(d)$ is primarily determined by differences in training data size and are therefore meaningless, these results suggest that ignoring the *esc* symbol altogether is a successful method for recovering accuracy of the classifier. Instead, it is possible to use the structural information held in the context probabilities $\bar{P}_c(d; d_{\max})$. In particular, Fig. 8b) shows that for all versions of PPM/AC, the probabilities $\bar{P}_c(d; d_{\max})$ of the de-

ceptive test to the deceptive model (solid) are higher than the deceptive test to the truthful model (dashed). This is sufficient for better compression by the deceptive model once the escapes are removed. Note, however, that once the escapes are ignored, we no longer have an algorithm that can decompress. For analytic purposes, this is not our concern, and so we have the ability to modify our algorithm to optimize the output of compression for optimal classification.

4.5 Normalizing the training data

Finally, we consider a different approach to improving the low accuracies observed for the deceptive documents in the *extended BG* dataset. In particular, we propose to treat the discrepancy in the size of truthful versus deceptive training data head on: by dividing the truthful training data into subsets containing roughly the same number of documents as the deceptive training data. We then apply PPMA/AC to each subset of the truthful model, together with the full deceptive model, and assign a category using a voting scheme across all subsets, *i.e.* the final resulting category is the category that is selected most often while comparing a test document to a model of a subset of truthful training documents with a model of the full deceptive training documents. Results from this approach are shown in Fig. 5, where it can be seen that our normalization voting scheme (dashed) recovers high accuracy of the deceptive documents without significantly sacrificing the accuracy of the truthful documents.

Once again, we can compute the statistics of the counts and probabilities of characters and escapes, in order to better understand the accuracy of the algorithm. In Fig. A1 it can be seen that our normalization scheme produces statistics with similar trends to the hotel dataset. This result is not surprising since the hotel dataset was "normalized" to begin with. In particular, $\bar{n}_c(d; d_{\max})$ for the correct model (solid) now peaks at higher orders than the incorrect model (dashed) for both the truthful (blue) and the deceptive (red) test documents; and the incorrect model (dashed) emits more escapes on average $\bar{P}_{\text{esc}}(d)$ across all orders than the correct model (solid) for both truthful (blue) and deceptive (red) test documents. Together these results suggest that indeed the low accuracy of classifying the deceptive documents is due to the discrepancy in the training data size, and that the normalization scheme discussed here is able to produce a classifier that recovers high accuracies for all categories.

5 Conclusion

In this work we extend compression-based algorithms for deception detection in text. The basic motivation for this work is that deceptive text contains structure statistically distinct from truthful text, and that this structure can be automatically detected using compression-based algorithms. We consider two approaches: one based on normalized compression distance (NCD) and one based on using Prediction by Partial Matching (PPM).

For our first approach, we use an unsupervised clustering based on the NCD between pairs of documents whereby we identify clusters by selecting a threshold for allowed distances. We found that NCD does well at identifying identity deception, where individuals are attempting to imitate another's style or obfuscate their own identity. Moreover, we found that these types of deceptions themselves were distinguishable. We also found that this technique was not as successful for detecting deception in terms of false opinions.

For our second approach, we use Prediction by Partial Matching (PPM) to train a classifier with conditional probabilities from labeled documents, followed by arithmetic coding (AC) to classify an unknown document based on which class gives the best compression. We found that this approach is able to fill the gap left by our NCD approach in that this approach was able to discern false opinions from truthful ones. However, in attempting to identify deception in identity, we found a significant dependence of the classifier on the *relative* volume of training data used to build the conditional probability distributions of the different classes. We show that by modifying the PPM/AC algorithm we are able to overcome the data size-dependence when compression is used as a means of measuring similarity. The most successful modifications are the removal of the *esc* and the introduction of a voting scheme that attempts to balance the training data for both categories of truthful and deceptive.

Acknowledgements

The authors acknowledge funding support from the U.S. Department of Energy. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

References

1. Afroz, S., Brennan, M., Greenstadt, R.: Detecting hoaxes, frauds, and deception in writing style online. In: Proceedings of the 2012 IEEE Symposium on Security and Privacy. pp. 461–475 (2012)
2. Amitay, E., Yogev, S., Yom-Tov, E.: Serial sharers: Detecting split identities of web authors. In: ACM SIGIR 2007 Amsterdam. Workshop on Plagiarism Analysis, Authorship Identification, and Near-Duplicate Detection (2007)
3. Bastian, M., Heymann, S., Jacomy, M.: Gephi: An open source software for exploring and manipulating networks (2009), http://www.aaai.org/ocs/index.php/ICWSM/09/paper/view/154
4. Biber, D., Johansson, S., Leech, G., Conrad, S., Finegan, E., Quirk, R.: Longman grammar of spoken and written English 2 (1999)
5. Bond, C.F., DePaulo, B.M.: Accuracy of deception judgments. Personality and Social Psychology Review 10, 214–234 (2006)

6. Brennan, M., Afroz, S., Greenstadt, R.: Adversarial stylometry: circumventing authorship recognition to preserve privacy and anonymity. ACM Transactions on Information and System Security 15, 12:1–12:22 (2012)
7. Brennan, M., Greenstadt, R.: Practical attacks against authorship recognition techniques. In: Proceedings of the Twenty-First Conference on Innovative Applications of Artificial Intelligence (IAAI), Pasadena, CA (2009)
8. Burgoon, J., Blair, J., Qin, T., Nunamaker Jr, J.F.: Detecting deception through linguistic analysis. In: Intelligence and Security Informatics. pp. 91–101 (2010)
9. Cilibrasi, R., Vitányi, P.M.B., de Wolf, R.: Algorithmic clustering of music based on string compression. Computer Music Journal 28, 49–67 (2004)
10. Cleary, J.G., Whitten, I.H.: Data compression using adaptive coding and partial string matching. IEEE Transactions on Communications 32, 396–402 (1984)
11. Feng, S., Banerjee, R., Choi, Y.: Syntactic stylometry for deception detection. In: Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics. pp. 171–175 (2012)
12. Frank, E., Chui, C., Whitten, I.H.: Text categorization using compression models. In: Data Compression Conference, 2000. Proceedings. DCC 2000 (2000)
13. Hancock, J.T., Curry, L.E., Goorha, S., Woodworth, M.: On lying and being lied to: a linguistic analysis of deception in computer-mediated communiation. Discourse Processes 57, 1–23 (2006)
14. Li, M., Chen, X., Li, X., Ma, B., Vitányi, P.M.B.: The similarity metric. IEEE Transactions on Information Theory 50, 3250–3264 (2004)
15. Marton, Y., Wu, N., Hellerstein, L.: On compression-based text classification. In: ECIR'05 Proceedings of the 27th European Conference on Advances in Information Retrieval Research. pp. 300–314 (2005)
16. Mayzin, D., Dover, Y., Chevalier, J.A.: Promotional reviews: an empirical investigation of online review manipulation (2012)
17. Newman, M.L., Pennebaker, J.W., Berry, D.S., Richards, J.M.: Lying words: predicting deception from linguistic styles. Personality and Social Psychology Bulletin 29, 665–675 (2003)
18. Nishida, K., Banno, R., Fujimura, K., Hoshide, T.: Tweet classification by data compression. In: Proceedings of the 2011 International Workshop on Detecting and Exploiting Cultural Diversity on the Social Web. pp. 29–34 (2011)
19. Ott, M., Cardie, C., Hancock, J.T.: Estimating the prevalence of deception in online review communities. In: Proceedings of the 21st International Conference on the World Wide Web. pp. 201–210 (2012)
20. Ott, M., Cardie, C., Hancock, J.T.: Negative deception opinion spam. In: Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. pp. 497–501 (2013)
21. Ott, M., Choi, Y., Cardie, C., Hancock, J.T.: Finding deceptive opinion spam by any stretch of the imagination. In: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies. pp. 309–319 (2011)
22. Pennebaker, J.W., Chung, C.K., Ireland, M., Gonzales, A., Booth, R.J.: The development and psychometric properties of LIWC2007. In: Austin, TX, LIWC. Net. (2007)
23. Pennebaker, J.W., Frances, M.E., Booth, R.J.: Linguistic Inquiry and Word Count: LIWC 2001. Lawrence Erlbaum, Mahwah, NJ (2001)
24. Rayson, P., Wilson, A., Leech, G.: Grammatical word class variation within the British National Corpus sampler. Language and Computers 36, 295–306 (2001)

25. Zheng, R., Li, J., Chen, H., Huang, Z.: A framework of authorship identification for online messages: writing style features and classification techniques. Journal of the American Society for Information Science and Technology 57, 378–393 (2006)
26. Zhou, L., Shi, Y., Zhang, D.: A statistical language modeling approach to online deception detection. IEEE Transactions on Knowledge and Data Engineering 20, 1077–1081 (2008)
27. Zhou, L., Twitchell, D.P., Qin, T., Burgoon, J.K., Nunamaker, J.F.: An exploratory study into deception detection in text-based computer mediated communication. In: Proceedings of the 36th Hawaii International Conference on System Sciences (2003)

Appendix for: Compression-Based Algorithms for Deception Detection

Christina L. Ting, Andrew N. Fisher, and Travis L. Bauer

Sandia National Laboratories, Albuquerque, NM 87123
{clting, anfishe, tlbauer}@sandia.gov

A1 Normalized compression distance

Our first method of distinguishing between truthful and deceptive documents is to use a type of similarity measure called the *normalized compression distance* (NCD) [1]. NCD attempts to determine how similar two strings are to each other while also taking their size into account. Rather than providing an absolute distance, NCD is normalized in the sense that a pair of small strings should not be considered closer to one another than a pair of large strings simply because they are smaller. NCD is defined by:

$$\mathrm{NCD}(x,y) = \frac{C(xy) - \min\{C(x), C(y)\}}{\max\{C(x), C(y)\}},$$

where $C(x)$ is a compression algorithm applied to a string x and xy denotes the concatenation of two strings x and y. We use LZMA as the compression algorithm.

To use the NCD in a clustering scheme, we compute the pairwise NCD between every pair of documents. We then use a threshold on the size of the distance in order to identify clusters.

A2 Prediction by partial matching and arithmetic coding

PPM using AC is a powerful method for compression and was used as the benchmark for compression algorithms for a number of years. The two algorithms complement each other in that PPM provides a model for predicting the next character in a document while AC takes a prediction model and uses the provided probabilities for each character to produce an encoded binary output. When the probability for a particular character or symbol is provided to AC we say that it is *emitted*. This terminology is helpful since PPM does not always provide the probability for the next character, but instead emits a sequence of other symbols called *escapes* to encode the fact that the PPM model is changing its internal state. These escapes, in turn, are used by a decoder so that the decoder can make the same changes.

The complementary nature of PPM and AC allows us to use them as part of a supervised learning scheme. We can use PPM to create models for truthful

documents, called the *truthful model*, and for deceptive documents, called the *deceptive model*. We can then classify a document by using each predictive model together with AC and assigning the document to the class whose model produces the fewest number of bits for the compressed output.

For this scheme to work, it is possible to use a standard implementation of AC and the details are not necessary for understanding our method or results. There are only two facts that one needs to know. The first is that only non-zero probabilities are allowed and the second is that the higher the probability assigned to the next character the fewer the eventual bits needed for the compressed output. Saying the second fact another way, the better the predictor is at predicting the next character, the fewer the bits that are required.

Unlike AC, our way of using PPM is not the standard method. Moreover, in Section 4 we use two variants of PPM, known as PPMA and PPMC, and we make additional modifications to the PPM algorithm. To understand these changes, it is necessary to give a detailed overview of PPM, which we do next.

In general, PPM uses a set of tables of predictors of the next character that are conditioned on previous characters. The previous characters used are called the *context*, where the number of characters in the context, d, is called the *order*. Each table is representative of all the prediction of a given context within a given order, and is called an order-d table. Note that since there are multiple contexts that have the same order, there are several order-d tables for each order $d \geq 1$. For example, if one has the characters *abab* as the context, then the prediction for the next character to be an a given *abab* is contained in a table of order-4 with context *abab*. Thus, an order-d table for context $c_0c_1 \ldots c_{d-1}$ is really the collection of conditional probabilities $P(c|c_0c_1 \ldots c_{d-1})$ providing the probability of seeing a token c given that $c_0c_1 \ldots c_{d-1}$ are the preceding characters. The special case when the order is 0 is $P(c)$, which is just the probability of c occurring.

For the versions of PPM that we consider (PPMA and PPMC), the orders that are allowed are bounded by a user–specified maximum order $d_{\max}$. When predicting the next character c, PPM first considers the previous $d = d_{\max}$ characters $c_0c_1 \ldots c_{d-1}$ as the context, where c_{d-1} is the character directly preceding c. If $P(c|c_0c_1 \ldots c_{d-1})$ is non-zero, then this probability is provided to AC. However, it may be the case that $P(c|c_0c_1 \ldots c_{d-1}) = 0$, which is not an allowable probability for AC. When this event occurs, we say that the table does not contain c and PPM issues a probability for a special symbol called an *escape* to indicate that the character was not found. Then, PPM changes state and attempts to provide the probability of the next smaller context $P(c|c_1 \ldots c_{d-1})$. If $P(c|c_1 \ldots c_{d-1}) = 0$, then $P(c|c_2 \ldots c_{d-1})$ is considered, etc. If the character is not found with any context, then $P(c)$ is considered. Finally, if PPM has no prediction for $P(c)$, then a default uniform distribution is assumed for all characters. The table for this distribution is considered to have order -1. In addition to the characters, every table of order $d > -1$ has two additional symbols: an end-of-file symbol *eof* and an escape symbol *esc*. These symbols are present even if the probability for the characters may be zero.

PPM uses an adaptive strategy for generating the tables. Instead of keeping track of the probabilities for a given context directly, a character count is maintained for each character seen with the given context up to context order $d_{\max}$. For example, consider the string *rowrowrow* and a max context order of $d_{\max} = 2$. After receiving the first character r, the count r is incremented by one in the order-0 table. When o is received, the count for o is incremented by one in the order-1 table with context r, and the order-0 table. When w is received, the count of w is incremented by one in the order-2 table with context ro, in the order-1 table with context o, and in the order-0 table. When the next r is received, the count of r is incremented by one in the order-2 table with context ow, the order-1 table with context w, and the order-0 table. This process continues until all characters are received. The resulting character counts for this example are shown in Table A1. Note that this process is the exact process used for PPMA and the process we use for PPMC, though technically a pure implementation of PPMC updates the tables differently. In our results, it is the escapes that seem particularly influential on the number of output bits, so we chose to focus on the change that PPMC does for the escapes and not the change PPMC does for tables. More details on the pure implementation of PPMC are in [2].

Table A1. Resulting characters counts for the PPM tables after receiving the character string *rowrowrow*.

Order	Context	Character Count			Total
		r	o	w	
2	*ro*			3	3
2	*ow*	2			2
2	*wr*	2			2
1	*r*	3			3
1	*o*			3	3
1	*w*	2			2
0		3	3	3	9

As mentioned above, it is not only the characters that are contained in the table, but also two symbols: *eof* and *esc*. The end-of-file symbol, *eof*, is always allocated a count of one. The count associated with the *esc* is different depending on whether we are using PPMA or PPMC. In fact, the difference in escape counts is the only difference between PPMA and PPMC that we use in our implementation. For PPMA, the *esc* symbol is allocated a single count while for PPMC, the escape is given a count equal to the number of non-zero character entries in the table, or 1 if no non-zero character entries are present. Thus, for the order-0 table in Table A1 the corresponding count for the *esc* according to PPMC is 3 while for any of the order-1 tables, the count is 1. The symbol counts for both methods are shown in Table A2.

Table A2. Resulting symbol counts for PPMA and PPMC tables after receiving the character string *rowrowrow*.

Order	Context	Symbol Count		
		eof	esc_A	esc_C
2	*ro*	1	1	1
2	*ow*	1	1	1
2	*wr*	1	1	1
1	*r*	1	1	1
1	*o*	1	1	1
1	*w*	1	1	1
0		1	1	3

To get the probabilities that are sent to AC, PPM just normalizes the entries in the tables. For example, for PPMA $P(r|ow) = \frac{2}{4}$ since r has a character count of 2, *eof* has a count of 1, and *esc* has a count of 1. So, the total count is 4, which is the normalization factor. The rest of the probabilities for PPMA and PPMC are shown in Table A3. Note that each row in this table is really a full PPM table for a given context of a given order-d, as defined in the beginning of this section.

References

1. Li, M., Chen, X., Li, X., Ma, B., Vitányi, P.M.B.: The similarity metric. IEEE Transactions on Information Theory 50, 3250–3264 (2004)
2. Moffat, A.: Implementing the PPM data compression scheme. IEEE Transactions on Communications 38, 1917–1921 (1990)

Table A3. Resulting probabilities for the characters and symbols provided by PPMA and PPMC.

PPMA

Order	Context	Probability				
		r	o	w	eof	esc_A
2	*ro*			$\frac{3}{5}$	$\frac{1}{5}$	$\frac{1}{5}$
2	*ow*	$\frac{2}{4}$			$\frac{1}{4}$	$\frac{1}{4}$
2	*wr*		$\frac{2}{4}$		$\frac{1}{4}$	$\frac{1}{4}$
1	*r*		$\frac{3}{5}$		$\frac{1}{5}$	$\frac{1}{5}$
1	*o*			$\frac{3}{5}$	$\frac{1}{5}$	$\frac{1}{5}$
1	*w*	$\frac{2}{4}$			$\frac{1}{4}$	$\frac{1}{5}$
0		$\frac{3}{11}$	$\frac{3}{11}$	$\frac{3}{11}$	$\frac{1}{11}$	$\frac{1}{11}$

PPMC

Order	Context	Probability				
		r	o	w	eof	esc_A
2	*ro*			$\frac{3}{5}$	$\frac{1}{5}$	$\frac{1}{5}$
2	*ow*	$\frac{2}{4}$			$\frac{1}{4}$	$\frac{1}{4}$
2	*wr*		$\frac{2}{4}$		$\frac{1}{4}$	$\frac{1}{4}$
1	*r*		$\frac{3}{5}$		$\frac{1}{5}$	$\frac{1}{5}$
1	*o*			$\frac{3}{5}$	$\frac{1}{5}$	$\frac{1}{5}$
1	*w*	$\frac{2}{4}$			$\frac{1}{4}$	$\frac{1}{5}$
0		$\frac{3}{13}$	$\frac{3}{13}$	$\frac{3}{13}$	$\frac{1}{13}$	$\frac{3}{13}$

Table A4. Machine learning on the Hotel corpus.

Machine Learning Algorithm	Accuracy
Random Forest	0.59
Gradient Boosted Forest	0.61
Linear Support Vector Machine	0.63
Support Vector Machine	0.65

Stylometry features
Number of line endings
Number of punctuations
Number of multiple punctuations
Longest multiple punctuation
Number of capital letters
Number of words
Average word length
Average sentence length
Average paragraph length

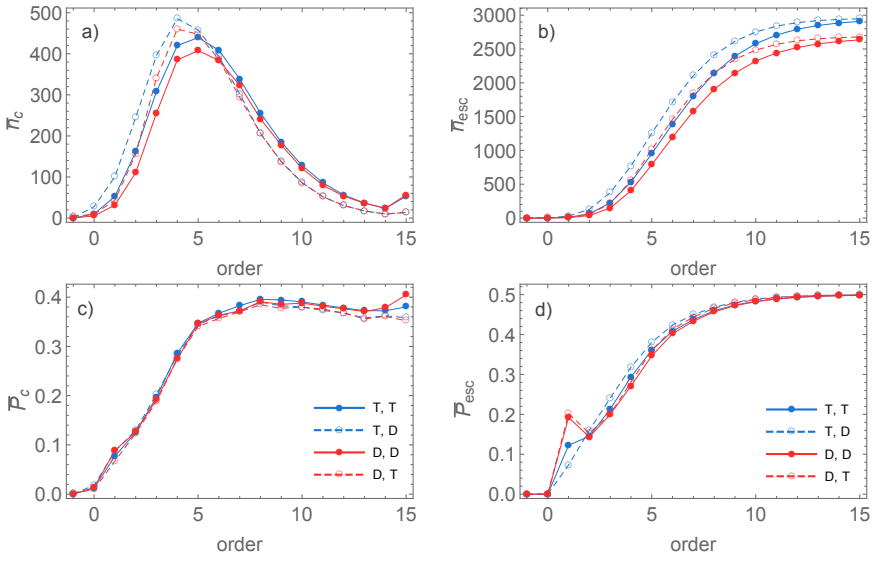

Fig. A1. Normalizing the *extended BG* dataset: expected values for the counts $\bar{n}$ and probabilities $\bar{P}$ as a function of context order for max order $d_{\max} = 15$. Panels a) and c) are for characters; b) and d) are for escapes. Figure label indicates test and model category, respectively.