LEARNING WITH N-GRAMS: FROM MASSIVE SCALES TO COMPRESSED REPRESENTATIONS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Hristo Spassimirov Paskov
March 2017

Abstract

Machine learning has established itself as an important driver of industrial progress and scientific discovery. The quest to expand its usage to address ever deeper questions and harder problems places particular emphasis on building sophisticated and statistically rigorous models that can handle the deluge of information being generated. The stakes are higher than ever; the success of global, billion dollar initiatives that can fundamentally change the landscape of human health rests on the existence of machine learning tools that can extract intricate relationships at unprecedented scales. In turn, machine learning paradigms are constantly evolving to address these needs, and some of the greatest advances have come from integrating combinatorial ideas with classical statistical ideas, such as the ability to perform principled feature selection using the Lasso. The underlying perspective of this thesis is that machine learning must rely on the algorithms and data structures that classically form the underpinnings of theoretical computer science in order to fully harness the potential of these combinatorial ideas.

To this end, we contribute two advances to machine learning based on N-gram features, a feature representation for strings that has stood the test of time and continues to provide state-of-the-art results in natural language processing and genomics. The first addresses the computational and statistical issues of learning with long, and possibly all, N-grams in a document corpus. Our main result leverages suffix trees to provide a *quadratic* memory and processing time improvement over current machine learning systems by virtue of a fast matrix-vector multiplication routine whose computational requirements are at worst *linear* in the length of the underlying document corpus. As the majority of machine learning algorithms rely on and are bottlenecked by matrix-vector multiplication to learn, our routine can speed up almost any learning system by simply replacing its multiplication routine with ours. The practical savings are substantial, including an efficiency gain of four orders of magnitude for DNA sequence data, and open a new realm of possibilities

for N-gram models. This routine also has large statistical implications; suffix trees perform a *quadratic* dimensionality reduction that substantially increases the robustness of machine learning systems when the appropriate level of data representation granularity is unknown. Finally, we provide an efficient persistent data storage system based on our algorithms that screens N-gram features according to a multitude of statistical criteria and produces data structures optimized for multiplication.

Our second contribution looks to classical ideas from compression to devise a new form of combinatorial Deep Learning for text termed Dracula. Dracula is based on a generalization of the compression criterion underlying dictionary–based compressors like Lempel-Ziv 78. It learns a dictionary of N-grams that efficiently compresses a text corpus, and then recursively compresses its own dictionary for additional space savings. In doing so, it selects N-grams that are useful features for learning and induces a graph–based regularizer that orders the N-grams into low and high frequency components. Importantly, solving Dracula can be expressed as a binary linear program that may be further relaxed to a linear program, allowing a plurality of tools from optimization and computer science to be used to analyze its properties. Computationally, Dracula is NP-Complete, but it exhibits substantial problem structure that allows approximate algorithms to scale to large datasets. Statistically, we show how Dracula can learn a multitude of representations to accommodate an underlying storage cost model and identify parameters that control the behavior of its solutions in meaningful ways. We also demonstrate that Dracula is amenable to fine tuning by proving that its solutions evolve in a predictable way as the storage cost model varies. We demonstrate the utility of Dracula's features using experiments over a variety of problem domains including natural language processing and bioinformatics.

Dedicated to my family.

Acknowledgments

Mathematical insight cannot happen in a vacuum; it is intimately tied to our conversations, emotions, and experiences. I am deeply lucky to have been surrounded by friends, family, fellow students, and professors who have been an endless source of such inspiration.

I am profoundly grateful to my advisers, John Mitchell and Trevor Hastie, for allowing me to explore a myriad of crazy directions and ideas and for offering guidance whenever I needed it. You made this PhD surpass my highest expectations; your kindness and wisdom have left a lasting impression.

I would also like to thank the professors whom I have worked with, learned from, and chatted with on numerous occasions in the hallways of Gates: Robert Tibshirani, Jonathan Taylor, Dan Boneh, Ryan Williams, Virginia Williams, Vaughan Pratt, Christopher Ré, Jure Leskovec, Hector Garcia Molina, and Stephen Boyd.

Then there is the Stanford crew that can only be described as the perfect mix of distraction, chaos, and wisdom. Bob, your prediction from Stanford's visit day over six years ago that we would be great friends if we went to Stanford could not be truer; it's been an amazing journey (although we still need to finish that book on beer opening techniques). Julian, you have been an endless source of amazing times, professorial wisdom, coffee breaks, and terrible lifting advices - my biceps are still bigger than yours. Krevl, besides being "one of the top human beings" I have ever had the privilege to call a dear friend, you are also the CS department's secret weapon; **none** of my papers would have been possible without your help. Ashton, thank you for making the Mission a second home; "liming" in SF kept the sanity going and made for some of my happiest memories. Jake, I am sorry for tossing you out of my apartment after Stanford's 2012 loss to Notre Dame, you turned out to be a great friend despite rooting for the wrong football team.

To the crew in New York, whom I have now known and partaken in countless adventures with for over two decades, your friendship and support has meant the world to me. During my oral examination for this

thesis I not only had the support of those physically present in the room, but also my dear friends in New York who tuned in online to watch and called to wish me luck: Yue, Alex, Kiril, Kenny, Salil, and Saureen. Yue, despite living three thousand miles away you have managed to support me through every major event and wish me luck on every presentation, test, and milestone. I will never forget the numerous times you surprised me on my birthday whenever I was back in New York around February; you are the best of friends one could ask for.

This work would have been impossible without the love and support of my family; you helped me get through the hardest of times and made the successes worth celebrating. Mom and Dad, thank you for instilling into me a never-ending mathematical inquisitiveness and bravery that has always led me to aspire to greater challenges. Ivan, it's been amazing having you out here in Stanford and now it's your turn to do a PhD (prediction: your dissertation will be five times longer than mine). Alex, you are just starting your academic journey and we are so excited to see where you end up. Keep that (friendly) brotherly competition going; you are both academic forces to be reckoned with, and I am so lucky to have such amazing brothers that I can collaborate with on life and research! Finally, to my family in Bulgaria: Hristo and Stamatka, Ivan and Kalinka, and Atanas and Minka, thank you for all of the amazing visits; though thousands of miles separate us, your warmth and love is ever-present with us.

Thank you to my family on the West Coast: Michelle, Michael, Jake, and Sara. The books you've bought me have each been the seeds of numerous ideas, and your warmth and support has helped those ideas blossom into publications. You've also successfully turned me into an undying Stanford football fan (perhaps even fanatic); I look forward to every fall with the utmost excitement and have really enjoyed sharing that with you!

And last but not least, I would like to thank my partner in crime, Kelley Paskov. Your love and support made this PhD possible and were my largest source of inspiration. And now an even bigger project named Delilah awaits us; I cannot wait for her to get here!

Contents

Abstract iv

Acknowledgments vii

1 Introduction and Background 1

1.1 Introduction . . . 1

1.1.1 Thesis Contributions and Organization . . . 6

1.1.2 Related Work . . . 9

1.2 Background . . . 11

1.2.1 Suffix Trees and Arrays . . . 12

2 Efficient Learning with Bag of N-Gram Models 14

2.1 Tree Traversals and Storage . . . 14

2.2 Matrix Multiplication and Learning . . . 16

2.3 Fast Multiplication . . . 17

2.3.1 Linear Dependencies in the Node Matrix . . . 17

2.3.2 Fast Multiplication Algorithm . . . 19

2.3.3 Matrix Data Structure . . . 21

2.4 Statistical Considerations . . . 22

2.4.1 Problem Reformulation . . . 23

2.4.2 Holding Out Data . . . 25

2.4.3 Normalization . . . 26

2.5 Persistent Storage Framework . . . 26

2.5.1 Computing the Persistent Storage Representation . . . 27

2.5.2 Feature Filtering and Normalizations . . . 27

2.5.3 Emitting Multiplication Optimized Matrices . . . 28

2.6 Experiments . . . 29

2.6.1 Memory and Timing Benchmarks . . . 29

2.6.2 Sentiment Analysis Tasks . . . 31

3 Data Representation and Compression Using Linear-Programming Approximations **35**
3.1 Compressive Feature Learning . . . 36
3.2 Adding Depth with DRaCULA . . . 38
3.3 Computational Properties . . . 40
3.3.1 Reconstruction Modules . . . 41
3.3.2 NP-Completeness . . . 43
3.3.3 Problem Relaxation and Polyhedral Refinement . . . 43
3.4 Learning with Compressed Features . . . 45
3.4.1 Dracula's Solution Path . . . 46
3.4.2 Dictionary Diffusion . . . 52
3.5 Solving Strategies for Shallow Compression . . . 54
3.5.1 Iterative Reweighting . . . 57
3.5.2 $\ell_1 \to \ell_\infty$ Homotopy . . . 59
3.5.3 ADMM Formulation . . . 60
3.5.4 Comparison with Iterative Reweighting . . . 62
3.6 Experiments . . . 64
3.6.1 Iterative Reweighting . . . 66
3.6.2 $\ell_1 \to \ell_\infty$ Homotopy . . . 70
3.6.3 Deep Compression . . . 73

4 Conclusion **77**

A Derivations for ADMM Subproblems **79**
A.1 ADMM Derivations for Iterative Reweighting . . . 79
A.1.1 Solving for w . . . 79
A.1.2 Solving for z . . . 83
A.1.3 Matrix Entries . . . 83
A.2 Linearized ADMM Derivations for $\ell_1 \to \ell_\infty$ Homotopy . . . 84
A.2.1 Additional Structure in X . . . 85

List of Tables

2.1 Mean Squared Error for Sentiment Analysis 31

3.1 Time Trials for Homotopic (H) and IR Algorithms 64
3.2 Classification accuracy on the 20 Newsgroups and IMDb datasets 67
3.3 Testing Accuracy on Author Identification Task 71
3.4 MSE when Predicting Rating or ABV 72
3.5 Bacteria Identification Accuracy using Protein Data 74
3.6 Author Identification Accuracy 75
3.7 Sentiment Classification Accuracy 76

List of Figures

2.1 Memory utilization for the BFMT, node, and all N-gram matrices as a function of maximum N-gram length K on the BeerAdvocate data (top), journal data (middle) and 1000 genomes data (bottom). 33
2.2 Average time to perform one matrix-vector multiply with the BFMT and node matrices as a function of maximum N-gram length K on the BeerAdvocate data (top) and 1000 Genomes Data (bottom). Node matrix times are missing for the latter because it was impractical to store. 34

3.1 Three different CFL solutions on a toy example. Dictionary cost: number of characters in dictionary. Pointer set cost: $\lambda \times$ number of pointers. **Left:** dictionary cost only ($\lambda = 0$). **Right:** expensive pointer cost ($\lambda = 8$). **Center:** balanced dictionary and pointer costs ($\lambda = 1$). 36
3.2 Fraction of pointers that correspond to unigrams, bigrams, and trigrams in the compressed representation of the BeerAdvocate dataset. We set the dictionary inclusion cost identically to one and use the same pointer cost for all pointers and allow up to trigrams. 37
3.3 Compression of "aabaabaax" using a 3-layered dictionary. Layer 0 consists of characters; layers 1 and 2 are dictionary n-grams. There are three kinds of pointers: character to dictionary N-gram (dashed blue lines), dictionary N-gram to (longer) dictionary N-gram (solid blue line), and dictionary N-gram to document (double red lines). 41
3.4 Part (a) shows a nonlinear projection of a subset of Dracula's constraint polyhedron $\mathcal{Q}$ in which every vertex corresponds to a distinct compression of "xaxabxabxacxac". Part (b) is the projection's polar; its faces delineate the (linear) costs for which each vertex in (a) is optimal. The red/ purple/ blue line in (b) demonstrates a continuous family of costs. All red (blue) costs are uniquely minimized by the vertex in (a) highlighted in red (blue), respectively; (c) shows the corresponding compressions. Purple costs lie on the edge between the faces containing the red and blue lines and are minimized by any convex combination of the vertices highlighted in (a). 48
3.5 Performance trial comparing our homotopic method to the IR scheme of [50]. The red line denotes a lower bound for the objective. Circles indicate the starting/stopping points of the algorithms and the "X" marks the point at which we increase ζ from 0. 65

3.6 Misclassification error and standard error bars when classifying alt.atheism (A) vs. comp.graphics (G) from 20 Newsgroups. The four leftmost results are on features from running LZ77 on documents ordered by class (AG, GA), randomly (Rand), or by alternating classes (Alt); the rightmost is on our compressed features. 67
3.7 PCA plots for 20 Newsgroups. **Left:** alt.atheism (blue), comp.graphics (red). **Right:** alt.atheism (blue), comp.sys.ibm.pc.hardware (green), rec.motorcycles (red), sci.space (cyan), talk.politics.mideast (magenta). **Top:** compressed features (our method). **Bottom:** all 5-grams. 68
3.8 Classification accuracy as the training set size varies for two classification tasks from 20 Newsgroups: **(a)** alt.atheism (A) vs. comp.graphics (G); **(b)** rec.sport.baseball (B) vs. rec.sport.hockey (H). To demonstrate the effects of feature selection, L_2 indicates L_2-regularization while EN indicates elastic-net regularization. 69
3.9 Testing and validation set accuracies of CFL features on author identification task as a function of the pointer cost. Unigrams correspond to $\lambda = 0$. 72
3.10 Proteins represented using the 4^{th} and 5^{th} singular vectors of Top features from Dracula. . . . 73

Chapter 1

Introduction and Background

1.1 Introduction

Machine learning is becoming a fundamental driver of global progress, opening a plethora of possibilities spanning everyday life conveniences, such as self-driving cars [44] and smart homes [16], to life-saving necessities, such as targeted therapies in cancer treatment [60] and human trafficking prevention [1], to fundamental and surprising discoveries in all branches of science including chemistry [13], astronomy [4], and biology [63]. Yet more than ever there is a need for fundamental innovation in this discipline. The global response to machine learning's success has been to collect ever larger amounts of data, and the onus is on machine learning to make sense of this data. For instance President Barack Obama's Precision Medicine initiative along with the Cancer Genome Atlas [46] are billion dollar, decade long projects that will sequence over one million genomes — petabytes of information — and combine them with health records in hopes of understanding cancer and human disease. The success of these massive undertakings is critically dependent upon the availability of machine learning paradigms that can detect intricate patterns at unprecedented data scales; in some cases lives depend on the existence of these algorithms.

The core philosophy of this thesis is that the classical algorithmic cornerstones of computer science are essential for the next generation of machine learning paradigms that can handle the computational demands and statistical sophistication required by modern massive datasets. Compounding these considerations is that the successful adoption of any particular paradigm also relies heavily on its ease of use and the development

times necessary to construct a successful machine learning system. In many respects learning paradigms are akin to programming languages. They each prescribe specific methodologies for building systems, make careful trade-offs in view of their perspectives, and are constantly evolving to better satisfy demands. The most recent advances have resulted in machine learning paradigms that jointly address the aforementioned desiderata. They owe their success to the inclusion of combinatorial elements into machine learning criteria, thereby placing special emphasis on algorithms and data structures. Thus, it is the decades of knowledge encompassed by classical theoretical computer science that will play an essential role in advancing modern machine learning.

This conclusion is underscored by the evolution of the major machine learning paradigms in view of our desiderata of scalability, statistical sophistication, ease of use, and development times. In order to discuss this history we identify, at a high level, the four major design decisions that underlie modern machine learning systems. These are guided by the underlying machine learning paradigm and must act harmoniously along computational and statistical lines for the system to be successful. As a starting point the *data* included for analysis must contain the information necessary to achieve the end goals of the system. The *feature representation* provides a numerical representation of the data, so it must highlight salient information in an accessible manner. It is the lens through which the *learning objective*, a mathematical encoding of the desired analysis along with any relevant prior information, can reason about the data. The objective must be carefully balanced to cater to useful intricacies in the data without overfitting. Finally, the corresponding *learning algorithm* minimizes the objective as accurately and stably as its computational budget will allow in order to construct the machine learning model.

We are now ready to examine the history of the major machine learning paradigms and will focus on supervised learning for simplicity. Classical methods, some of which predate the invention of the digital computer, stand as the pillars of machine learning and include the K-Nearest Neighbors, Naïve Bayes, and Least Squares algorithms for classification and regression [25]. These methods' simplicity results in models that are easy to interpret, are backed by a long history of statistical guarantees, and are massively scalable. However, this simplicity is a double-edged sword in that it makes these methods woefully sensitive to the choice of feature representation. Combating the curse of dimensionality while trying to identify sufficiently expressive features has resulted in unacceptably long development times — in some cases spanning decades without major success.

The complexity of feature engineering gave rise to two fundamentally different sets of paradigms that can be seen as taking primarily continuous or combinatorial perspectives. The former includes Regularized Kernel Methods [57] and Gaussian Processes [52], linear models which focus on *implicitly* projecting a simple feature representation into a more complex one via the *Kernel trick* in hopes of better capturing data nuance. These projections generally create a combinatorial explosion of features, so particular emphasis is placed on regularization. At the time, this powerful combination achieved groundbreaking performance from surprisingly simple base features. However, the implicit nature of these paradigms also creates substantial drawbacks. The Kernel trick generally produces a plurality of spurious features, but it is fundamentally incompatible with a variety of regularizers – particularly ones for feature selection that we will discuss shortly. Thus, the resulting models can become overwhelmed by spurious features because they can diminish their importance, but never discard them. Moreover, it is generally computationally intractable to identify particularly important features since this process requires computing an *explicit* representation. Computationally, these methods do not scale well to large datasets as they rely on storing and manipulating the Kernel matrix, a matrix whose size grows *quadratically* with the number of training examples present.

The corresponding "combinatorial" paradigms stem from Decision Trees [25] and include the various ensemble methods to improve predictions such as Bagging, Boosting, and Random Forests [25]. These paradigms build a collection of predictive trees — hierarchical conjunctive rules — and then combine the trees' predictions in a linear manner that can be interpreted as a disjunction. The combinatorial nature of these paradigms arises from their emphasis on which rules to include in each tree and which trees to include in the collection, tasks that are NP-Complete [27]. Taken in isolation, Decision Trees are easy to use, scalable, resilient to spurious features by virtue of their rule construction, and provide interpretable models. However, their inability to handle linear functions and high variance leads to poor prediction accuracy. The various ensemble methods address these shortcomings to substantially improve predictive performance, but they are less scalable, interpretable, and easy to use.

Our point of departure for "modern" paradigms is one of synthesis, combining the advantages of tree based paradigms with the sophistication of regularization and rich features to cater to our totality of desiderata. The seminal works that enable this perspective include the Lasso [65], which shows that combinatorial functions such as the ℓ_1 norm perform feature selection when used to regularize even simple objectives like Least Squares. Taken together with the connections between ensemble methods and linear models that solve

a sparsity-inducing objective over a basis of weak learners [25], e.g. shallow trees, the stage was set for the paradigm of Structured Sparsity Inducing Regularization (SSIR). The idea underlying SSIR is simple; *explicitly* enumerate a multitude of simple features and add non-differentiable regularizers like the ℓ_1 norm into the objective to perform feature selection and encode relevant prior information. This perspective shifts the focus away from laborious feature engineering and places it on encoding relevant data qualities in functional form via regularization. Model tuning is achieved by varying the weight placed on various regularizers, so exploring the *regularization path* — how the model varies as these parameters change — is essential. The resulting models can achieve state-of-the-art performance, are statistically interpretable, and are backed by a variety of statistical guarantees such as confidence intervals around their coefficients.

The objective pertaining to a linear SSIR model using d features can be written as a minimization problem of the form

$$\underset{w\in\mathbb{R}^d, b\in\mathbb{R}^p}{\text{minimize}}\; L_y(Xw, b) + \lambda R(w) \tag{1.1}$$

where w, b are the model parameters to be learned, L_y is a loss function that encodes any labels if the problem is supervised, and R is the regularization penalty applied to the parameters w. The model makes predictions by combining the unpenalized offset term b with the product Xw. Here $X \in \mathbb{R}^{n\times d}$ is the *feature matrix* whose entries store the value of each of the d features for each of the n training examples. In this context, machine learning algorithms are numerical optimization methods that minimize this objective. Indeed, the success of the SSIR paradigm is in large part due to the dissemination of ideas from convex optimization into the statistics and machine learning communities.

This "loss plus regularizer" format is general and encompasses the majority of machine learning objectives in primal or dual form. The core driver of SSIR's ability to select features is the non-differentiability of the regularization penalty R. For instance, a simple and popular choice is the Elastic-Net regularizer

$$\frac{\alpha}{2}\|w\|_2^2 + (1-\alpha)\|w\|_1$$

which trades off between the sparsity inducing ℓ_1 norm of the Lasso and a stabilizing ridge penalty via $\alpha \in [0, 1]$. When the objective is convex the subdifferential at w given by

$$\psi(w) = X^T \partial_{Xw} L_y(Xw, b) + \lambda \partial_w R(w) \tag{1.2}$$

is an essential component of virtually any optimization method. The condition $\mathbf{0} \in \psi(w)$ is necessary and sufficient for the optimality of w, the quantity $\inf_{\xi \in \psi(w)} \|\xi\|_2$ can be helpful for estimating proximity to the minimizer, and members of $\psi(w)$ are critical for determining descent directions that bring w closer to optimality.

For many choices of R typically seen in machine learning the subdifferential $\partial_w R(w)$ forms a polyhedron at w [15] — a fundamentally combinatorial object. For instance, continuing with our example the Elastic-Net's subgradient set is the offset d-dimensional and scaled unit hypercube $\mathcal{C}_d$

$$\partial_w(\frac{\alpha}{2}\|w\|_2^2 + (1-\alpha)\|w\|_1) = \left\{\alpha w + (1-\alpha)s \middle| s \in [-1,1]^d\right\} = \alpha w + (1-\alpha)\mathcal{C}_d.$$

The key operations for optimization involve checking the interaction between the subdifferential polyhedron defined at the current coefficient estimate and an appropriate matrix-vector product of the form $X^T\zeta$ involving the columns of the explicit feature representation. Scaling this combinatorial problem to handle larger features sets and the subdifferentials of more sophisticated regularizers will necessarily rely on the algorithms and data structures that form the pillars of theoretical computer science.

The other major modern learning paradigm is Deep Learning, which stems from Neural Networks and has produced impressive performance on image and audio processing tasks [37][32]. This form of learning places special emphasis on automating feature engineering, and it does so by constructing multi-layered networks of "neurons" that each output non-linearly transformed linear combinations of their inputs to downstream neurons. This process effectively replaces the historically time consuming process of feature engineering with network engineering, whereby practitioners must fine tune the network topology to optimize prediction accuracy. Unfortunately, the non-convexity of Deep Learning's objectives creates statistical and computational problems. Network training is time consuming since it involves solving for a plethora of variables over unwieldy energy landscapes rife with saddle points and poor local optima. The dissemination of Deep Learning owes its success in large part to advances in computing hardware and infrastructures to make training times tolerable. Moreover, the resulting models are hard to interpret and are not well understood statistically or theoretically.

We end by highlighting the role that algorithmic scalability and speed plays in the machine learning development cycle. Indeed, the ability to rapidly train and experiment with models impacts the quality of the final machine learning system beyond the surface level time spent training the model; model quality and training

speed are inextricably dependent. This occurs because most machine learning system design cycles are iterative, starting with simple models to explore relevant characteristics of the data and gradually escalating the complexity of the system to better capture idiosyncrasies. In fact, IBM's famed Watson artificial intelligence platform [9] is specifically geared to optimize this aspect of the development cycle. Scalability also impacts the models' statistical power. Genome wide association studies (GWAS) are fundamental to bioinformatics and try to uncover genomic markers that are predictive of disease and other traits of interest. Unfortunately, the number of locations that need to be investigated — millions of single nucleotide polymorphisms or billions of locations along the entire genome — relegate most studies to performing univariate hypothesis tests that can only look at each location separately and suffer from low power due to multiple hypothesis testing corrections. Methods that can scale to treat the genome in its entirety to identify and exploit low dimensional genomic structure are essential for improving the power and sophistication of this analysis.

1.1.1 Thesis Contributions and Organization

This thesis uses fundamental ideas from computer science including suffix trees, compression, linear and binary linear programming, network flows, and polyhedra to advance modern machine learning methods for text data. We focus on N-gram models, feature representations for text that represent a document by counting the occurrences of various substrings in it. These models are the mainstay of text based machine learning systems in industry and academia alike, with applications to natural language processing (NLP) , information retrieval, and, increasingly, computational biology. They have been applied successfully in sentiment analysis [50], text categorization [12], author identification [26], DNA function prediction [28], metagenomic binning [68], and numerous other tasks. The allure of N-gram models comes from their simplicity and interpretability; a document corpus is represented by its N-gram feature matrix with each row and column corresponding to a distinct document and N-gram, respectively, and each entry counting the number of occurrences of that N-gram in the document. Correspondingly, the coefficients of a statistical model trained on this representation can be interpreted as a score indicating the relevance of each N-gram to the task. Importantly, the predictive performance of N-gram models has stood the test of time; despite being a decades old feature representation, well tuned N-gram models continue to define state-of-the-art accuracy in numerous tasks [70] [62].

At the simplest extreme, unigrams provide a summary of the word distribution in each document and

serve as an effective baseline representation for a variety of NLP tasks. Higher order N-grams provide more nuance by capturing short-term positional information and can achieve state of the art results on a variety of tasks [70][50]. A canonical example of the value of longer N-grams is given by the phrase "I don't like horror movies, but this was excellent," which fails to convey its positive sentiment when its words are scrambled. Unfortunately, this additional information comes at a cost: a document of n words may contain up to $\Theta(Kn)$ *distinct* N-grams of length K[1]. This growth makes the memory and computational burden of training N-gram models beyond bigrams impractical for large natural language corpora. Statistically, these larger feature representations suffer from the curse of dimensionality [25] and may lead the model to overfit, so careful regularization is necessary.

The first half of this thesis, presented in Chapter 2, caters to the SSIR learning paradigm, by ameliorating the computational burden of learning with long N-grams. We demonstrate how the structure of suffix trees can be used to store and multiply[2] any N-gram feature matrix in time and space that is at most *linear* in the length of the underlying corpus. This is a *quadratic* improvement over prior methods, and we verify that it provides substantial computational savings on real problems. As most learning algorithms rely on matrix-vector multiplication to learn and predict, our results equate the computational cost of learning with N-gram matrices to scanning through the original corpus. Our method can speed up *any* learning algorithm that exhibits such structure by simply replacing its multiplication routine with ours. Fast multiplication is possible by means of a specialized data structure that efficiently represents the algebraic structure of the N-gram matrix. We also provide a linear running time and memory framework that can be used to permanently store the corpus in a format optimized for machine learning. Given a new task this framework filters N-grams by various criteria, computes necessary column normalizations, and outputs an N-gram feature matrix, represented by our efficient data structure, that is custom tailored to the task at hand. The emphasis of this framework is minimality; by only storing the topological structure of the suffix tree we achieve memory requirements that are comparable to storing the original document corpus.

This work also shows how suffix trees can address the statistical issues associated with long N-grams by potentially *quadratically* reducing the number of features that need to be considered. These savings are particularly important in the regime where the correct feature granularity is unknown as exemplified by representing a natural language corpus at the character level of instead of the statistically easier word level.

[1] We use N-grams of length K to mean N-grams of length *at most* K for brevity.
[2] Multiplication always refers to matrix-vector multiplication.

The dimensionality reduction afforded by suffix trees adds robustness to the learning system, making it more likely that the regularization will gracefully handle the spurious features introduced by working at an overly granular level.

The second part of this thesis, presented in Chapter 3, looks to compression to present an unorthodox way of performing feature selection that can be interpreted as a novel combinatorial form of Deep Learning based on string concatenation. Most practitioners compress their large datasets for storage and uncompress them to extract features. Our viewpoint is based on the minimum description length (MDL) principle; the space savings afforded by compression may be due to fundamental structure that is also useful for learning, so we look to extract feature representations directly from the compression. Indeed, the minimum description length principle can be used to justify regularization as well as various model selection criteria [21], while both, unsupervised problems deep autoencoders [56] and the classical K-means algorithm, seek a parsimonious description of data. Meanwhile, off-the-shelf compressors, such as LZ-77 [74], have been successfully applied to natural language problems as Kernels that compute pairwise document similarities [8].

We demonstrate that explicit features derived from off-the-shelf compressors are unstable because of their sensitivity to the order in which documents are compressed. We propose an order–invariant paradigm, *Dracula*, so called because it simultaneously finds a useful *data representation and compression using linear-programming approximations* of the criterion that motivates dictionary-based compressors like LZ-78 [75]. Dracula finds an *explicit* feature representation for the documents in a corpus by learning a dictionary of N-grams that is used to losslessly compress the corpus. It then recursively compresses the dictionary. This recursion makes Dracula select a deep representation that considers all N-grams appearing in the corpus and promotes similar and frequently occurring ones to enter into its dictionary. The N-grams comprising the dictionary can then be used as features either by counting how many times each N-gram is directly used in the compressed representation of a corpus document or by also taking into account how they are themselves compressed, i.e. the dictionary structure. This dictionary structure induces an ordering among N-grams akin to low and high frequency components in Fourier analysis, so taking into account dictionary structure regularizers models towards lower frequency components.

Our paradigm is expressed as a binary linear program that can viewed as a linear program over a sufficiently constrained polyhedron or relaxed to a linear program by relaxing its integrality constraints. This is

a notable departure from traditional Deep Learners [56, 61, 36], which are formulated as non-convex, non-linear optimization problems. This structure makes it possible to analyze Dracula in view of well known techniques from convex analysis (e.g. the KKT conditions), polyhedral combinatorics, graph theory, and other devices from theoretical computer science. We use this structure to prove that solving Dracula is NP-Complete but also that it is comprised of two easy combinatorial problems with total unimodularity and network flow structure. We leverage this structure to construct and experiment with approximate algorithms for a shallow special case of Dracula, termed Compressive Feature Learning (CFL), that can easily scale to a gigabyte of text despite having to a repeatedly solve a linear program involving nearly 1 billion variables.

There are two major use cases for Dracula. The first is to compress data to minimize on-disk storage space and to use the resulting representation directly for features. This is particularly pertinent for unsupervised and exploratory learning scenarios where the end goals of the analysis can be vague, e.g. to uncover interesting structure, so a parsimonious feature representation can accentuate pertinent structure. The second use case is based on the observation that Dracula's objective is parameterized by a storage cost model which can take on arbitrarily values; it is possible to encode a maximally space wasting cost model that includes all possible N-grams into its "compressed" representation. It is therefore possible to fine tune Dracula's representation to the learning problem at hand by varying this cost model, perhaps starting at the cost model pertaining to traditional notions of storage space. Here Dracula's problem structure pays statistical dividends as it allows us to formally study its "regularization path" when parameterized by the cost model. We identify parameters that control the depth and diversity of its dictionary and prove that its regularization path is well behaved in that it obeys combinatorial structure dictated by the data, so solutions will not jump unexpectedly. We also verify that Dracula does indeed select useful features for a variety of tasks spanning natural language and bioinformatics, observing that these tasks prefer cost models that do not stray too far from minimizing on disk space.

1.1.2 Related Work

Suffix trees and arrays are used by [69], [64], and [54] for Kernels that efficiently compute pair-wise document similarities based on N-grams. Computing the similarity of all document pairs limits Kernels to moderately sized datasets and the lack of explicit features prevents the use of sparsity inducing regularizers such as in the Lasso [65]. Next, [72] use suffix trees to identify useful N-grams in a text corpus and to show that

the all N-gram matrix may be pruned since it contains redundant columns. We show in Section 2.3 that the resulting N-gram matrix may still have too many entries to be practical for large corpora and observe this experimentally. Suffix trees are also used by [71] to efficiently represent and perform inference with a hierarchical process for text. Finally, while [3] and [29] provide space efficient frameworks for working with suffix arrays, our framework is specialized to statistical processing and achieves greater memory efficiency.

The work on CFL and Dracula draws on a deep connection between data compression and machine learning, exemplified early on by the celebrated MDL principle [55]. More recently, researchers have experimented with off-the-shelf compression algorithms as machine learning subroutines. Instances are Frank et al.'s [19] compression-based approach to text categorization, as well as compression-based distance measures, where the basic intuition is that, if two texts x and y are very similar, then the compressed version of their concatenation xy should not be much longer than the compressed version of either x or y separately. Such approaches have been shown to work well on a variety of tasks such as language clustering [5], authorship attribution [5], time-series clustering [14, 30], anomaly detection [30], and spam filtering [8].

Distance-based approaches are akin to kernel methods, and thus suffer from the problem that constructing the full kernel matrix for large datasets might be infeasible. Furthermore, Frank et al. [19] deplore that "it is hard to see how efficient feature selection could be incorporated" into the compression algorithm. But Sculley and Brodley [59] show that many compression-based distance measures can be interpreted as operating in an implicit high-dimensional feature space, spanned by the dictionary elements found during compression. We build on this observation to address Frank et al.'s above-cited concern about the impossibility of feature selection for compression-based methods. Instead of using an off-the-shelf compression algorithm as a black-box kernel operating in an implicit high-dimensional feature space, we develop an optimization-based compression scheme *whose explicit job it is* to perform feature selection.

It is illuminating to discuss a related approach suggested (as future work) by Sculley and Brodley [59], namely "to store substrings found by Lempel–Ziv schemes as explicit features". This simplistic approach suffers from a serious flaw that our method overcomes. Imagine we want to extract features from an entire corpus. We would proceed by concatenating all documents in the corpus into a single large document D, which we would compress using a Lempel–Ziv algorithm. The problem is that the extracted substrings are dependent on the order in which we concatenate the documents to form the input D. For the sake of concreteness, consider LZ77 [74], a prominent member of the Lempel–Ziv family (but the argument applies

equally to most standard compression algorithms). Starting from the current cursor position, LZ77 scans D from left to right, consuming characters until it has found the longest prefix matching a previously seen substring. It then outputs a pointer to that previous instance—we interpret this substring as a feature—and continues with the remaining input string (if no prefix matches, the single next character is output). This approach produces different feature sets depending on the order in which documents are concatenated. Even in small instances such as the 3-document collection $\{D_1 = abcd, D_2 = ceab, D_3 = bce\}$, the order (D_1, D_2, D_3) yields the feature set $\{ab, bc\}$, whereas (D_2, D_3, D_1) results in $\{ce, ab\}$ (plus, trivially, the set of all single characters).

1.2 Background

Let Σ be a finite vocabulary with a strict total ordering $\prec$ over its elements. A *document* $D = x_1x_2...x_n$ of length n is a list of n characters drawn from Σ and an N-gram is any substring of the document(s) under discussion. We will refer to each of the n suffixes in D via $D[i] = x_ix_{i+1}...x_n$. We denote the set of all substrings in D by $D^* = \{\, x_i...x_{i+k} \mid 1 \leq i \leq n, 0 \leq k \leq n - i \,\}$ and the set of all substrings in a document corpus of N documents $\mathcal{C} = \{D_1, \ldots, D_N\}$ as $\mathcal{C}^* = \bigcup_{i=1}^{N} D_i^*$.

Given a subset $\mathcal{S} \subseteq \mathcal{C}^*$ of the set of substrings in (any of) the documents, entry X_{is} of the N-gram matrix $X \in \mathbb{Z}_+^{N \times |\mathcal{S}|}$ counts how many times substring $s \in \mathcal{S}$ appears in document D_i. We use M_i to indicate the i^{th} column of matrix M; when each column pertains to a specific mathematical object, such as an N-gram or tree node, we may use that object as an index (to avoid imposing a particular ordering over the objects).

A *compact tree* $\mathcal{T} = (V, E)$ is a tree with nodes V and edges E where every internal node is required to have at least 2 children. This ensures that if $\mathcal{T}$ has n leaves, then there are at most $n - 1$ internal nodes. We use $\text{ch}(v) \subset V$ and $\text{p}(v) \in V$ to denote the children and parent of $v \in V$, respectively. The root node is given by $\text{root}(\mathcal{T})$, the depth of any node $v \in V$ is $\text{d}(v)$ (with $\text{d}(\text{root}(\mathcal{T})) = 1$), and $\text{depth}(T)$ is the maximum depth of any node in V. Finally, a *branch* of $\mathcal{T}$ is a path starting at the root and ending at a leaf; we will use the terminal leaf to identify branches. We will also be concerned with subtrees $\hat{\mathcal{T}} = (\hat{V}, \hat{E})$ of $\mathcal{T}$ which contain a subset $\hat{V} \subset V$ of its nodes. We allow the new edge set $\hat{E}$ to be arbitrary and add a second argument to $\text{ch}(v, \hat{E})$ and $\text{p}(v, \hat{E})$ to indicate that parent/child relationships are taken with respect to this new edge set.

For any $s \in \mathcal{C}^*$ a *pointer* p is a triple $p = (s, l \in \{1, \ldots, |s|\}, z \in \mathcal{S})$ indicating that $z = s_l \ldots s_{l+|z|-1}$. We say that p *uses* z at location l in s. Let $\mathcal{P}$ be the set of all valid pointers and for any $P \subset \mathcal{P}$ we use

$P(s) = \{p \in P | p = (s, l, z)\}$ to select pointers whose first element is s, e.g. $\mathcal{P} = \cup_{s \in \mathcal{S}} \mathcal{P}(s)$. Moreover, P *uses* $z \in \mathcal{S}$ if there is some $p \in P$ using z, and P *reconstructs* $s \in \mathcal{S}$ if every location in s is covered by at least one pointer, i.e. $\cup_{(s,l,v) \in P(s)} \{l, \ldots, l + |v| - 1\} = \{1, \ldots, |s|\}$. Conceptually, s is recovered from P by iterating through the $(s, l, v) \in P$ and "pasting" a copy of v into location l of a blank string. It will be helpful to define $\mathcal{P}_\mathcal{C} = \cup_{s \in \mathcal{C}} \mathcal{P}(s)$ to be the set of pointers that can only be used to reconstruct the corpus.

1.2.1 Suffix Trees and Arrays

Given a document $D = x_1 x_2 ... x_n$ whose characters belong to an alphabet Σ, the *suffix tree* $\mathcal{T}_D = (V, E)$ for D is a compact tree with n leaves, each of which corresponds to a distinct suffix of D and is numbered according to the starting position of the suffix $1, \ldots, n$. The edges along branch i are labeled with non-empty substrings that partition $D[i]$: suffix $D[i]$ can be recovered by concatenating the edge labels from the root to leaf i. Let $\mathrm{l}(e)$ for $e \in E$ be the label of edge e and define the *node character* $c(v)$ of any non-root node $v \in V$ to be the first character of $\mathrm{l}((\mathrm{p}(v), v))$. The nodes of $\mathcal{T}_D$ are constrained so that siblings may not have the same node character and are ordered according to the $\prec$ relation on these characters. These constraints ensure that every node has at most $|\Sigma|$ children and they allow for well-defined traversals of $\mathcal{T}_D$. Moreover, every substring $s \in D^*$ is represented by a unique path in $\mathcal{T}_D$ that starts at the root node and terminates in — possibly the middle of — an edge. Similarly to suffixes, s equals the concatenation of all characters encountered along edges from the root to the path's terminus (only a prefix of the final edge will be concatenated if the path ends in the middle of an edge).

Remarkably, $\mathcal{T}_D$ can be constructed in $O(n)$ time (Gusfield, 1997) and has n leaves and at most $n - 1$ internal nodes, yet it represents all $O(n^2)$ distinct substrings of D. This is possible because any substrings whose path representation in $\mathcal{T}$ ends at the same edge belong to the same *equivalence class*. In particular, for $v \in V \backslash \{\mathrm{root}(\mathcal{T}_D)\}$ suppose that edge $(\mathrm{p}(v), v)$ has a label $t = x_i \ldots x_{i+k}$ and let s be the string obtained by concatenating the edge labels on the path from $\mathrm{root}(\mathcal{T}_D)$ down to $p(v)$. Then the strings $S(v) = \{sx_i, sx_{i+1}, \ldots, sx_{i+k}\}$ belong to the same equivalence class because they occur in the same locations, i.e. if sx_i starts at location l in D, then so do all members of $S(v)$. For example, in the string "xaxaba" the substrings "x" and "xa" belong to the same equivalence class.

The *generalized suffix tree* $\mathcal{T}_\mathcal{C}$ for a document corpus $\mathcal{C}$ of n words compactly represents the set of all substrings in $\mathcal{C}^*$ and has n leaves pertaining to every suffix of every document in $\mathcal{C}$. Leaves are also annotated

with the document they belong to and $\mathcal{T}_\mathcal{C}$ inherits all of the linear-time storage and computational guarantees of the regular suffix tree (with respect to the corpus length n).

Chapter 2

Efficient Learning with Bag of N-Gram Models

This Chapter presents our work on the linear–time and memory matrix-vector multiplication algorithm for the N-gram feature matrix. In what follows we will *always* take X to be an N-gram matrix for an implicitly given corpus. After discussing preliminaries in Section 2.1, Section 2.2 describes the importance of matrix-vector multiplication in machine learning. Section 2.3 derives the fast multiplication algorithm by showing that after redundant columns in the N-gram matrix are removed, the algebraic structure of the resulting submatrix is encoded by the suffix tree of the underlying corpus. We then investigate the statistical ramifications of this matrix in various common learning scenarios in Section 2.4. Section 2.5 presents our preprocessing framework. Timing and memory benchmarks that demonstrate the efficacy of the multiplication algorithm are presented in Section 2.6. We also find that high-order N-grams can improve prediction accuracy in large-scale sentiment analysis tasks.

2.1 Tree Traversals and Storage

The majority of algorithms in this chapter can be expressed as a bottom-up or top-down traversal of a tree $T = (V, E)$ (typically the suffix tree or one of its subtrees) in which information is only exchanged between a parent and its children. Given a fixed ordering of V, the necessary information for a traversal is the topology

of T, i.e. its parent-child relationships, as well as any node annotations necessary for the computation. We use two formats which efficiently store this information and make traversals easy: the *breadth-first format* (BFF) and *preorder depth-first format* (DFF). In both cases we distinguish between the internal nodes and leaves of T and divide them into their respective sets $I \cup L = V$. In the BFF we order the nodes of I according to their *breadth-first traversal* whereas in the DFF we order the nodes of I according to their *preorder depth first traversal*; both formats assign indices $[0, \ldots, |I|)$ to the nodes in I. Note that for these traversals to be well defined we assume that the children of each node are ordered in some (arbitrary) but fixed manner. Next, the leaves of T, i.e. L, are assigned indices $[|I|, \ldots, |V|)$ so that if $u, v \in L$ and $p(u)$ comes before $p(v)$ – note that both parents must be in I – then u comes before v. This ordering ensures that leaves are ordered into contiguous blocks with respect to their parent and that the blocks are in the same order as I.

A pair of arrays $(\text{ch}^I, \text{ch}^L)$, each of size $|I|$, capture the topology of T: for all $v \in I$, $\text{ch}^I_v = |\text{ch}(v) \cap I|$ stores the number of internal children of v and $\text{ch}^L_v = |\text{ch}(v) \cap L|$ stores the number of leaf children of v. The number of bits needed to store this topology is

$$|I|(\lceil \log_2 U(I) \rceil + \lceil \log_2 U(L) \rceil) \tag{2.1}$$

where $U(I), U(L)$ are the largest values in ch^I, ch^L respectively, i.e. the largest number of internal/leaf children for any node. Given node annotations in the same order as the BFF or DFF, top-down/bottom-up traversals are easy to perform by a linear sweep of the annotations and ch^I, ch^L arrays. All memory access is sequential and can be performed efficiently by standard (i.e. desktop) memory and processors.

A speed/memory trade-off exists for the two formats. The amount of random access memory necessary for a traversal is proportional to the *depth* of T for DFF versus the *width* of T for the BFF. As we discuss in Section 2.5, the former is likely to be smaller than the latter for our purposes. The space savings of the DFF are achieved by maintaining a stack of active nodes pertaining to the current branch being processed. The additional logic required for this bookkeeping makes the DFF slightly slower than the BFF for the traversal. As such, the DFF is useful for more complicated computations in which the amount of information stored per node may be large, whereas the BFF is useful for simple computations that will be performed many times.

2.2 Matrix Multiplication and Learning

We briefly discuss the importance of matrix-vector multiplication for learning. Let $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^d$ be N data points with corresponding labels $y_1, \ldots, y_N \in \mathcal{Y}$ and let $X \in \mathbb{R}^{N \times d}$ be the feature matrix that stores $\mathbf{x}_i$ as its i^{th} row. Matrix-vector multiplication operations abound in all phases of supervised and unsupervised learning: basic preprocessing that computes normalizings factors of the form $X^T\mathbf{1}$ (or $X\mathbf{1}$) for every feature (or data point); screening rules that use $|X^T y|$ (when $\mathcal{Y} \subset \mathbb{R}$) to exclude uninformative features (Tibs., 2010); or predictions of the form $f(Xw)$ where w is a learned vector of weights.

Multiplication is also essential for many of the optimization techniques that lie at the core of these learning algorithms. A variety of learning problems can be expressed as optimization problems of the form

$$\underset{w \in \mathbb{R}^d, \beta \in \mathbb{R}^p}{\text{minimize}} \; L_y(Xw, \beta) + \lambda R(w) \tag{2.2}$$

where w, β are the learning parameters, L_y is a loss function that encodes the y_i labels (if the problem is supervised), and R is a regularization penalty. It is important to remember that this framework captures a number of *unsupervised* learning problems as well, such as Principle Component Analysis, which is useful directly and as a preprocessing step for clustering, deep learning, and other techniques (Hastie, 2001). Any (sub)gradient[1] of 2.2 with respect to w is given by

$$g_w \in X^T \partial_{Xw} L(Xw, \beta) + \lambda \partial_w R(w). \tag{2.3}$$

where $\partial_z f(z)$ is the subdifferential of f with respect to z.

Since every (sub)gradient descent method (Parikh, 2013) or accelerated variant critically relies on g_w as a search direction, computing Xw and then $X^T[\partial_{Xw} L(Xw, \beta)]$ is essential and often the most costly part of the optimization. A number of other popular large-scale optimization methods also reduce to multiplying X repeatedly. These include Krylov subspace algorithms such as the conjugate gradient method, and various quasi-Newton methods including BFGS and its limited memory variant (Nocedal, 2006).

[1] To handle non-differentiable objectives, see (Parikh, 2013).

2.3 Fast Multiplication

This section presents our fast multiplication algorithm. Let $\mathcal{T}_\mathcal{C} = (V, E)$ be the suffix tree for a document corpus $\mathcal{C} = \{D_1, \ldots, D_N\}$ and let X be an N-gram matrix containing a column for every $s \in \mathcal{S} \subseteq \mathcal{C}^*$, i.e. the N-grams we are interested in. In order to uncover the necessary algebraic structure for our algorithm we must first remove redundant columns in X. As observed in [72], redundant columns occur whenever strings in $\mathcal{S}$ belong to the same equivalence class. This implies the following lemma:

Lemma 1. *For any $v \in V$, any $s, s' \in \mathcal{S} \cap S(v)$ have the same distribution among the documents in $\mathcal{C}$ so $X_s = X_{s'}$.*

We remove this redundancy by working with the *node matrix* $\mathcal{X} \in \mathbb{Z}_+^{N \times M}$, a submatrix of X that contains a single column for the M equivalence classes present in $\mathcal{S}$. Formally, node $v \in V$ is present in X if $S(v) \cap \mathcal{S} \neq \emptyset$ and we define $\mathcal{V} \subset V$[2] to be the set of all nodes present in X. Column $\mathcal{X}_v$ for $v \in \mathcal{V}$ is obtained by picking an arbitrary $s \in S(v) \cap \mathcal{S}$ and setting $\mathcal{X}_v = X_s$. We can also reconstruct X from $\mathcal{X}$ by replicating column $\mathcal{X}_v$ $|S(v) \cap \mathcal{S}|$ times; this underscores the inefficiency in the N-gram matrix.

2.3.1 Linear Dependencies in the Node Matrix

We are now ready to show how the topology of $\mathcal{T}_\mathcal{C}$ determines the linear dependencies among the columns of $\mathcal{X}$. Central to our analysis is the lemma below, which shows that the document frequency of any node is determined entirely by the leaves of its subtree:

Lemma 2. *The number of times node $v \in V \backslash \{root(\mathcal{T}_\mathcal{C})\}$ appears in document $D_i \in \mathcal{C}$ equals the number of leaves that belong to D_i in the subtree rooted at v.*

The simplest case occurs when $\mathcal{V} = V \backslash \{\text{root}(\mathcal{T}_\mathcal{C})\}$, i.e. every node in $\mathcal{T}_\mathcal{C}$ (except for the root) has a corresponding column in $\mathcal{X}$. In this case lemma 2 directly establishes a recursive definition for the columns of $\mathcal{X}$:

$$\mathcal{X}_v = \begin{cases} \mathbf{e}^N_{\text{doc}(v)} & \text{if } v \text{ is a leaf} \\ \sum_{u \in \text{ch}(v)} \mathcal{X}_u & \text{otherwise.} \end{cases} \tag{2.4}$$

[2] Note that $\mathcal{V}$ never includes the root node.

Here $\mathbf{e}_i^N$ is the i^{th} canonical basis vector for $\mathbb{R}^N$ and $\text{doc}(v)$ indicates the document index leaf v is labeled with. Importantly, 2.4 shows that the column corresponding to any internal node can be expressed as a simple linear combination of the columns of its children. This basic property lies at the core of our fast multiplication algorithm.

We now show how to apply the reasoning behind 2.4 to the more general case when $\mathcal{V}$ is an arbitrary subset of V, i.e. a node's children may be partly missing. Define $\mathcal{T}_\mathcal{C}(\mathcal{V}) = (\hat{V}, \hat{E})$, the *restriction* of $\mathcal{T}_\mathcal{C}$ to $\mathcal{V}$, to be a tree with nodes $\hat{V} = \mathcal{V} \cup \{\text{root}(\mathcal{T}_\mathcal{C})\}$. In addition, for any $v \in V \backslash \{\text{root}(\mathcal{T}_\mathcal{C})\}$ let $\text{la}(v, \hat{V}) \in \hat{V}$ be the closest *proper ancestor* of v in $\mathcal{T}_\mathcal{C}$ that is also in $\hat{V}$; since $\text{root}(\mathcal{T}_\mathcal{C}) \in \hat{V}$, this mapping is always well defined. The edge set $\hat{E}$ preserves the ancestor relationships among the nodes in $\hat{V}$: every $v \in \mathcal{V}$ is connected to $\text{la}(v, \hat{V})$ as a child. An inductive argument shows that if $u, v \in \hat{V}$, then u is an ancestor of v in $\mathcal{T}_\mathcal{C}$ *if and only if* u is also an ancestor of v in $\mathcal{T}_\mathcal{C}(\mathcal{V})$.

Associated with $\mathcal{T}_\mathcal{C}(\mathcal{V})$ is a matrix $\Phi \in \mathbb{Z}_+^{N \times |\mathcal{V}|}$ that subsumes the role of leaf document labels. Φ contains a column for every node $v \in \mathcal{V}$ and accounts for all of the leaves in $\mathcal{T}_\mathcal{C}$. When v is a leaf in $\mathcal{T}_\mathcal{C}$ and v is included in $\mathcal{V}$ we set $\Phi_v = \mathbf{e}^N_{\text{doc}(v)}$. Otherwise, v is accounted for in $\Phi_{\text{la}(v,\hat{V})}$, the column pertaining to v's closest ancestor in $\mathcal{V}$. In particular, if $u \in \mathcal{V}$ is *not* a leaf in $\mathcal{T}_\mathcal{C}$, then

$$\Phi_u = \sum_{\substack{v \in \text{leaves}(\mathcal{T}_\mathcal{C}) \backslash \mathcal{V} \\ \text{la}(v,\hat{V}) = u}} \mathbf{e}^N_{\text{doc}(v)}. \tag{2.5}$$

This bookkeeping allows us to relate the columns of $\mathcal{X}$ when $\mathcal{V}$ is any subset of V:

Theorem 1. *The columns of the node matrix $\mathcal{X}$ for $\mathcal{V} \subseteq V \setminus \{root(\mathcal{T}_\mathcal{C})\}$ are given recursively by*

$$\mathcal{X}_v = \Phi_v + \sum_{u \in ch(v;\hat{E})} \mathcal{X}_u$$

where Φ and $\mathcal{T}_\mathcal{C}(\mathcal{V}) = (\hat{V}, \hat{E})$ are defined above.

This theorem shows that $\mathcal{X}_v$ is a simple linear combination of the columns of its children in $\mathcal{T}_\mathcal{C}(\mathcal{V})$ plus a correction term in Φ. We utilize this structure below to give a fast matrix-vector multiplication algorithm for node matrices.

2.3.2 Fast Multiplication Algorithm

A simple application of Theorem 1 shows that the matrix-vector product $\mathcal{X}w$ for $w \in \mathbb{R}^{|\mathcal{V}|}$ can be obtained by recursively collecting entries of w into a vector $\beta \in \mathbb{R}^{|\mathcal{V}|}$:

$$\beta_v = w_v + \beta_{\mathrm{p}(v;\hat{E})} \tag{2.6a}$$

$$\mathcal{X}w = \Phi\beta \tag{2.6b}$$

Here we use the convention $\beta_{\mathrm{root}(\mathcal{T}_\mathcal{C}(\mathcal{V}))} = 0$. The transposed operation $\mathcal{X}^T y$ for $y \in \mathbb{R}^N$ can also be written recursively by expressing each entry as

$$(\mathcal{X}^T y)_v = \Phi_v^T y + \sum_{u \in \mathrm{ch}(v;\hat{E})} (\mathcal{X}^T y)_u. \tag{2.7}$$

Equations 2.6-2.7 lead to the following theorem, for which we provide a proof sketch:

Theorem 2. *Let $\mathcal{C}$ be a document corpus of n words and let $\mathcal{X}$ be any node matrix derived from this corpus. Then $\mathcal{X}$ requires $O(n)$ memory to store. Multiplying a vector with $\mathcal{X}$ or $\mathcal{X}^T$ requires $O(n)$ operations.*

Proof. Vector β in equation 2.6 can be computed in $O(|\mathcal{V}|) \in O(n)$ operations by a top-down traversal that computes each of its entries in constant time. The matrix Φ is a sparse matrix with *at most* one entry per leaf of the suffix tree $\mathcal{T}_\mathcal{C}$, i.e. at most n entries. It follows that the product $\Phi\beta$ requires $O(n)$ operations which proves the theorem for multiplication with $\mathcal{X}$. The transposed case follows similarly by noting that we must compute a matrix-vector product with Φ^T and perform a bottom-up traversal that performs constant time operations for every node in $\mathcal{V}$. □

Efficiency Gains

We use naïve multiplication to mean sparse matrix-vector multiplication in what follows. The following complexity separation result is based on examples which show that naïve multiplication with the N-gram matrix X can be asymptotically slower than naïve multiplication with $\mathcal{X}$, which in turn can be asymptotically slower than multiplication with our recursive algorithm.

Theorem 3. *There exists documents of n words for which*

1. *The all N-grams matrix X requires $\Theta(n^2)$ storage and operations to multiply naïvely.*

2. *The all N-grams node matrix $\mathcal{X}$ requires $\Theta(n\sqrt{n})$ storage and operations to multiply naïvely.*

3. *In all cases recursive multiplication of the node matrix requires $O(n)$ storage and operations.*

Proof. We start with a canonical example from the suffix tree literature which highlights the inefficiency of the N-gram matrix. Suppose that the document corpus consists of a single document $D_1 = c_1c_2 \dots c_n$ of n distinct characters, i.e. $c_i \neq c_j$ if $i \neq j$. There are $\frac{n^2+n}{2}$ distinct substrings in this document, so the N-gram matrix pertaining to all possible N-grams is a row vector of $\frac{n^2+n}{2}$ ones. In contrast, the node matrix $\mathcal{X}$ only consists of n entries pertaining to every distinct character. Direct multiplication with X requires $\Theta(n^2)$ operations whereas multiplication with $\mathcal{X}$ requires $\Theta(n)$ operations.

Next, to show that the node matrix can be inefficient, consider a document corpus comprised of K documents and an alphabet of K distinct characters $c_1, \dots, c_K$. The i^{th} document $D_i = c_1c_2 \dots c_i$ is comprised of the first i characters of the alphabet and the total corpus length is $n = \frac{K^2+K}{2}$. By inspecting the structure of the suffix tree $\mathcal{T}_\mathcal{C}$ for this corpus, it is possible to show that both the all N-grams matrix X and all N-grams node matrix $\mathcal{X}$ have $\Theta(K^3)$ non-zero entries and thus require $\Theta(n\sqrt{n})$ memory to store and $\Theta(n\sqrt{n})$ operations to multiply.

In particular, consider the branch β_1 corresponding to suffix $D_K[1]$, i.e. the suffix consisting of K characters and equal to the entire document D_K. Note that there is a document D_i equalling every *prefix* $[i]D_K = c_1c_2 \dots c_i$ of D_K. By construction, for $i = 1, \dots, K-1$, every occurrence of the substring $[i]D_K$ in $\mathcal{C}$ is either followed by c_{i+1} (for example in document D_{i+1}) or is the end of a document (i.e. D_i). This structure implies that β_1 contains $K-1$ internal nodes pertaining to the first $K-1$ characters in $D_K[1]$ and that the edge labels connecting these nodes contain a single character. For $i < K$ the internal node pertaining to character c_i has two children: a leaf indicating the end of document D_i and another internal node corresponding to character c_{i+1}. The final node in β_1 has character label c_K and is a leaf signalling the end of D_K. If we count this node (for simplicity), the node pertaining to character i appears in exactly $K-i+1$ documents, so the column for substring $[i]D_K$ in the (all) node matrix $\mathcal{X}$ contains $K-i+1$ non-zero entries. The K prefixes of D_K each pertain to a node in β_1 and have a column in $\mathcal{X}$ with a total of

$$\sum_{i=1}^{K}(K-i+1) = \frac{K^2+K}{2}$$

non-zero entries.

The other strings in the corpus are formed in a similar manner by looking at the prefixes of $c_i \dots c_K$, i.e. all prefixes of every suffix of D_K. Note that the corpus length is $n = \frac{K^2+K}{2}$ and there are n distinct substrings, equivalence classes, and nodes in $\mathcal{T}_\mathcal{C}$ (that correspond to these equivalence classes) so $\mathcal{X}$ has n columns. By iterating our earlier reasoning we see that branch β_k corresponds to (all prefixes of) suffix $D_K[k]$ and it accounts for k of these nodes. In total these k nodes contribute

$$\sum_{i=1}^{k}(k-i+1) = \frac{k^2+k}{2} \tag{2.8}$$

non-zero entries to $\mathcal{X}$.

By summing equation 2.8 from $k = 1, \dots, K$ we find that $\mathcal{X}$ has $\Theta(K^3)$, i.e. $\Theta(n\sqrt{n})$, non-zero entries and therefore is as inefficient as the naïve all N-grams matrix. □

2.3.3 Matrix Data Structure

The fast multiplication algorithm can be performed directly on the suffix tree derived from $\mathcal{C}$, but it is faster to use a dedicated data structure optimized for the algorithm's memory access patterns. The breadth-first multiplication tree (BFMT) stores the topology of $\mathcal{T}_\mathcal{C}(\mathcal{V})$ in the BFF (discussed in Section 2.1) and the frequency information in Φ as a sparse matrix in a modified compressed sparse column (csc) format (see below) whose columns are ordered according to the order of the BFF. We chose this format because executing equations 2.6 and 2.7 requires a simple linear sweep of the BFMT. We expect that the vectors being multiplied can be stored in memory and therefore opted for the speed afforded by the BFF instead of the memory savings of the DFF.

The total number of bits necessary to store the BFMT is given by equation 2.1 along with the total number of bits necessary to store Φ, which is

$$|\mathcal{V}|\lceil\log_2 U^\Phi\rceil + \text{nz}(\lceil\log_2 M\rceil + \lceil\log_2 N\rceil). \tag{2.9}$$

Here $U^\Phi = \max_{v\in\mathcal{V}}\|\Phi_v\|_0$ is the largest number of non-zero elements in a column of Φ, nz is the total number of non-zero elements in Φ, and M is the largest entry in Φ. It is easy to verify that $|\mathcal{I}| \le |\mathcal{V}| \le \text{nz} \le n$ and the term involving nz typically dominates the memory requirements.

The standard compressed sparse column (CSC) format for a sparse $M \times N$ matrix X consisting of nz non-zero entries stores three arrays:

1. The jc array, an array of size $N + 1$ such that $\text{jc}[i + 1] - \text{jc}[i]$ gives the number of non-zero entries in column i.

2. The ir array, an array of size nz in which indices $\text{jc}[i], \ldots, \text{jc}[i + 1] - 1$ contain the row ids of the non-zero entries in column i.

3. The x array, a double array of size nz containing the non-zero entries of X in the same order that they are listed in the ir array.

This matrix format is inefficient when storing frequency data since we know all entries in x are non-negative integers. Moreover, the number of bits needed to store each index in the jc array is $\lceil \log_2 nz \rceil$ which can be significantly larger than $\lceil \log_2 U^X \rceil$ where U^X is the largest number of non-zero elements in any column. Our modified CSC format simply replaces the jc array with an integer array of size N that stores the number of non-zero elements in each column and it replaces x by an integer array of frequency counts. This modification can lead to substantial savings when appropriate.

2.4 Statistical Considerations

We now discuss how several common machine learning scenarios can be adapted to use our representation of the node matrix or preferably, to treat multiplication with $\mathcal{X}$ as a black-box routine. The most straightforward use case is to replace the original N-gram matrix with the more succint node matrix. By virtue of our discussions in the previous section, there can be *quadratically* fewer features to consider with $\mathcal{X}$ because of the redundant columns in the original N-gram feature matrix. The effect that these features have depends on the underlying learning problem. For instance, consider ridge regression when using the original N-gram feature matrix

$$\underset{w \in \mathbb{R}^d}{\text{minimize}} \frac{1}{2}\|y - Xw\|_2^2 + \frac{\lambda}{2}\|w\|_2^2. \tag{2.10}$$

It is easy to show that if $\lambda > 0$ and N-grams s, t belong to the same equivalence class, then $w_s = w_t$. We can simulate the effect of these duplicated variables by collecting terms. Let $\mathcal{S}$ be the set of N-grams present in X, $\mathcal{V}$ the set of suffix tree nodes present in X, and define $\mathcal{S}(v) = S(v) \cap \mathcal{S}$ for brevity. For all

$v \in \mathcal{V}$ let $z_v = |\mathcal{S}(v)| w_s$ for some $s \in \mathcal{S}(v)$. Then $\sum_{s \in \mathcal{S}(v)} X_s w_s = \mathcal{X}_v z_v$ and $\sum_{s \in \mathcal{S}(v)} w_s^2 = |\mathcal{S}(v)|^{-1} z_v^2$ so problem 2.10 is equivalent to a *smaller* weighted ridge regression using $\mathcal{X}$:

$$\underset{z \in \mathbb{R}^{|\mathcal{V}|}}{\text{minimize}} \frac{1}{2} \|y - \mathcal{X} z\|_2^2 + \frac{\lambda}{2} \sum_{v \in \mathcal{V}} \frac{z_v^2}{|\mathcal{S}(v)|}. \tag{2.11}$$

In effect, the redundant columns of the N-gram matrix downweight the ridge penalty of each equivalence class in proportion to its size. This may be an undesirable effect since this downweighting is unlikely to improve performance; it instead contributes to overwhelming the ridge penalty. Moreover, this effect will occur for any strictly convex regularizer.

The argument in favor of using $\mathcal{X}$ in place of the N-gram matrix is further bolstered by considering the learning scenario in which the appropriate level of granularity with which strings should be represented is unknown. A canonical example of this is when operating at the character level with natural language instead of the more appropriate word level (that treats every word as a character). Working at an overly coarse level risks destroying relevant signal, so it is desirable to operate at the finest level of granularity. In the parlance of our natural language example, it is easy to show that every equivalence class in the word-level representation maps to *exactly one* equivalence class in the character level representation. The ridge penalty would therefore be penalized approximately in proportion to the length of each word, which can be seen as accidentally introducing an interaction between word length and the frequency counts.

2.4.1 Problem Reformulation

Nonetheless, a variety of optimization problems can also be reformulated so that they are *equivalent* to using the original N-gram matrix. A simple example of such a conversion comes from using ridge regression to model label $y_i \in \mathbb{R}$ based on the i^{th} row of the N-gram matrix X.

We can characterize the set of optimization problems that have an equivalent problem where the N-gram matrix can be replaced with the node matrix. Define a partition $\mathcal{J}$ of the integer set $\{1, \ldots, d\}$ to be a set of m integral intervals $\zeta_k = \{i, \ldots, j\}$ such that $\bigcup_{k=1}^{m} \zeta_k = \{1, \ldots, d\}$ and $\zeta_k \cap \zeta_j = \emptyset$ if $k \neq j$. A function $f : \mathbb{R}^d \to \mathbb{R}^p$ is *permutation invariant* with respect to $\mathcal{J}$ (abbrev. $\mathcal{J}$-PI) if for all $x \in \mathbb{R}^d$, $f(x) = f(\pi[x])$ where $\pi : \mathbb{R}^d \to \mathbb{R}^d$ is any permutation that only permutes indices within the same interval $\zeta_k \in \mathcal{J}$. For our purposes it is important to note that L_p-norms are $\mathcal{J}$-PI as are affine functions $Ax + b$ whenever columns

$A_i = A_j \ \forall i, j \in \zeta_k, \forall \zeta_k \in \mathcal{J}$. It is straightforward to show that if $f, g : \mathbb{R}^d \to \mathbb{R}^p$ are both $\mathcal{J}$-PI and $c : \mathbb{R}^p \to \mathbb{R}^q$ then $f(x) + g(x)$ and $c(f(x))$ are also $\mathcal{J}$-PI.

We prove the following theorem to connect permutation invariance to optimization. It establishes that any convex loss of the form $L(Xw, b)$; e.g. SVM, logistic regression, least squares; added to any L_p norm, e.g. L_2 ridge or L_1 lasso penalty, can be simplified to an equivalent learning problem that uses the node matrix instead of the N-gram matrix.

Theorem 4. *Let $f : \mathbb{R}^d \to \mathbb{R}$ be any convex function that is $\mathcal{J}$-PI where $m = |\mathcal{J}|$. Then there exists a convex function $g : \mathbb{R}^m \to \mathbb{R}$ over m variables such that the problem $\min_{x \in \mathbb{R}^d} f(x)$ is equivalent to $\min_{z \in \mathbb{R}^m} g(z)$. If z^* is optimal for the second problem, then $x_i = z_k^* \ \forall i \in \zeta_k, \ \forall \zeta_k \in \mathcal{J}$ is optimal for the first problem.*

Proof. Suppose that f is $\mathcal{J}$-PI where $\mathcal{J} = \{\zeta_1, \ldots, \zeta_m\}$ and let X^* be the set of minimizers of $\min_{x \in \mathbb{R}^d} f(x)$. If X^* is empty then our proof is trivial, so we assume that X^* is not empty. The central idea behind our proof is that X^* must contain a Cartesian product of permutahedrons (Ziegler, 1995). In particular, given a finite vector $a \in \mathbb{R}^n$, the permutahedron $\mathbb{P}(a) \subset \mathbb{R}^n$ on a is the polyhedron formed by taking the convex hull of all $n!$ n-vectors whose entries are some permutation of the entries of a.

In order to see how this relates to f, let $x \in X^*$ be optimal and let x_{ζ_k} denote the $n_k = |\zeta_k|$ entries in x with indices in ζ_k. Since f is $\mathcal{J}$-PI, it follows that f's value remains unchanged if we permute the x_{ζ_k} arbitrarily. In fact, by definition, if $\hat{x}$ is the vector formed by arbitrarily permuting the entries within each $\zeta_k \in \mathcal{J}$, then $f(x) = f(\hat{x})$ so $\hat{x} \in X^*$ is optimal as well. Assume, without loss of generality, that $\zeta_1 = \{1, \ldots, n_1\}, \zeta_2 = \{n_1 + 1, \ldots, n_1 + n_2\}$ and so on and define

$$\mathcal{Q} = \mathbb{P}(x_{\zeta_1}) \times \mathbb{P}(x_{\zeta_2}) \times \cdots \times \mathbb{P}(x_{\zeta_m}).$$

Our reasoning shows that any $z \in \mathcal{Q}$ is optimal and hence $\mathcal{Q} \subset X^*$.

Now consider the centroid of $\mathcal{Q}$, $\mu \in \mathbb{R}^d$. The centroid of $\mathbb{P}(a)$ for $a \in \mathbb{R}^n$ is simply the n-vector with $\frac{1}{n} \sum_{i=1}^n a_i$ in every entry (Ziegler, 1995). Moreover, since $\mathcal{Q}$ is a Cartesian product of polyhedra, its centroid is given by stacking the centroids of its constituent polyhedra. Let $\eta \in \mathbb{R}^m$ have its entries be $\eta_k = \frac{x_{\zeta_k}^T \mathbf{1}}{n_k}$, i.e. the mean of the elements in x_{ζ_k} and define $V \in \{0, 1\}^{d \times m}$ to be the binary matrix in which column k has ones in indices ζ_k and is all 0 otherwise. It follows that $\mu = V\eta$, and since $\mu \in \mathcal{Q} \subset X^*$, there must be a minimizer of f whose entries are identical in each of the ζ_k.

This reasoning then shows that constrained problem

$$\underset{x \in \mathbb{R}^d}{\text{minimize}} \quad f(x) \quad \text{subject to} \quad x \in \operatorname{col} V. \tag{2.12}$$

is a constrained convex problem (with a linear constraint) and therefore has a minimum that is *lower bounded* by the minimum of our original (unconstrained) problem. By construction of μ, we see that it satisfies the linear constraint and is an optimal point for both problems. It follows, then, that the minimizers of the problem in equation 2.12 are a subset of X^*. Moreover, solving equation 2.12 will always provide a minimizer of the original optimization problem.

We can then replace the subspace constraint by noting that $x \in \operatorname{col} V$ if and only if $x = Vz$ for some $z \in \mathbb{R}^d$. This leads to a problem which is equivalent to the problem in 2.12, namely

$$\underset{z \in \mathbb{R}^m}{\text{minimize.}} \quad f(Vz) \tag{2.13}$$

It follows that we obtain a minimizer of our original problem simply by setting $x = Vz$, i.e. $x_i = z_k$ where $i \in \zeta_k$. Importantly, equation 2.13 is a smaller minimization problem over m variables instead of d variables. We note that this proof is entirely geometric and the details of how problem 2.13 might further be reduced algebraically are problem dependent. □

2.4.2 Holding Out Data

Oftentimes the document corpus is organized into T (possibly overlapping) integral sets $\mathcal{Q}_1, \ldots, \mathcal{Q}_T$ indexing the documents. For instance, splitting documents into training and testing sets yields $T = 2$ index sets, and further subdividing the training set for K-fold crossvalidation introduces $2K$ additional sets (indicating the hold out and training data for each split). In this case we are not interested in multiplying all of $\mathcal{X}$, but only the submatrix whose rows' indices are in the given $\mathcal{Q}_i$. This matrix-vector product can be computed by calling the recursive multiplication algorithm with the topology information in $\mathcal{T}_{\mathcal{C}}(\mathcal{V})$ (derived from the full corpus) and with the submatrix of Φ whose rows' indices are in $\mathcal{Q}_i$. Also note that if only a subset of the documents will ever be used for training, we can ignore any nodes in $\mathcal{T}_{\mathcal{C}}$ that do not appear in the training set since they (should) be ignored by any learning algorithm; we discuss this further in Section 2.5.

2.4.3 Normalization

Mean centering and column normalization can be performed *implictly*, without modifying $\mathcal{X}$, by premultiplying and adding a correction term:

$$\left(\left(\mathcal{X} - \mathbf{1}\mu^T\right)\Sigma\right) w = \mathcal{X}(\Sigma w) - (\mu^T \Sigma w)\mathbf{1}$$

Here μ is a vector of column means and Σ is a diagonal matrix of column normalizing factors. Analogous formulas exist for row centering and normalization.

2.5 Persistent Storage Framework

We use an intermediary data structure, the depth-first pre-processing tree (DFPT), to output the BFMT. The DFPT is computed from a corpus $\mathcal{C}$ and stores the *minimal* information in $\mathcal{T}_\mathcal{C} = (V, E)$ necessary to produce any BFMT and to prune the nodes in V. It can be computed once and used to store $\mathcal{C}$ in a format that is amenable for arbitrary machine learning tasks. Given a new learning problem the DFPT proceeds in two stages: 1) it identifies useful N-grams in V and calculates relevant column normalizations, and 2) it emits a BFMT tailored to that task. Construction of the DFPT, as well as its processing stages, requires $O(n)$ time and memory with respect to the corpus length n.

As suggested by its name, the DFPT stores the topology of $\mathcal{T}_\mathcal{C}$ in DFF, its leaf-document annotations, and if filtering by N-gram length, the edge label length for each internal node of V. Its processing stages are a sequence of of top-down/bottom-up traversals of $\mathcal{T}_\mathcal{C}$ that are individually more sophisticated than those required by our multiplication algorithm, so we opted for the memory savings afforded by the DFF. Indeed, depth($\mathcal{T}_\mathcal{C}$) is bounded by the length of the longest document in $\mathcal{C}$ while the tree width is bounded by the corpus length; the memory savings of the DFF over the BFF are substantial. Importantly, the traversals stream through the DFPT so it is reasonable to operate on it via external memory, e.g. a hard drive, if memory is limited.

2.5.1 Computing the Persistent Storage Representation

Computing the DFPT from $\mathcal{C}$ represents the least memory efficient part of our framework as we first compute a suffix array (SA) [24] from the text and then convert the SA into the DFPT. The process requires $3n\lceil\log_2 n\rceil + n\lceil\log_2(|\Sigma| + N)\rceil$ bits and $O(n)$ time. We emphasize that our framework is completely modular so the DFPT only needs to be computed once. We leave it as an open problem to determine if a more memory efficient algorithm exists that directly computes the DFPT.

Recalling that each leaf of $\mathcal{T}_\mathcal{C}$ is numbered according to the suffix it corresponds to, the SA is a permutation of the integers $[0, \ldots, n)$ that stores the leaves of $\mathcal{T}_\mathcal{C}$ in a *pre-order depth-first traversal*. We use an SA rather than a suffix tree because the former typically requires 4 times less memory than a suffix tree and can also be constructed in $O(n)$ time and memory. We use the implementation of [45], which requires $m = 3n\lceil\log_2 n\rceil + n\lceil\log_2(|\Sigma| + N)\rceil$ bits to construct the SA, where the second term corresponds to a modified copy of $\mathcal{C}$. This was the most memory efficient linear-time suffix array construction algorithm we could find; asymptotically slower but more memory efficient algorithms may be preferable for DNA sequence data.

Converting the suffix array into the DFPT relies on the framework discussed in [29], as it allows us to simulate a *post-order depth-first traversal* of $\mathcal{T}_\mathcal{C}$ using the SA. By carefully managing memory and off-loading unused information to external storage, each step of the conversion requires at most $m - n\lceil\log_2 n\rceil$ bits to be stored in main memory at any time. The *total* memory requirements, including storing the DFPT while it is constructed, never exceed the maximum of $m - n\lceil\log_2 n\rceil$ and $2n\lceil\log_2 n\rceil + (n + |I|)\lceil\log_2 N\rceil$ bits; both are less than m.

2.5.2 Feature Filtering and Normalizations

The first stage of the DFPT's processing determines which nodes in $\mathcal{T}_\mathcal{C}$ should be present in the final BFMT. It also computes any required normalizations, such as the column mean or norm, of the node matrix $\mathcal{X}$ the BFMT represents. We assume that only the internal nodes $I \subset V$ of $\mathcal{T}_\mathcal{C}$ will ever be used; each leaf appears in only a single document and is unlikely to carry useful information. We model the screening process as a sequence of filters that are applied to I: associated with I is a boolean array $b \in \{0, 1\}^{|I|}$ where $b_v = 1$ indicates that node $v \in I$ is useful and $b_v = 1\ \forall v \in I$ initially. Each filter takes as input the DFPT and b, and updates b (in place) with its own criteria. All of our filters are memory efficient and only need to store $|I| + O(\text{depth}(\mathcal{T}_\mathcal{C}))$ bits in memory as the BFMT can reasonably be streamed from slower external storage.

With the exception of the unique document filter, all of the filters listed below run in $O(n)$ time:

- **N-gram length**: removes nodes whose shortest corresponding N-gram is longer than a given threshold.
- **Training set**: removes nodes that do not appear in any documents designated as the training set.
- **Unique document frequency**: removes nodes that do not appear in at least some number of *distinct* documents. We use a memory efficient algorithm which runs in $O(n\alpha^{-1}(n))$ time, where α^{-1} is the inverse Ackermann function ($\alpha^{-1}(10^{80}) = 4$) and is essentially linear-time. A $O(n)$ algorithm [24] is possible, but it requires complicated pre-processing and an additional $n\lceil \log_2 n \rceil$ bits of memory.
- **Strong rules**: given *mean centered* document labels $y \in \mathbb{R}^N$, removes all nodes v for which $|\mathcal{X}_v^T y| < \lambda$ for a threshold λ. This implements the strong rules of [66] and can be applied to a subset of the documents $\mathcal{I}_{\text{tr}} \subset \{1, \ldots, N\}$ (e.g. training data) by mean centering only $y_{\mathcal{I}_{\text{tr}}}$ and setting $y_i = 0$ for all $i \notin \mathcal{I}_{\text{tr}}$. Column normalizations are achieved by checking $\eta_v^{-1}|\mathcal{X}_v^T y| < \lambda$, where η_v^{-1} is the normalization for column v. This filter essentially multiplies $\mathcal{X}^T y$ using the DFPT and the normalization can be computed on the fly (see discussion below).

Once we know which nodes will be used in the BFMT we typically require the column mean $\mu = \frac{1}{N}\mathcal{X}^T\mathbf{1}$ and some kind of normalization η_v^{-1} for each column of $\mathcal{X}$. Noting that all entries of $\mathcal{X}$ are non-negative, the L_1-norm of each column is $\eta = \mathcal{X}^T\mathbf{1}$. Each of these quantities is a matrix-vector multiply that we perform using the DFPT. These quantities can be specialized to training data by setting appropriate entries of the $\mathbf{1}$ vector to 0. We can also compute the L_2-norm of each column of $\mathcal{X}$ or the L_1/L_2-norm of each column of $\mathcal{X} - \mathbf{1}\mu^T$, the mean centered node matrix. These normalizations however require $O(N|I|)$ time and $O(N\text{depth}(\mathcal{T}_\mathcal{C}))$ memory; the space savings of the DFF are critical for the memory bound. These running times are tolerable if performed only once, especially on the short and wide trees that tend to occur with natural language.

2.5.3 Emitting Multiplication Optimized Matrices

The final stage in our pipeline produces the BFMT using the DFPT and filter b. The following lemma follows from the definitions of breadth-first and depth-first traversals and is essential for easy conversion between the two formats:

Lemma 3. *Given a tree $T = (V, E)$, let β be an (ordered) list of the nodes in V in breadth-first order and define δ to be a list of V in depth-first preorder. Define $\beta(d)$ and $\delta(d)$ to be the (sub) lists of β and δ respectively containing only nodes at depth d. Then $\beta(d) = \delta(d)\ \forall d = 1, \ldots,$ depth(T).*

This lemma states that the breadth-first and depth-first preorder traversals list nodes in the same order if we only consider the nodes of a tree at a specific depth. We thus allocate memory for the BFMT by counting the number of nodes with $b_v = 1$ at each depth in the DFPT. The lemma then allows us to copy the relevant nodes in the DFPT into the BFMT skeleton by maintaining a stack of size depth$(\mathcal{T}_\mathcal{C})$ that keeps track of how many nodes have been written to the BFMT at each depth. The depth-first traversal also makes it is easy to determine edges by keeping track of each node's nearest ancestor (in $\mathcal{T}_\mathcal{C}$) that is in the BFMT. The copying process streams through the DFPT and b in a single linear sweep and requires storing the BFMT and $O(\text{depth}(\mathcal{T}_\mathcal{C}))$ bits in memory.

2.6 Experiments

This section provides benchmarks for our multiplication algorithm and applies it to solve several large-scale sentiment analysis tasks. We implemented our framework in C [3] and compiled it used the GCC compiler version 4.4.7 for an x86-64 architecture. Our reference machine uses an Intel Xeon E5-2687W processor with 8 cores running at 3.1 GHz and has 128 Gb of RAM.

2.6.1 Memory and Timing Benchmarks

We evaluate our multiplication algorithm on three kinds of data to investigate its performance in a variety of scenarios: short natural language articles, long technical papers, and DNA sequence data. The first is the BeerAdvocate dataset [42], a corpus of $1,586,088$ beer reviews totalling 1 Gb of plaintext and each consisting of a median of 126 words. The second is a collection of $70,728$ journal articles collected from NCBI [47] with a median length of 6955 words and totaling 3 Gb of plaintext[4]. Our third dataset is derived from the 1000 Genomes Project [51] and it consists of $6,196,151$ biallelic markers, i.e. binary values, along chromosome 1 for 250 people.

[3] Please contact the first author for source code.

[4] This data was graciously made available by the Saccharomyces Genome Database at Stanford.

Our preprocessing framework required at most 3.5 times the memory of the original datasets for the natural language data. The third dataset however presents a worst case scenario for our framework and suffix tree/arrays in general. It requires 185 megabytes to store because of its small alphabet size, yet its suffix array requires $n\lceil \log_2 n \rceil$ bits, i.e. 31 times more memory, and several times this amount to compute. While the DFPT ameliorates this memory usage, it still requires 10 times more memory than the original data and total memory usage went up to 18 gigabytes when computing it from the suffix array.

Figure 2.1 compares the memory requirements of the BFMT to explicitly storing the node and N-gram matrices for all N-grams up to length K that appear in at least 2 documents. We show space requirements for our modified sparse matrix format (MSF) as well as the standard sparse format (SSF), e.g. used in Matlab. The top two graphs correspond to the natural language datasets and have similar patterns: memory usage for the explicit representations rises quickly for up to $K = 7$ and then tapers off as overly long N-grams are unlikely to appear in multiple documents. In all cases the BFMT is superior, requiring approximately 3 times less memory than the MSF and up to 14 times less memory than its floating-point counterpart. While there is some separation between the node matrix and naïve all N-gram matrix, the gap – which is more pronounced in the journal articles – is mitigated by filtering N-grams that do not appear in multiple documents.

The third graph presents a striking difference between the representations: the BFMT requires up to 41 times less memory than the MSF node matrix and over $4,600$ times less memory than the naive N-gram matrix. The floating point counterparts for these matrices accentuate the differences by a factor of 5. Interestingly, the size of the BMFT *decreases* as K increases from 10^3 to 10^4. This occurs because when $K \geq 10^4$, the BFMT behaves as if all N-grams are included so that all entries in the frequency matrix Φ are 0 or 1. When $K \approx 10^3$, most of the entries are bounded by 1, but a few large entries exist and force additional bits to be used for *all* non-zero frequencies in Φ.

Next, Figure 2.2 compares the average multiplication time for the BFMT to ordinary sparse multiplication with the node matrix. The top figure presents results for the BeerAdvocate data; we did not include timings for the journal data since they are essentially the same. We were unable to provide timing results for the node matrix on the DNA data because it quickly exceeded our computer's memory. All trials were performed using a single core for fairness of comparison. The difference between the BFMT and the node matrix closely follows the memory requirement differences. This is to be expected as both multiplication routines make a single pass over the data so running time is proportional to the amount of data that must be scanned. We also

note that the BFMT running time scales gracefully on the DNA data; time increases at a logarithmic rate with respect to K since the x-axis is logarithmic.

2.6.2 Sentiment Analysis Tasks

We applied our framework to sentiment analysis tasks on three large datasets: the BeerAdavocate dataset, a set of $6,396,350$ music reviews from Amazon [43] (4.6 Gb of text), and a set of $7,850,072$ movie reviews also from Amazon [43] (7.4 Gb of text). Each review's sentiment is a value between 1 and 5 (indicating negative or positive) and we tried to predict this sentiment using a ridge regression model on features provided by the node matrix. Each dataset was randomly split into training, validation, and testing sets comprised of $75\%, 12.5\%$, and 12.5% of the total data; all parameters discussed below were selected based on their validation set performance.

We solved the regression by implementing a conjugate gradient solver in C that uses our fast multiplication routine. The ridge parameter λ was tuned on a grid of up to 100 values. We stopped tuning once the validation error increased for 5 consecutive λ values and the procedure typically terminated after trying 60 values. N-grams were pruned by maximum N-gram length and were required to appear in at least 20 distinct documents – we experimented with several document frequency thresholds. We also used the strong rules to select a subset of the features for each λ value and used $\alpha\lambda$ as the threshold; $\alpha = 1$ always gave the best performance. Finally, all columns were mean centered and normalized by their L_2 norm. Our framework computed this normalization in 2.5 minutes for the larger movie dataset. The largest and most time intensive feature set contained 19.4 million features and occurred for $K = 5$ on the movie dataset. It took 26 hours to solve for and evaluate 69 lambda values while running on a *single* core. We were able to effectively run all N-gram trials in parallel.

Table 2.1: Mean Squared Error for Sentiment Analysis

K	Beer	Music	Movies
1	0.286	0.766	0.765
2	0.254	0.481	0.237
3	0.245	0.366	0.140
4	**0.244**	0.333	0.121
5	**0.244**	**0.325**	**0.115**

Table 3.4 summarizes the mean-squared error of our regression model on the testing set. All three datasets benefit from longer N-grams, but we note that the longer datasets seem to benefit more (size increases from left to right). Confounding this potential effect are peculiarities specific to the tasks and specific to BeerAdvocate versus Amazon reviews (recall that the music and movie reviews both come from the same data source). Nonetheless, it is also possible that larger datasets are better equipped to utilize long N-grams: they provide enough examples to counter the variance incurred from estimating coefficients for long, and therefore relatively infrequent, N-grams. It will be interesting to verify this potential effect with more experiments.

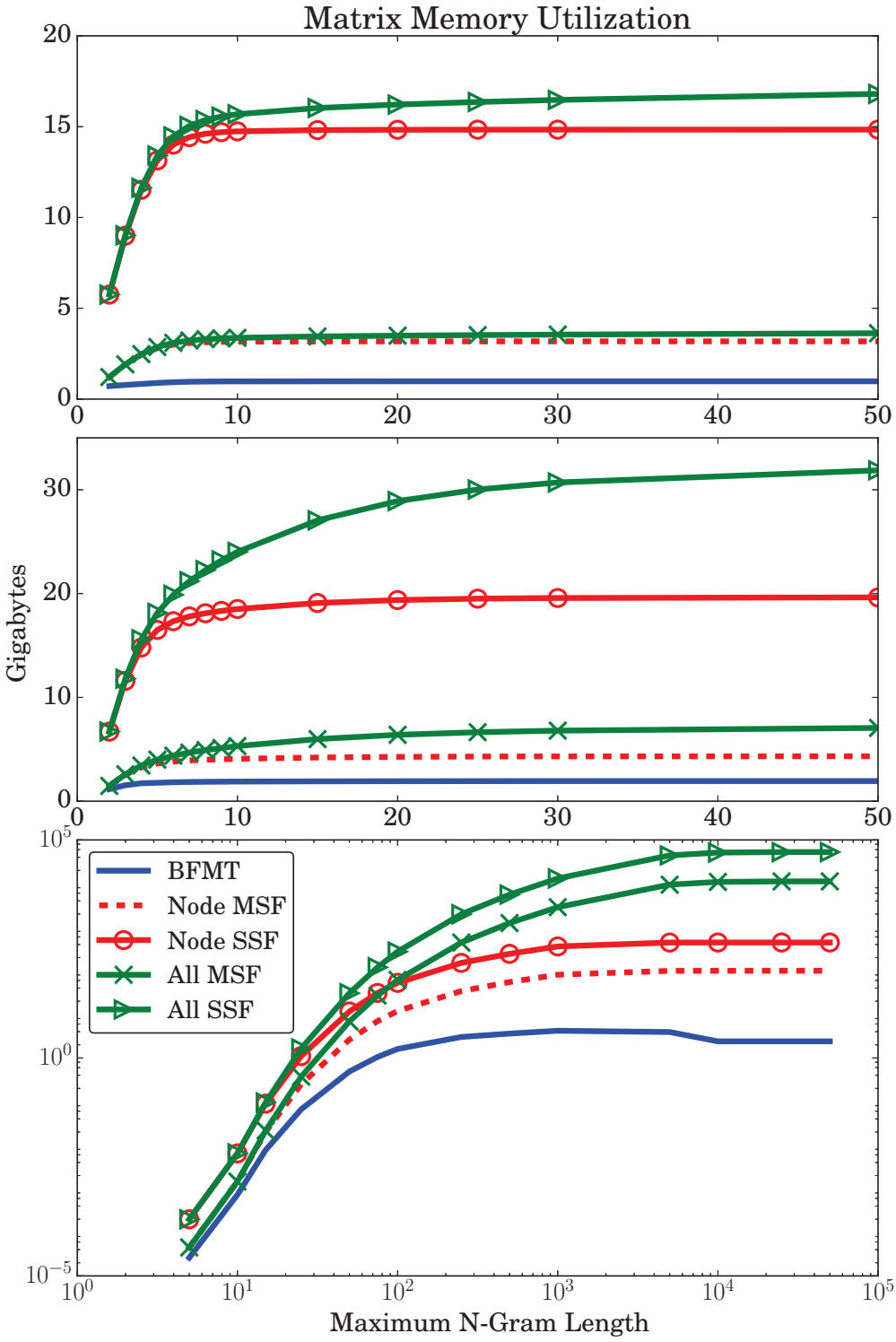


Figure 2.1: Memory utilization for the BFMT, node, and all N-gram matrices as a function of maximum N-gram length K on the BeerAdvocate data (top), journal data (middle) and 1000 genomes data (bottom).

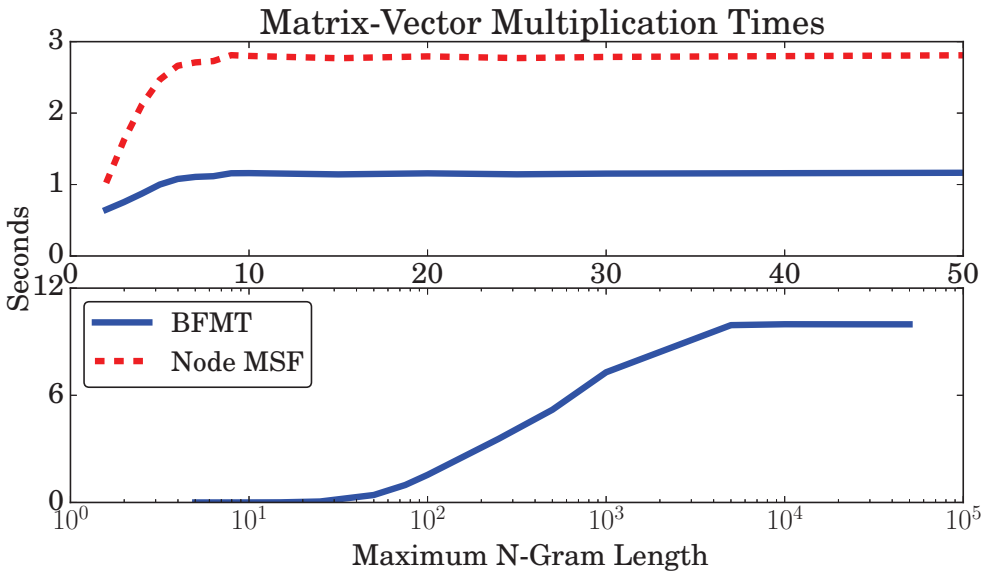


Figure 2.2: Average time to perform one matrix-vector multiply with the BFMT and node matrices as a function of maximum N-gram length K on the BeerAdvocate data (top) and 1000 Genomes Data (bottom). Node matrix times are missing for the latter because it was impractical to store.

Chapter 3

Data Representation and Compression Using Linear-Programming Approximations

We now turn our attention to Dracula and its shallow variant Compressive Feature Learning (CFL); this work focuses on using classical ideas from compression to derive useful feature representations for sequential data such as text. We begin by introducing CFL in Section 3.1 and show how to generalize the criterion to derive Dracula in Section 3.2. Section 3.3 investigates Dracula's computational properties by exploring its fundamental subproblems and linear programming interpretations; it also shows that both Dracula and CFL are NP-Complete. Next, Section 3.4 uses Dracula's polyhedral interpretation to explore the compressed representations it finds as its storage cost model varies. It also discusses how to extract features directly from a compression and how to integrate dictionary structure into the features. Section 3.5 explores two approximation algorithms for CFL that leverage its mathematical structure via the Alternation Directions Method of Multipliers optimization framework. Finally, Section 3.6 provides empirical evidence that CFL and Dracula find representations that are simultaneously useful for learning and compression.

3.1 Compressive Feature Learning

In what follows we will assume a fixed document corpus $\mathcal{C}$. CFL represents document corpus $\mathcal{C}$ by storing a *dictionary* $S \subset \mathcal{C}^*$, a set of N-grams, along with a *pointer set* $P \subset \mathcal{P}_\mathcal{C}$ that only uses dictionary N-grams and losslessly reconstructs each of the documents in $\mathcal{C}$. Importantly, CFL stores the dictionary directly in plaintext. The overall representation is chosen to minimize its total storage cost for a given storage cost model that specifies d_s, the cost of including N-gram $s \in \mathcal{C}^*$ in the dictionary, as well as c_p, the cost of including pointer $p \in \mathcal{P}_C$ in the pointer set.

For a concrete example, see Figure 3.1, which shows three ways of representing a document in terms of a dictionary and pointers. These representations are obtained by setting all dictionary inclusion costs to 1 and using the same pointer storage cost λ for each pointer and varying λ. The two extreme solutions focus on minimizing either the dictionary cost ($\lambda = 0$) or the pointer cost ($\lambda = 8$) solely, while the middle solution ($\lambda = 1$) trades off between minimizing a combination of the two. We are particularly interested in this trade-off: in this simple setting the dictionary and pointer costs pull the solution in opposite directions. Varying λ traces out a regularization path that varies between the two extremes of minimum dictionary cost and minimum pointer cost. Optimizing the former yields a solution that only uses unigrams and has many pointers. On the other hand, the minimal pointer set cost solution stores the entire document as a single dictionary element. Figure 3.2 shows how CFL balances the two costs as λ varies on a larger corpus. It generates a path of solutions that interpolate between the two extremes. This path gives CFL additional flexibility to adapt its solution to the task at hand, something traditional compression schemes cannot do.

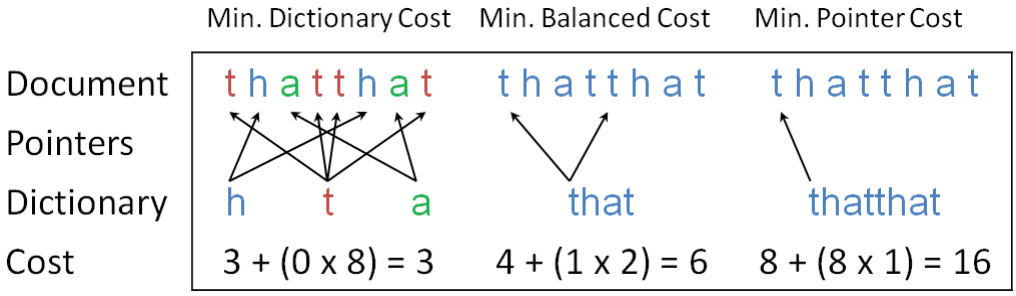


Figure 3.1: Three different CFL solutions on a toy example. Dictionary cost: number of characters in dictionary. Pointer set cost: $\lambda \times$ number of pointers. **Left:** dictionary cost only ($\lambda = 0$). **Right:** expensive pointer cost ($\lambda = 8$). **Center:** balanced dictionary and pointer costs ($\lambda = 1$).

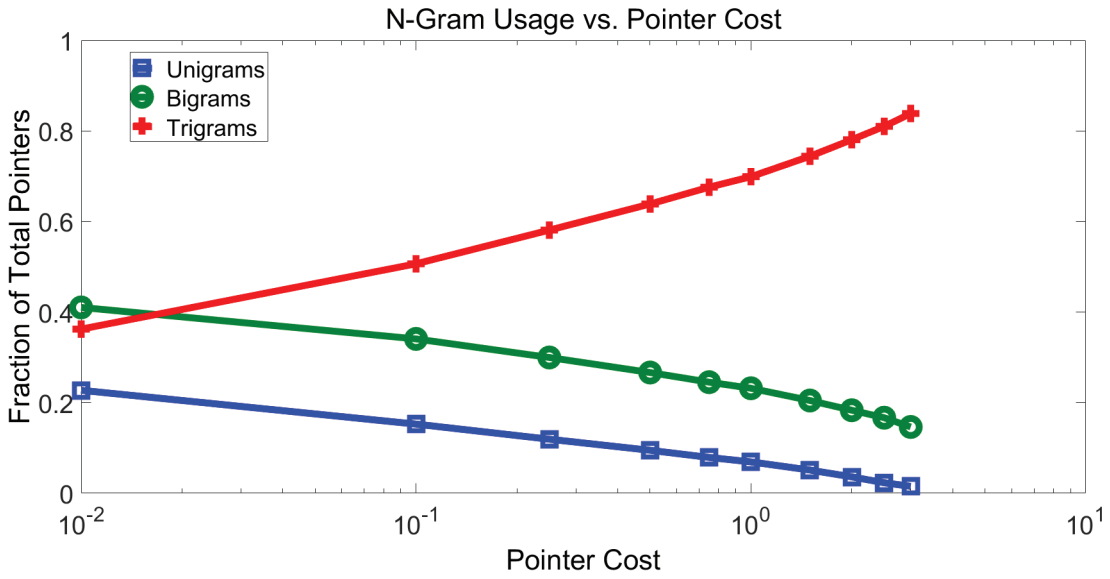


Figure 3.2: Fraction of pointers that correspond to unigrams, bigrams, and trigrams in the compressed representation of the BeerAdvocate dataset. We set the dictionary inclusion cost identically to one and use the same pointer cost for all pointers and allow up to trigrams.

Selecting an optimal CFL representation may be expressed as

$$\underset{S\subset\mathcal{C}^*,P\subset\mathcal{P}_\mathcal{C}}{\text{minimize}} \sum_{p\in P} c_p + \sum_{s\in S} d_s \quad \text{subject to} \quad P \text{ reconstructs } D_k \; \forall D_k \in \mathcal{C};\; P \text{ only uses } s \in S. \tag{3.1}$$

We are particularly interested in the setting where the pointer cost is the same for all pointers, i.e., $c_p = \lambda$, since it forces the dictionary cost to oppose the pointer set cost. Optimizing the former yields a solution that only uses unigrams and has many pointers. On the other hand, the minimal pointer set cost solution stores the entire document as a single dictionary element. Figure 3.2 shows how CFL balances the two costs as λ varies: it generates a path of solutions that interpolate between the two extremes. This path gives CFL additional flexibility to adapt its solution to the task at hand, something traditional compression schemes cannot do.

This optimization problem naturally decomposes into subproblems by observing that when the dictionary is fixed, selecting the optimal pointer set decouples into $|\mathcal{C}|$ separate problems of optimally reconstructing each corpus document. We thus define the *reconstruction module* for document $D_k \in \mathcal{C}$, which takes as input a dictionary S and outputs the minimum cost of reconstructing D_k with pointers that only use strings in S. Note that specific pointers and dictionary strings can be disallowed by setting their respective costs to ∞. For

example setting $d_s = \infty$ for all $s \in \mathcal{C}^*$ longer than a certain length limits the size of dictionary N-grams. Of course, in practice, any variables with infinite costs are simply disregarded.

The reconstruction module can be expressed as a BLP by associating with every pointer $p \in \mathcal{P}(D_k)$ a binary indicator variable $w_p \in \{0,1\}$ whereby $w_p = 1$ indicates that p is included in the optimal pointer set for D_k. We similarly use binary variables $t_s \in \{0,1\}$ to indicate that $s \in \mathcal{C}^*$ is included in the dictionary. Since there is a one-to-one correspondence between pointer sets (dictionaries) and $w \in \{0,1\}^{|\mathcal{P}(D_k)|}$ ($t \in \{0,1\}^{|\mathcal{C}^*|}$), the vector storing the w_p (t_s), we will directly refer to these vectors as pointer sets (dictionaries). Lossless reconstruction is encoded by the constraint $X^{D_k} w \geq \mathbf{1}$ where $X^{D_k} \in \{0,1\}^{|D_k| \times |\mathcal{P}(D_k)|}$ is a binary matrix indicating the indices of D_k that each pointer can reconstruct. In particular, for every $p = (D_k, l, z) \in \mathcal{P}(D_k)$, column $X_p^{D_k}$ is all zeros except for a contiguous sequence of 1's in indices $l, \ldots, l + |z| - 1$. Control of which pointers may be used (based on the dictionary) is achieved by the constraint $w \leq V^{D_k} t$ where $V^{D_k} \in \{0,1\}^{|\mathcal{P}(D_k)| \times |\mathcal{C}^*|}$ contains a row for every pointer indicating the string it uses. In particular, for every $p = (D_k, l, z)$, $V_{p,z}^{D_k} = 1$ is the only non-zero entry in the row pertaining to p. The BLP may now be expressed as

$$R_{D_k}(t; c) = \underset{w \in \{0,1\}^{|\mathcal{P}(D_k)|}}{\text{minimize}} \quad \sum_{p \in \mathcal{P}(D_k)} w_p c_p \quad \text{subject to} \quad X^{D_k} w \geq \mathbf{1};\ w \leq V^{D_k} t. \tag{3.2}$$

The optimization problem corresponding to an optimal CFL representation may now be written as a BLP by sharing the dictionary variable t among the reconstruction modules for all documents in $\mathcal{C}$:

$$\underset{t \in \{0,1\}^{|\mathcal{C}^*|}}{\text{minimize}} \sum_{D_k \in \mathcal{C}} R_{D_k}(t, c) + \sum_{s \in \mathcal{C}^*} t_s d_s \tag{3.3}$$

3.2 Adding Depth with DRaCULA

The simplicity of CFL's dictionary storage scheme is a fundamental shortcoming that is demonstrated by the string $aa \ldots a$ consisting of the character a replicated 2^{2n} times. Let the cost of using any pointer be $c_p = 1$ and the cost of storing any dictionary N-gram be its length, i.e. $d_s = |s|$. The best CFL can do is to store a single dictionary element of length 2^n and repeat it 2^n times, incurring a total storage cost of 2^{n+1}. In contrast, a "deep" compression scheme that recursively compresses its own dictionary by allowing dictionary strings to be represented using pointers attains *exponential* space savings relative to CFL. In particular, the

deep scheme constructs dictionary strings of length $2, 4, \ldots, 2^{2n-1}$ recursively and incurs a total storage cost of $4n$ [1].

Dracula extends CFL precisely in this hierarchical manner by allowing dictionary strings to be expressed as a combination of characters and pointers from shorter dictionary strings. CFL thus corresponds to a shallow special case of Dracula which only uses characters to reconstruct dictionary N-grams. This depth allows Dracula to leverage similarities among the dictionary strings to obtain further compression of the data. It also establishes a hierarchy among dictionary strings that allows us to interpret Dracula's representations as a directed acyclic graph (DAG) that makes precise the notion of representation depth.

Formally, a Dracula compression (compression for brevity) of corpus $\mathcal{C}$ is a triple $\mathfrak{D} = (S \subset \mathcal{C}^*, P \subset \mathcal{P}_\mathcal{C}, \hat{P} \subset \mathcal{P})$ consisting of dictionary, a pointer set P that reconstructs the documents in $\mathcal{C}$, and a pointer set $\hat{P}$ that reconstructs every dictionary string in S. As with CFL, any pointers in P may only use strings in S. However, a pointer $p \in \hat{P}$ reconstructing a dictionary string $s \in \mathcal{C}^*$ is valid if it uses a unigram (irrespective of whether the unigram is in S) or a *proper substring* of s that is in S. This is necessary because unigrams take on the special role of characters for dictionary strings. They are the atomic units of any dictionary, so the character set Σ is assumed to be globally known for dictionary reconstruction. In contrast, document pointers are not allowed to use characters and may only use a unigram if it is present in S; this ensures that all strings used to reconstruct the corpus are included in the dictionary for use as features.

Finding an optimal Dracula representation may also be expressed as a BLP through simple modifications of CFL's objective function. In essence, the potential dictionary strings in $\mathcal{C}^*$ are treated like documents that only need to be reconstructed if they are used by some pointer. We extend the storage cost model to specify costs c_p for all pointers $p \in \mathcal{P}_\mathcal{C}$ used for document reconstruction as well as costs $\hat{c}_p$ for all pointers $p \in \mathcal{P}$ used for dictionary reconstruction. In keeping with the aforementioned restrictions we assume that $\hat{c}_p = \infty$ if $p = (s, 1, s)$ illegally tries to use s to reconstruct s and s is not a unigram. The dictionary cost d_s is now interpreted as the "overhead" cost of including $s \in \mathcal{C}^*$ in the dictionary without regard to how it is reconstructed; CFL uses the d_s to also encode the cost of storing s in plaintext (e.g. reconstructing it only with characters). Finally, we introduce dictionary reconstruction modules as analogs to the (document) reconstruction modules for dictionary strings: the reconstruction module for $s \in \mathcal{C}^*$ takes as input a dictionary

[1] Note that the recursive model is allowed to use pointers in the dictionary and therefore selects from a larger pointer set than CFL. Care must be taken to ensure that the comparison is fair since the "size" of a compression is determined by the storage cost model and we could "cheat" by setting all dictionary pointer costs to 0. Setting all pointer costs to 1 ensures fairness.

and outputs the cheapest valid reconstruction of s if s needs to be reconstructed. This can be written as the BLP

$$\hat{R}_s(t;\hat{c}) = \underset{w\in\{0,1\}^{|\mathcal{P}(s)|}}{\text{minimize}} \quad \sum_{p\in\mathcal{P}(s)} w_p\hat{c}_p \quad \text{subject to} \quad X^s w \geq t_s\mathbf{1};\; w \leq \hat{V}^s t. \tag{3.4}$$

Here X^s is analogously defined as in equation (3.4) and $\hat{V}^s$ is analogous to V^s in equation (3.4) except that it does not contain any rows for unigram pointers. With this setup in mind, the optimization problem corresponding to an optimal Dracula representation may be written as the BLP

$$\underset{t\in\{0,1\}^{|\mathcal{C}^*|}}{\text{minimize}} \sum_{D_k\in\mathcal{C}} R_{D_k}(t,c) + \sum_{s\in\mathcal{C}^*}\Big[t_s d_s + \hat{R}_s(t;\hat{c})\Big] \tag{3.5}$$

Finally, any compression can be interpreted graphically as, and is equivalent to, a DAG whose vertices correspond to members of Σ, S, or $\mathcal{C}$ and whose labeled edge set is determined by the pointers: for every $(s,l,z) \in P$ or $\hat{P}$ there is a directed edge from z to s with label l. Note that $\mathfrak{D}$ defines a multi-graph since there may be multiple edges between nodes. Figure 3.3 shows the graph corresponding to a simple compression. As this graph encodes all of the information stored by $\mathfrak{D}$, and vice versa, we will at times treat $\mathfrak{D}$ directly as a graph. Since $\mathfrak{D}$ has no cycles, we can organize its vertices into layers akin to those formed by deep neural networks and with connections determined by the pointer set: layer 0 consists only of characters (i.e. there is a node for every character in Σ), layer 1 consists of all dictionary N-grams constructed solely from characters, higher levels pertain to longer dictionary N-grams, and the highest level consists of the document corpus $\mathcal{C}$. While there are multiple ways to organize the intermediate layers, a simple stratification is obtained by placing $s \in S$ into layer i only if $\hat{P}(s)$ uses a string in layer $i-1$ and no strings in layers $i+1,\ldots$. We note that our architecture differs from most conventional deep learning architectures which tend to focus on pairwise layer connections – we allow arbitrary connections to higher layers.

3.3 Computational Properties

This section focuses on the computational properties of CFL and Dracula. We examine the algorithmic structure of the reconstruction modules and give simple and efficient algorithms for the inverse operations of encoding and decoding a document once the dictionary is known. We also show that simultaneously finding an optimal dictionary and document compressions is NP-Complete.

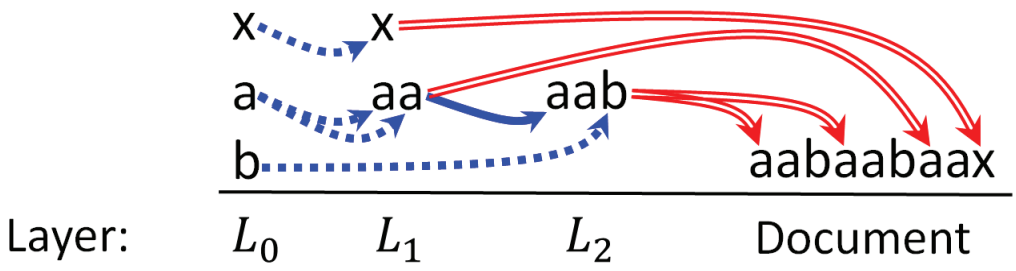


Figure 3.3: Compression of "aabaabaax" using a 3-layered dictionary. Layer 0 consists of characters; layers 1 and 2 are dictionary n-grams. There are three kinds of pointers: character to dictionary N-gram (dashed blue lines), dictionary N-gram to (longer) dictionary N-gram (solid blue line), and dictionary N-gram to document (double red lines).

Decoding An efficient algorithm for decoding was given while motivating the CFL objective: for every $(s, l) \in U$we reconstruct a document by placing s in location l. This simple procedure runs in $O(n)$ time when strings do not overlap and requires $O(Kn)$ when strings overlap and at most n pointers are specified for the document. This procedure easily extends to reconstructing the dictionary by operating in a bottom up fashion from the shortest dictionary elements to the longest ones.

3.3.1 Reconstruction Modules

The reconstruction modules $R_{D_k}/\hat{R}_s$ are the basic building blocks of Dracula; when t is fixed solving (3.5) is tantamount to solving the reconstruction modules *separately*. These simple BLPs have a number of properties that result in computational savings because of the structure of the constraint matrix X^{D_k}/X^s. In order to simplify notation we define

$$T_s(t, v; b, V) = \underset{w \in \{0,1\}^{|\mathcal{P}(s)|}}{\text{minimize}} \quad \sum_{p \in \mathcal{P}(s)} w_p b_p \quad \text{subject to} \quad X^s w \geq v\mathbf{1};\ w \leq Vt. \tag{3.6}$$

Using $T_{D_k}(t, 1; c, V^{D_k}) = R_{D_k}(t; c)$ and $T_s(t, t_s; \hat{c}, \hat{V}^s) = \hat{R}_s(t; \hat{c})$ results in the document or dictionary reconstruction modules. Now note that every column in X^s is all zeros except for a contiguous sequence of ones so that X^s is an *interval matrix* and therefore totally unimodular (TUM). Define T_s^c to be the LP relaxation of T_s obtained by replacing the integrality constraints:

$$T_s^c(t, v; b, V) = \underset{w \in [0,1]^{|\mathcal{P}(s)|}}{\text{minimize}} \quad \sum_{p \in \mathcal{P}(s)} w_p b_p \quad \text{subject to} \quad X^s w \geq v\mathbf{1};\ w \leq Vt. \tag{3.7}$$

Aside from X, the remaining constraints on w are bound constraints. It follows from [6] that T_s^c is an LP over a integral polyhedron so we may conclude that

Proposition 1. *If the arguments t, v are integral, then all basic solutions of $T_s^c(t, v; b, V)$ are binary.*

Indeed, encoding a document D of length n in terms of a terms of dictionary $\mathcal{C}^*$ can be performed in $O(Kn)$ time using dynamic programming where K is the maximum N-gram length used in the dictionary. We assume that $\mathcal{C}^*$ contains all N-grams in D; we can disallow certain N-grams from being used by setting their pointers' costs to ∞. Define $F(i)$ to be the minimal cost of encoding D up to, but not past, position i so that $F(n)$ is the cost of the optimal encoding. The algorithm finds $F(n)$ by computing $F(i)$ from $i = 1$ up to n using the previous values of $F(j)$ for $1 \leq j < i$ to find $F(i)$ efficiently. In order to formalize this procedure, let $d(i, k)$ be cost of the the pointer that *starts* at position $i - k + 1$ and *ends* at location i (whose substring is of length k). Using the convention $d(i, k) = \infty$ if $k > i$ or if that pointer's substring is not in our dictionary, we can write

$$F(i) = \min_{k=1,\ldots,K} \left[d(i,k) + \min_{j=1,\ldots,k} F(i-j) \right]$$

As such, $F(i)$ can be found in $O(K)$ time given $F(0), \ldots, F(i-1)$ by noting that the inner minimization requires $O(1)$ operations for each value of k because of nesting. We can also find the set of pointers that optimally encode D by storing, for each i, the values of k and j that achieve $F(i)$ and working backwards once we compute $F(n)$. The overall running time of our procedure is therefore $O(Kn)$.

Our second property reformulates T_s^c by transforming the constraint matrix X^s into a simpler form. The resulting matrix has at most 2 non-zero entries per column instead of up to $|s|$ non-zero entries per column in X^s. This form is more efficient to work with when solving the LP and it shows that T_s^c is equivalent to a min-cost flow problem over an appropriately defined graph. Define $Q \in \{0, \pm 1\}^{|s| \times |s|}$ be the full rank lower triangular matrix with entries $Q^s_{ii} = -Q^s_{(i+1)i} = 1$ and 0 elsewhere (and $Q^s_{|s||s|} = 1$). The interval structure of X^s implies that column i of $Z^s = Q^s X^s$ is all zeros except for $Z^s_{ij} = -Z^s_{ik} = 1$ where j is the first row in which $X^s_{ij} = 1$ and $k > j$ is the first row in which $X^s_{ik} = 0$ after the sequences of ones (if such a k exists). By introducing non-negative slack variables for the $X^s w \geq v\mathbf{1}$ constraint, i.e. writing $X^s w = v\mathbf{1} + \xi$, and

noting that $Q^s\mathbf{1} = \mathbf{e}_1$, where $\mathbf{e}_1$ is all zeros except for a 1 as its first entry, we arrive at:

$$\begin{aligned} T_s^c(t, v; b, V) = \underset{w,\xi}{\text{minimize}} \quad & \sum_{p \in \mathcal{P}(s)} w_p b_p \\ \text{subject to} \quad & Z^s w - Q^s \xi = v\mathbf{e}_1, \\ & \mathbf{0} \leq w \leq Vt,\ \mathbf{0} \leq \xi. \end{aligned} \tag{3.8}$$

The matrix $\Psi = [Z^s | -Q^s]$ has special structure since every column has at most one 1 and at most one -1. This allows us to interpret Ψ as the incidence matrix of a directed graph if we add source and sink nodes with which to fill all columns out so that they have exactly one 1 and one -1. T_s^c may then be interpreted as a min-cost flow problem.

3.3.2 NP-Completeness

Thus far we have shown that the encoding/decoding procedures can be solved efficiently. Similarly, if the pointer sets are known for each document or dictionary string then it is easy to find the corresponding dictionary t by checking which strings are used (in linear time relative to the number of pointers). One would hope that the easiness of Dracula's subproblems leads to an easy overall learning problem. However, learning the dictionary and pointer sets simultaneously makes this problem hard: CFL and Dracula are both NP-Complete. In fact, a simpler version of CFL known as the *External Macro Data Compression* (EMDC) problem is discussed in [22] and shown to be NP-Complete via reduction from the Vertex Cover problem [22]. EMDC represents a compressed document as an array in which each element is either a unigram stored in plaintext or a pointer indicating which dictionary string it should be replaced by. This scheme can be cast as a special case of CFL by letting the dictionary cost for unigrams be 0 and setting the pointer cost for all unigram pointers to be the cost of storing a unigram in plaintext. All other pointer costs are set to the same value λ. Finally, since we have shown that CFL and Dracula can be expressed as binary linear program, we conclude that CFL and hence its generalization Dracula are NP-Complete.

3.3.3 Problem Relaxation and Polyhedral Refinement

We now consider solving Dracula approximately via its LP relaxation. This is obtained by replacing all binary constraints in equations (3.2),(3.4),(3.5) with interval constraints $[0, 1]$. We let $\mathcal{Q}_C$ denote this LP's constraint

polyhedron and note that it is a subset of the unit hypercube. Importantly, we may also interpret the original problem in equation (3.5) as an LP over a polyhedron $\mathcal{Q}$ whose vertices are always binary and hence always has binary basic solutions. Here $\mathcal{Q}^2$ is the convex hull of all (binary) Dracula solutions and $\mathcal{Q} \subset \mathcal{Q}_C$; all valid Dracula solutions may be obtained from the linear relaxation. In fact, the Chvátal-Gomory theorem [58] shows that we may "prune" $\mathcal{Q}_C$ into $\mathcal{Q}$ by adding additional constraints. We describe additional constraints below that leverage insights from suffix trees to prune $\mathcal{Q}_C$ into a tighter approximation $\mathcal{Q}'_C \subset \mathcal{Q}_C$ of $\mathcal{Q}$. Remarkably, when applied to natural language data, these constraints allowed Gurobi [23] to quickly find *optimal binary solutions*. While we did not use these binary solutions in our learning experiments, they warrant further investigation.

As the pointer and dictionary costs vary, the resulting problems will vary in difficulty as measured by the gap between the objectives of the LP and binary solutions. When the costs force either t or the w^{D_k}/w^s to be binary, our earlier reasoning shows that the entire solution will lie on a binary vertex of $\mathcal{Q}_C$ that is necessarily optimal for the corresponding BLP and the gap will be 0. This reasoning also shows how to round any continuous solution into a binary one by leveraging the easiness of the individual subproblems. First set all non-zero entries in t to 1, then reconstruct the documents and dictionary using this dictionary to yield binary pointers, and finally find the minimum cost dictionary based on which strings are used in the pointers.

We now show how to tighten Dracula's LP relaxation by adding additional constraints to Q_C to shrink it closer to Q based on equivalence classes. We will say that equivalence class $\varepsilon \subset \mathcal{C}^*$ appears at a location if any (i.e. all) of its members appear at that location. We prove the following theorem below. This theorem verifies common sense and implies that, when the pointer costs do not favor any particular string in ε, adding the constraint $\sum_{s\in\varepsilon} t_s \leq 1$ to the LP relaxation to tighten $\mathcal{Q}_C$ will not remove any binary solutions.

Theorem 1. *Let Ω denote the set of all equivalence classes in corpus $\mathcal{C}$ and suppose that all costs are non-negative and $\forall \varepsilon \in \Omega, \forall z \in \mathcal{C}^*, \forall s, x \in \varepsilon$, the dictionary costs $d_s = d_x$ are equal, the pointer costs $c_p^z = c_q^z$ ($\hat{c}_p^z = \hat{c}_q^z$) are equal when $p = (l, s)$ and $q = (l, x)$, and $c_p^s = c_q^x$ ($\hat{c}_p^s = \hat{c}_q^x$) whenever pointers $p = q = (l, h)$ refer to the same location and use the same string (or character) h. Then there is an optimal compression $\mathfrak{D} = (S, P\hat{P})$ in which S contains* at most *one member of ε.*

Proof. Suppose that the conditions for theorem 1 hold, let ε be an equivalence class, let $\mathfrak{D} = (S, P, \hat{P})$ be an optimal compression, and suppose for the sake of contradiction that $s_1, s_2 \in \varepsilon$ are both included in the

[2]Note that unlike $\mathcal{Q}_C$, this polyhedron is likely to be difficult to describe succinctly unless $P = NP$.

optimal dictionary. Without loss of generality we assume that $|s_1| < |s_2|$. Consider first document pointer p which uses s_1 for document D_k. By assumption there is another pointer q which uses s_2 in the same location and $c_p^{D_k} = c_q^{D_k}$ so we are indifferent in our choice. We thereby may replace all document pointers that use s_1 with equivalent ones that use s_2 without changing the objective value.

Consider next the usage of s_1 to construct higher order dictionary elements. We must be careful here since if some dictionary element s_3 is in the optimal dictionary S and can be expressed as $s_3 = zs_1$ for some string z then we may not use s_2 in place of s_1 since it would lead to a different dictionary string. The key step here is to realize that s_3 must belong to the same equivalence class as string zs_2 and we can use zs_2 in place of s_3 in all documents. If s_3 is itself used to construct higher order dictionary elements, we can apply the same argument for s_2 to zs_2 in an inductive manner. Eventually, since our text is finite, we will reach the highest order strings in the dictionary, none of whose equivalence class peers construct any other dictionary n-grams. Our earlier argument shows that we can simply take the longest of the highest order n-grams that belong to the same equivalence class. Going back to s_3, we note that our assumptions imply that the cost of constructing zs_2 is identical to the cost of constructing s_3 so we may safely replace s_3 with zs_2. The only remaining place where s_1 may be used now is to construct s_2. However, our assumptions imply that the cost of constructing s_1 "in place" when constructing s_2 is the same. By eliminating s_1 we therefore never can do worse, and we may strictly improve the objective if $t_{s_1} > 0$ or s_1 is used to construct s_2 and its pointer cost is non-zero. □

3.4 Learning with Compressed Features

This section explores the feature representations and compressions that can be obtained from Dracula. Central to our discussion is the observation of Section 3.3.3 that all compressions obtained from Dracula are the vertices of a polyhedron. Each of these vertices can be obtained as the optimal compression for an appropriate storage cost model[3], so we take a dual perspective in which we vary the storage costs to characterize which vertices exist and how they relate to one another. The first part of this section shows how to "walk" around the surface of Dracula's polyhedron and it highlights some "landmark" compressions that are encountered, including ones that lead to classical bag-of-N-grams features. Our discussion applies to both, the binary and relaxed, versions of Dracula since the former can viewed as an LP over a polyhedron $\mathcal{Q}$ with only binary

[3]The storage costs pertaining to each vertex form a polyhedral cone, see [73] for details.

vertices. The second part of this section shows how to incorporate dictionary structure into features via a dictionary diffusion process.

We derive features from a compression in a bag-of-N-grams (BoN) manner by counting the number of pointers that use each dictionary string or character. It will be useful to explicitly distinguish between strings and characters when computing our representations and we will use squares brackets to denote the character inside a unigram, e.g. $[c]$. Recall that given a compression $\mathfrak{D} = (S, P, \hat{P})$, a unigram pointer in P (used to reconstruct a document) is interpreted as a string whereas a unigram pointer in $\hat{P}$ is interpreted as a character. We refer to any $z \in S \cup \Sigma$ as a feature and associate with every document $D_k \in \mathcal{C}$ or dictionary string $s \in S$ a BoN feature vector $x^{D_k}, x^s \in \mathbb{Z}_+^{|S|+|\Sigma|}$, respectively. Entry $x_z^{D_k}$ counts the number of pointers that use z to reconstruct D_k, i.e. $x_z^{D_k} = |\{p \in P(s) | \, p = (D_k, l, z)\}|$, and will necessarily have $x_z^{D_k} = 0$ for all $z \in \Sigma$. Dictionary strings are treated analogously with the caveat that if $p = (s, l, z) \in \hat{P}$ uses a unigram, p counts towards the character entry $x_{[z]}^{D_k}$, not $x_z^{D_k}$.

3.4.1 Dracula's Solution Path

Exploring Dracula's compressions is tantamount to varying the dictionary and pointer costs supplied to Dracula. When these costs can be expressed as continuous functions of a parameter $\lambda \in [0, 1]$, i.e. $\forall s \in \mathcal{C}^*, p \in \mathcal{P}_\mathcal{C}, \hat{p} \in \mathcal{P}$ the cost functions $d_s(\lambda), c_p(\lambda), \hat{c}_{\hat{p}}(\lambda)$ are continuous, the optimal solution sets vary in a predictable manner around the surface of Dracula's constraint polyhedron $\mathcal{Q}$ or the polyhedron of its relaxation $\mathcal{Q}_C$. We use $\mathscr{F}(Q)$ to denote the set of faces of polyhedron Q (including Q), and take the dimension of a face to be the dimension of its affine hull. The following theorem establishes the behavior of the solutions of linear programs as well binary linear programs (when viewed as sufficiently constrained linear programs) when the problem's cost function varies continuously. We defer its proof to the end of this section.

Theorem 2. *Let $Q \subset \mathbb{R}^d$ be a bounded polyhedron with nonempty interior and $b : [0, 1] \to \mathbb{R}^d$ a continuous function. Then for some $N \in \mathbb{Z}_+ \cup \{\infty\}$ there exists a countable partition $\Gamma = \{\gamma_i\}_{i=0}^N$ of $[0, 1]$ with corresponding faces $F_i \in \mathscr{F}(Q)$ satisfying $F_i \neq F_{i+1}$ and $F_i \cap F_{i+1} \neq \emptyset$. For all $\alpha \in \gamma_i$, the solution set of the LP constrained by Q and using cost vector $b(\alpha)$ is $F_i = \arg\min_{x \in Q} x^T b(\alpha)$. Moreover, F_i never has the same dimension as F_{i+1} and the boundary between γ_i, γ_{i+1} is $)[$ iff $\dim F_i < \dim F_{i+1}$ and $]($ otherwise.*

This theorem generalizes the notion of a continuous solution path typically seen in the context of regularization (e.g. the Lasso) to the LP setting where unique solutions are piecewise constant and transitions occur

by going through values of λ for which the solution set is not unique. For instance, suppose that vertex v_0 is uniquely optimal for some $\lambda_0 \in [0, 1)$, another vertex v_1 is uniquely optimal for a $\lambda_0 < \lambda_1 \leq 1$, and no other vertices are optimal in (λ_0, λ_1). Then Theorem 2 shows that v_0 and v_1 must be connected by a face (typically an edge) and there must be some $\lambda \in (\lambda_0, \lambda_1)$ for which this face is optimal. As such, varying Dracula's cost function continuously ensures that the solution set for the binary or relaxed problem will not suddenly "jump" from one vertex to the next; it must go through an intermediary connecting face. This behavior is depicted in Figure 3.4 on a nonlinear projection of Dracula's constraint polyhedron for the string "xaxabxabxacxac".

It is worthwhile to note that determining the exact value of λ for which the face connecting v_0 and v_1 is optimal is unrealistic in practice, so transitions may appear abrupt. While it is possible to smooth this behavior by adding a strongly convex term to the objective (e.g. an L_2 penalty), the important insight provided by this theorem is that the trajectory of the solution path depends entirely on the *combinatorial structure* of $\mathcal{Q}$ or $\mathcal{Q}_C$. This structure is characterized by the face lattice[4] of the polyhedron and it shows which vertices are connected via edges, 2-faces, ..., facets. It limits, for example, the set of vertices reachable from v_0 when the costs vary continuously and ensure that transitions take place only along edges [5]. This predictable behavior is desirable when fine tuning the compression for a learning task, akin to how one might tune the regularization parameter of a Lasso, and it is not possible to show in general for non-convex functions.

Proof of Path Theorem

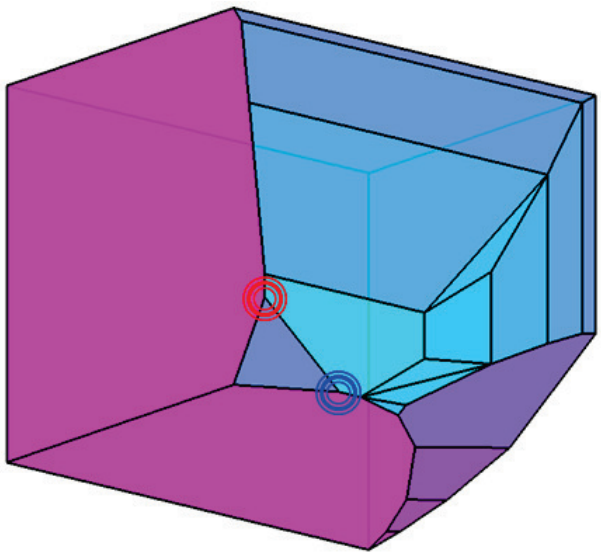
The fundamental theorem of linear programming states that for any $c \in \mathbb{R}^d$, $S(c, Q) \equiv \arg\min_{x \in Q} x^T c(\alpha) \in \mathscr{F}(Q)$ since Q has non-empty interior and is therefore non-empty. We will use a construction known as the normal fan of Q, denoted by $\mathcal{N}(Q)$, that partitions $\mathbb{R}^d$ into a finite set of polyhedral cones pertaining to (linear) objectives for which each face in $\mathscr{F}(Q)$ is the solution set. We begin with some helpful definitions.

A *partition* $P \subset 2^X$ of a set X is any collection of sets satisfying $\bigcup_{p \in P} p = X$ and $\forall p, q \in P\ p \neq q$ implies $p \cap q = \emptyset$. The *relative interior* of a convex set $X \subset \mathbb{R}^d$, denoted by relint X, is the interior of X with respect to its affine hull. Formally, relint $X = \{x \in X \mid \exists \varepsilon > 0, B(x, \varepsilon) \cap \text{aff} X \subset X\}$. The following definition is taken from [40]: A *fan* is a finite set of nonempty polyhedral convex cones in $\mathbb{R}^d$, $\mathcal{N} = \{N_1, N_2, \ldots, N_m\}$, satisfying:

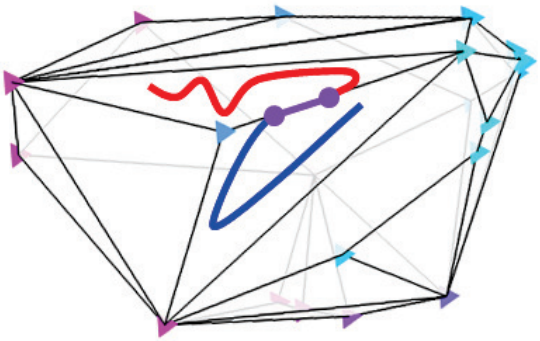
1. any nonempty face of any cone in $\mathcal{N}$ is also in $\mathcal{N}$,

[4] We leave it as an open problem to analytically characterize Dracula's face lattice.

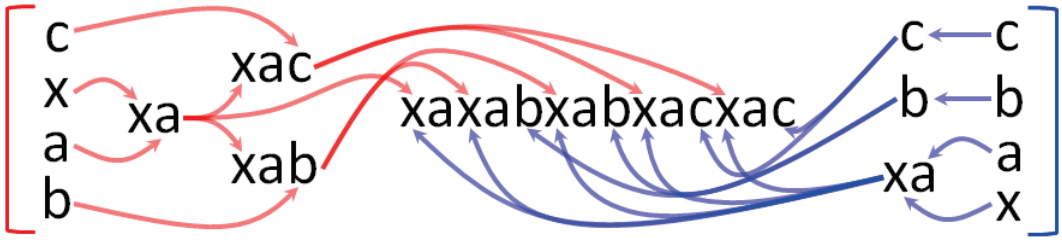
[5] Restricting transitions only to edges is possible with probability 1 by adding a small amount of Gaussian noise to c.



(a)



(b)



(c)

Figure 3.4: Part (a) shows a nonlinear projection of a subset of Dracula's constraint polyhedron $\mathcal{Q}$ in which every vertex corresponds to a distinct compression of "xaxabxabxacxac". Part (b) is the projection's polar; its faces delineate the (linear) costs for which each vertex in (a) is optimal. The red/ purple/ blue line in (b) demonstrates a continuous family of costs. All red (blue) costs are uniquely minimized by the vertex in (a) highlighted in red (blue), respectively; (c) shows the corresponding compressions. Purple costs lie on the edge between the faces containing the red and blue lines and are minimized by any convex combination of the vertices highlighted in (a).

2. any nonempty intersection of any two cones in $\mathcal{N}$ is a face of both cones.

This definition leads to the following lemma, which is adapted from [40]:

Lemma 1. *Let $\mathcal{N}$ be a fan in $\mathbb{R}^d$ and $S = \bigcup_{N\in\mathcal{N}} N$ the union of its cones.*

1. *If two cones $N_1, N_2 \in \mathcal{N}$ satisfy $(\text{relint}N_1) \cap N_2 \neq \emptyset$ then $N_1 \subset N_2$,*
2. *The relative interiors of the cones in $\mathcal{N}$ partition S, i.e. $\bigcup_{N\in\mathcal{N}} \text{relint}N = S$.*

Lemma 1 is subtle but important as it contains a key geometric insight that allow us to prove our theorem. Next, let $Q \subset \mathbb{R}^d$ be a bounded polyhedron with vertex set V and nonempty interior, i.e. whose affine hull is d-dimensional. For any face $F \in \mathscr{F}(Q)$ define $V(F) = F \cap V$ to be the vertices of F and $N_F = \left\{y \in \mathbb{R}^d \mid \forall x \in F, \forall z \in Q, y^T x \leq y^T z\right\}$ to be the normal cone to F. That N_F is a (pointed) polyhedral cone follows from noting that it can be equivalently expressed as a finite collection of linear constraints involving the vertices of F and Q: $N_F = \left\{y \in \mathbb{R}^d \mid \forall x \in V(F), \forall z \in V, y^T x \leq y^T z\right\}$. The *normal fan* for Q, $\mathcal{N}(Q) = \{N_F\}_{F\in\mathscr{F}(Q)}$, is defined to be the set of all normal cones for faces of Q. Noting that Q is bounded and therefore has a recession cone of $\{0\}$, the following Lemma is implied by Proposition 1 and Corollary 1 of [40]:

Lemma 2. *Let $\mathcal{N}(Q)$ be the normal fan of a bounded polyhedron Q with non-empty interior in $\mathbb{R}$. Then*

1. *$\mathcal{N}(Q)$ is a fan,*
2. *for any nonempty faces $F_1, F_2 \in \mathscr{F}(Q)$, $F_1 \subset F_2$ iff $N_{F_1} \supset N_{F_2}$,*
3. *$\bigcup_{F\in\mathscr{F}(Q)} \text{relint}N_F = \mathbb{R}^d$,*
4. *every nonempty face $F \in \mathscr{F}(Q)$ satisfies $\text{relint}N_F = \left\{y \in \mathbb{R}^d \mid F = S(y, Q)\right\}$.*

We will also makes use of the following two results. The first is implied by Theorem 2.7, Corollary 2.14, and Problem 7.1 in [73]:

Lemma 3. *Let $Q \subset \mathbb{R}^d$ be a bounded polyhedron with nonempty interior, $F \in \mathscr{F}(Q)$, and N_F the normal cone to F. Then $\dim F + \dim N_F = d$.*

The second Lemma states a kind of neighborliness for the cones in $\mathcal{N}(Q)$:

Lemma 4. *Let $Q \subset \mathbb{R}^d$ be a bounded polyhedron with nonempty interior. For any $N \in \mathcal{N}(Q)$ and $x \in \operatorname{relint} N$ there exists a $\delta > 0$ such that for any $y \in B(x, \delta)$ there is a $N' \in \mathcal{N}(Q)$ with $y \in \operatorname{relint} N'$ and $N \subset N'$.*

Proof. Let $N \in \mathcal{N}(Q)$ and $x \in N$ be given. We say that $N' \in \mathcal{N}(Q)$ *occurs within* δ (for $\delta > 0$) if there is some $y \in B(x, \delta)$ with $y \in \operatorname{relint} N'$. Now suppose that there is an $N' \in \mathcal{N}(Q)$ that occurs within δ for all $\delta > 0$. Since N' is a closed convex cone it must be that $x \in N'$ so we may conclude from Lemma 1 that $N \subset N'$. Next, let $\mathcal{M}$ be the set of cones in $\mathcal{N}(Q)$ which do not contain N and suppose that for all $\delta > 0$ there is some $N' \in \mathcal{M}$ that occurs within δ. Since $|\mathcal{M}|$ is finite, this is only possible if there is a cone $N' \in \mathcal{M}$ that occurs within δ for all $\delta > 0$. However, this leads to a contradiction since N' must contain N so the Lemma follows. □

We are now ready to prove our main theorem which is restated below with $S(c, Q) = \arg\min_{x \in Q} x^T c(\alpha)$ for simplicity.

Theorem 3. *Let $Q \subset \mathbb{R}^d$ be a bounded polyhedron with nonempty interior and $c : [0, 1] \to \mathbb{R}^d$ a continuous function. Then for some $N \in \mathbb{Z}_+ \cup \{\infty\}$ there exists a countable partition $\Gamma = \{\gamma_i\}_{i=0}^N$ of $[0, 1]$ with corresponding faces $F_i \in \mathscr{F}(Q)$ satisfying $F_i \neq F_{i+1}$ and $F_i \cap F_{i+1} \neq \emptyset$ and $F_i = S(c(\alpha), Q)\, \forall \alpha \in \gamma_i$. Moreover, F_i never has the same dimension as F_{i+1} and the boundary between γ_i, γ_{i+1} is $)[$ iff $\dim F_i < \dim F_{i+1}$ and $]($ otherwise.*

Proof. For ease of notation let $f(x) = S(c(x), Q)$ and for $k = 0, \ldots, d$ define $\omega_k = \left\{x \in [0, 1] \mid \dim N_{f(x)} \geq k\right\}$ to be the set of all arguments to c whose normal cone is at least k-dimensional. Moreover, for any $x \in [0, 1]$ define $\sigma(x) = \{y \in [0, x] \mid \forall z \in [y, x], f(x) = f(z)\} \cup \{y \in [x, 1] \mid \forall z \in [x, y], f(x) = f(z)\}$ to be the largest contiguous set containing x over which f remains constant and let $m(x) = \inf \sigma(x)$ and $M(x) = \sup \sigma(x)$ be its infinimum and supremem, respectively. The proof follows by induction on $k = d, d-1, \ldots, 0$ with the inductive hypothesis that for some $N_k \in \mathbb{Z}_+ \cup \{\infty\}$ there exists a countable partition $\Gamma^k = \left\{\gamma_i^k\right\}_{i=0}^{N_k}$ of ω_k with corresponding faces $F_i^k \in \mathscr{F}(Q)$ satisfying $F_i^k = S(c(\alpha), Q)\, \forall \alpha \in \gamma_i^k$.

Base case ($k = d$): Let $x \in \omega_d$ so that $\sigma(x) \subset \omega_d$. Since $N_{f(x)}$ is d-dimensional, $\operatorname{int} N_{f(x)} = \operatorname{relint} N_{f(x)}$ so continuity of c implies that $\sigma(x)$ is a (non-empty) open interval with $m(x) < M(x)$. It follows that $\Gamma^k = \{\sigma(x) \mid x \in \omega_d\}$ defines a partition of ω_d into a set of open intervals. Each interval contains (an infinite number) of rational numbers, and we see that Γ^k is countable by assigning to each interval a rational number

that it contains.

Inductive step: Let $x \in \omega_k \backslash \omega_{k+1}$. There are two cases to consider. If $m(x) < M(x)$ then $(m(x), M(x)) \subset \sigma(x)$ contains a rational number. Thus, the set $\Gamma_o^k = \{\sigma(x) \mid x \in \omega_k \backslash \omega_{k+1}, m(x) < M(x)\}$ is countable. Otherwise, if $m(x) = x = M(x)$ then by Lemma 4 there is a $\delta > 0$ such that if $y \in B(x, \delta)$ then $N_{f(x)} \subset N_{S(y,Q)}$. Continuity of c implies that there is a $\varepsilon > 0$ for which $c((x - \varepsilon, x + \varepsilon)) \subset B(x, \delta)$ and hence $(x - \varepsilon, x + \varepsilon) \backslash \{x\} \subset \omega_{k+1}$. Assigning to x any rational number in $(x - \varepsilon, x + \varepsilon)$ and letting $\Gamma_c^k = \{\sigma(x) \mid x \in \omega_k \backslash \omega_{k+1}, m(x) = M(x)\}$, we may appeal to the inductive hypothesis to conclude that Γ_c^k is countable. Finally, $\Gamma^k = \Gamma_o^k \cup \Gamma_c^k \cup \Gamma^{k+1}$ is a finite union of countable sets and therefore countable.

Since $\omega_0 = [0, 1]$ we have shown that $\Gamma = \Gamma^0$ is a countable partition of $[0, 1]$ into intervals over which f is constant. Now consider two consecutive intervals $\gamma_i, \gamma_{i+1} \in \Gamma$ and let M be the supremum of γ_i. If $M \notin \gamma_i$ then since cone N_{F_i} is closed, $c(M) \in N_{F_i}$. Since $c(M) \in \text{relint}\, N_{F_{i+1}}$ by assumption, it follows that $N_{F_{i+1}}$ is a proper subset of N_{F_i} and hence that F_i is a proper subset of F_{i+1}. Otherwise, if $M \in \gamma_i$ then the continuity of c and Lemma 4 imply that N_{F_i} is a proper subset of $N_{F_{i+1}}$ so F_{i+1} is a proper subset of F_i. In either case $F_i \cap F_{i+1} \neq \emptyset$ and Lemma 3 implies the dimensionality result of our Theorem. □

Interpretable Cost Settings

We now provide a simple linear cost scheme that has globally predictable effects on the dictionary. For all $s \in \mathcal{C}^*, p \in \mathcal{P}_\mathcal{C}, \hat{p} \in \mathcal{P}$ we set $d_s = \tau$, $c_p = 1$, $\hat{c}_{\hat{p}} = \alpha\lambda$ if $\hat{p}$ uses as unigram (i.e. is a character), and $\hat{c}_{\hat{p}} = \lambda$ otherwise. We constrain $\tau, \lambda \geq 0$ and $\alpha \in [0, 1]$. In words, all document pointer costs are 1, all dictionary costs τ, and dictionary pointer costs are λ if they use a string and $\alpha\lambda$ if they use a character. The effects these parameters have on the compression may be understood by varying a single parameter and holding all others constant:

Varying τ controls the minimum frequency with which $s \in \mathcal{C}^*$ must be used before it enters the dictionary; if few pointers use s it is cheaper to construct s "in place" using shorter N-grams. Long N-grams appear less frequently so $\uparrow \tau$ biases the dictionary towards shorter N-grams.

Varying λ has a similar effect to τ in that it becomes more expensive to construct s as λ increases, so the overall cost of dictionary membership increases. The effect is more nuanced, however, since the *manner* in which s is constructed also matters; s is more likely to enter the dictionary if it shares long substrings with

existing dictionary strings. This suggests a kind of grouping effect whereby groups of strings that share many substrings are likely to enter together.

Varying α controls the Dracula's propensity to use characters in place of pointers in the dictionary and thereby directly modulates dictionary depth. When $\alpha < \frac{1}{K}$ for $K = 2, 3, \ldots$, all dictionary N-grams of length at most K are constructed entirely from characters.

Landmarks on Dracula's Polyhedron

While Dracula's representations are typically deep and space saving, it is important to note that valid Dracula solutions include all of CFL's solutions as well as a set of fully redundant representations that use as many pointers as possible. The BoN features computed from these "space maximizing" compressions yield the traditional BoN features containing all N-grams up to a maximum length K. A cost scheme that includes all pointers using all N-grams up to length K is obtained by setting all costs to be *negative*, except for $t_s = \infty$ for all $s \in \mathcal{C}^*$ where $|s| > K$ (to disallow these strings). The optimal compression then includes all pointers with negative cost and each document position is reconstructed K times. Moreover, it is possible to restrict representations to be valid CFL solutions by disallowing all non-unigram pointers for dictionary reconstruction, i.e. by setting $\hat{c}_p = \infty$ if p is not a single character string.

3.4.2 Dictionary Diffusion

We now discuss how to incorporate dictionary information from a compression $\mathfrak{D} = (S, P, \hat{P})$ into the BoN features for each corpus document. It will be convenient to store the BoN feature vectors x^{D_k} for each document as rows in a feature matrix $X \in \mathbb{Z}^{|\mathcal{C}| \times (|S| + |\Sigma|)}$ and the BoN feature vectors x^s for each dictionary string as rows in a feature matrix $G \in \mathbb{Z}^{(|S|+|\Sigma|) \times (|S|+|\Sigma|)}$. We also include rows of all 0's for every character in Σ to make G a square matrix for mathematical convenience. Graphically, this procedure transforms $\mathfrak{D}$ into a simpler DAG, $\mathfrak{D}_{\mathfrak{R}}$, by collapsing all multi-edges into single edges and labeling the resulting edges with an appropriate x^s_z. For any two features s, z, we say that s is higher (lower) order than z if it is a successor (predecessor) of z in $\mathfrak{D}$.

Once our feature extraction process throws away positional information in the pointers higher order features capture more information than their lower order constituents since the presence of an $s \in S$ formed by concatenating features $z_1 \ldots z_m$ indicates the order in which the z_i appear and not just that they appear.

Conversely, since each z_i appears in the same locations as s (and typically many others), we can obtain better estimates for coefficients associated with z_i than for the coefficient of s. If the learning problem does not require the information specified by s we pay an unnecessary cost in variance by using this feature over the more frequent z_i.

In view of this reasoning, feature matrix X captures the highest order information about the documents but overlooks the features' lower order N-grams (that are indirectly used to reconstruct documents). This latter information is provided by the dictionary's structure in G and can be incorporated by a graph diffusion process that propagates the counts of s in each document to its constituent z_i, which propagate these counts to the lower order features used to construct them, and so on. This process stops once we reach the characters comprising s since they are atomic. We can express this information flow in terms of G by noting that the product $G^T x^{D_k} = \sum_{s \in S \cup \Sigma} x_s^{D_k} x^s$ spreads $x_s^{D_k}$ to each of the z_i used to reconstruct s by multiplying $x_s^{D_k}$ with $x_{z_i}^s$, the number of times each z_i is directly used in s. Graphically, node s in $\mathfrak{D}_{\mathfrak{R}}$ sends $x_s^{D_k}$ units of flow to each parent z_i, and this flow is modulated in proportion to $x_{z_i}^s$, the strength of the edge connecting z_i to s. Performing this procedure a second time, i.e. multiplying $G^T(G^T x^{D_k})$, further spreads $x_s^{D_k} x_{z_i}^s$ to the features used to reconstruct z_i, modulated in proportion to their usage. Iterating this procedure defines a new feature matrix $\hat{X} = XH$ where $H = I + \sum_{n=1}^{\infty} G^n$ spreads the top level x^{D_k} to the entire graph[6].

When G is generated from the relaxation of Dracula and $t \in (0, 1]^{|S|}$ are the dictionary coefficients, any $s \in S$ with $t_s < 1$ will have $G_{sz} \leq t_s \forall z \in S$. In order to prevent overly attenuating the diffusion we may wish to normalize row s in G by t_s^{-1} for consistency. We note that a variety of other weightings are also possible to different effects. For example, weighting G by a scalar $\rho \geq 0$ attenuates or enhances the entire diffusion process and mitigates or enhances the effect of features the farther away they are from directly constructing any feature directly used in the documents.

We can interpret the effect of the dictionary diffusion process in view of two equivalent regularized learning problems that learn coefficients $\beta, \eta \in \mathbb{R}^{|S \cup \Sigma|}$ for every feature in $S \cup \Sigma$ by solving

$$\begin{aligned} &\underset{\beta \in \mathcal{R}^{|S \cup \Sigma|}}{\text{minimize}} \quad L(\hat{X}\beta) + \lambda R(\beta) \\ \equiv &\underset{\eta \in \mathcal{R}^{|S \cup \Sigma|}}{\text{minimize}} \quad L(X\eta) + \lambda R\left((I - G)\eta\right). \end{aligned} \tag{3.9}$$

[6]This sum converges because G corresponds to a finite DAG so it can be permuted to a strictly lower triangular matrix so that $\lim_{n \to \infty} G^n = \mathbf{0}$.

We assume that L is a convex loss (that may implicitly encode any labels), R is a convex regularization penalty that attains its minimum at $\beta = \mathbf{0}$, and that a minimizer β^* exists. Note that adding an unpenalized offset does not affect our analysis. The two problems are equivalent because H is defined in terms of a convergent Neumann series and, in particular, $H = (I - G)^{-1}$ is invertible. We may switch from one problem to the other by setting $\beta = H^{-1}\eta$ or $\eta = H\beta$.

When $\lambda = 0$ the two problems reduce to estimating β/η for unregularized models that only differ in the features they use, $\hat{X}$ or X respectively. The equivalence of the problems shows, however, that using $\hat{X}$ in place of X has *no effect* on the models as their predictions are always the same. Indeed, if β^* is optimal for the first problem then $\eta^* = H\beta^*$ is optimal for the second and for any $z \in \mathbb{R}^{|S\cup\Sigma|}$, the predictions $z^T\eta^* = (z^T H)\beta^*$ are the same. Unregularized linear models – including generalized linear models – are therefore *invariant* to the dictionary reconstruction scheme and only depend on the document feature counts x^{D_k}, i.e. how documents are reconstructed.

When $\lambda > 0$, using $\hat{X}$ in place of X results in a kind of graph Laplacian regularizer that encourages η_s to be close to $\eta^T x^s$. One interpretation of this is effect is that η_s acts a "label" for s: we use its feature representation to make a prediction for what η_s should be and penalize the model for any deviations. A complementary line of reasoning uses the collapsed DAG $\mathfrak{D}_{\mathfrak{R}}$ to show that (3.9) favors lower order features. Associated with every node $s \in S \cup \Sigma$ is a flow η_s and node z sends η_z units of flow to each of its children s. This flow is attenuated (or amplified) by x^s_z, the strength of the edge connecting z to s. In turn, s adds its incoming flows and sends out η_s units of flow to its children; each document's prediction is given by the sum of its incoming flows. Here R acts a kind of "flow conservation" penalty that penalizes nodes for sending out a different amount of flow than they receive and the lowest order nodes (characters) are penalized for any flow. From this viewpoint it follows that the model prefers to disrupt the flow conservation of lower order nodes whenever they sufficiently decrease the loss since they influence the largest number documents. Higher order nodes influence fewer documents than their lower order constituents and act as high frequency components.

3.5 Solving Strategies for Shallow Compression

This section discusses two approximate solving strategies for Dracula by exploring them with its simpler shallow variant CFL when all costs are non-negative. It is, of course, possible to simply used Dracula's

linear programming relaxation as a solution, and this is the perspective our experiments with Dracula use in the Deep Experiments Section 3.6.3. The relaxed solution can also be rounded to a binary one according to the procedure discussed in Section 3.3.3, but the hope of this section is to discuss alternative strategies that may be faster or yield better binary solutions. At a high level, the strategies outlined in this section can be seen as instances of *iterative binarization* procedures that minimize a sequence of interrelated problems until arriving at a binary solution. In particular, iterative binarization explores the domain of a function $f(x)$ via a parameterized surrogate $f_\zeta(x)$ until it finds a binary solution. It does this by keeping track of the current solution x and a state parameter ζ and alternating between updating ζ and (possibly approximately) minimizing $f_\zeta(x)$ using x as a warm start. The hope is that if $f_\zeta(x)$ and the update procedure for ζ are chosen carefully, iterative binarization will arrive at a nearly optimal binary solution. High level pseudocode for iterative binarization is given below:

while x is not binary **do**

update ζ

$x \leftarrow$ **argmin** $f_\zeta(x)$

end while

We have left the update rule for ζ general because it may simply rely on the iteration count or may use x in a more sophisticated manner, as in expectation maximization or homotopy/ continuation methods.

It is easiest to work with the mathematical structure of our problems using the Alternating Directions Method of Multipliers (ADMM) [7] optimization framework. While ADMM has traditionally been used for convex problems, it also converges (to a local optimum) for non-convex problems [7]. It minimizes a function $f(w) = h(w) + g(Aw)$ that is separable into two or more terms by solving the equivalent problem $\min_{w,z} \; h(w) + g(z)$ subject to $z = Aw$. This minimization is performed by operating on the augmented Lagrangian

$$\mathcal{L}_\rho(w, z, y) = h(w) + g(z) + y^T(Aw - z) + \frac{\rho}{2}\left\|Aw - z\right\|_2^2 \tag{3.10}$$

Here y is a dual variable that enforces the equality constraint and serves as a conduit of information between w and z. At each step ADMM minimizes $\mathcal{L}_\rho$ with respect to w while holding all other variables fixed, then z, and finally updates y according to [7]. This procedure solves for w or z separately and brings them into agreement as the algorithm progresses; upon convergence $\mathcal{L}_\rho$ equals f because the equality constraint is met and the last two terms are 0.

The key to ADMM is that minimization with respect to w (or z) be fast: the way in which f is split is essential to the success of the algorithm. A complicating issue is that the w–update can become difficult when $A \neq I$. Linearized ADMM [49] remedies this by replacing the last term in (3.10) with its first order Taylor expansion around the previous value of w, $\bar{w}$, and a regularization term:

$$\frac{\rho}{2}\left\|Aw - z\right\|_2^2 \approx \rho(A\bar{w} - z)^T Aw + \frac{\mu}{2}\left\|w - \bar{w}\right\|_2^2. \tag{3.11}$$

This change is made only during the w–update, i.e., updates to all other variables remain the same, and it transforms the w–update into a simpler proximal mapping.

There are generally several fundamentally different ways to split a problem for the ADMM framework and we will now discuss two different iterative binarization procedures that split CFL in different ways. For convenience we write the relevant linearly relaxed versions of equations (3.2) and (3.3) below.

$$\tilde{R}_{D_k}(t; c) = \underset{w \in [0,1]^{|\mathcal{P}(D_k)|}}{\text{minimize}} \quad \sum_{p \in \mathcal{P}(D_k)} w_p c_p \quad \text{subject to} \quad X^{D_k} w \geq \mathbf{1};\ w \leq V^{D_k} t. \tag{3.12}$$

$$\underset{t \in [0,1]^{|\mathcal{C}^*|}}{\text{minimize}} \sum_{D_k \in \mathcal{C}} \tilde{R}_{D_k}(t, c) + \sum_{s \in \mathcal{C}^*} t_s d_s \tag{3.13}$$

It will be convenient to work with the linear programming relaxation of CFL in an alternate form that implicitly solves for the dictionary variables t. To this end, we define $w(k)$ to be the minimizer of equation (3.12) for document D_k and let $w = [w(1)^T, \ldots, w(N)^T]^T$ be the vector found by stacking the $w(k)$ for each of our N corpus documents. We will similarly let c be the correspond vector obtained by stacking the pointer costs c_p so that w and c are both $m = |\mathcal{P}_\mathcal{C}|$–dimensional vectors. Next, let $X = \text{diag}(X^{D_1}, \ldots, X^{D_N})$ be the block-diagonal matrix containing the reconstruction constraint matrices and define $J(s) \subset \mathcal{P}_\mathcal{C}$ to be the set of all pointers that correspond to the same string s at different locations in the corpus. Using n to be the total corpus length in characters, CFL may now be written as

$$\begin{aligned} \underset{w}{\text{minimize}} \quad & w^T c + \sum_{s \in \mathcal{C}^*} d_s \|w_{J(s)}\|_\infty \\ \text{subject to} \quad & Xw \geq \mathbf{1}, \quad w \leq \mathbf{1} \end{aligned} \tag{3.14}$$

where we have eliminated the non-negativity constraint on w since any optimal solution will necessarily satisfy it.

3.5.1 Iterative Reweighting

Iterative reweighting solves a sequence of related linear programs that weight all terms involving the pointer variables by a diagonal $m \times m$ matrix $P^{(i)}$ of positive weights.

$$\underset{w}{\text{minimize}} \quad w^T \tilde{c}^{(i)} + \sum_{s \in \mathcal{C}^*} d_s \|P^{(i)}_{J(s)J(s)} w_{J(s)}\|_\infty \qquad \text{subject to} \quad Xw \geq \mathbf{1}, \quad \mathbf{0} \leq w \leq \mathbf{1}. \tag{3.15}$$

Here $\tilde{c}^{(i)} = P^{(i)}c$ for brevity. We use $P^{(1)} = I$ for the first iteration and $P^{(i+1)}_{pp} = \max\left\{1, (w^{(i)}_p + \epsilon)^{-1}\right\}$, where $w^{(i)}$ is the solution to the previous problem. This scheme is inspired by the iterative reweighting method of Candès et al. [11] for solving problems involving L_0 regularization. At a high level, reweighting can be motivated by noting that (3.15) recovers the correct *binary* solution if ϵ is sufficiently small and we use as weights a nearly binary solution to the binary problem (3.3). Since we do not know the correct weights, we estimate them from our best guess to the solution of (3.15). In turn, $P^{(i+1)}$ punishes coefficients that were small in $w^{(i)}$ and, taken together with the constraint $Xw \geq \mathbf{1}$, pushes the solution to be binary.

ADMM Solution

We demonstrate an efficient and parallel algorithm to solve (3.15) based on ADMM that uses problem structure to achieve a run time of $O(K^2 n)$ per step of ADMM (linear in the corpus length) assuming the maximum N-gram length is K. This is helpful because K is relatively small in most scenarios: long N-grams tend to appear only once and are not helpful for compression. Moreover, they are rarely used in NLP applications since the relevant signal is captured by smaller fragments.

Dropping the $P^{(i)}$ superscripts for legibility, we can exploit problem structure by splitting (3.15) into

$$h(w) = w^T \tilde{c} + \sum_{s \in \mathcal{C}^*} d_s \|P_{J(s)J(s)} w_{J(s)}\|_\infty + I(w \geq \mathbf{0}), \qquad g(z) = I(Xz - \mathbf{1} \geq \mathbf{0}) \tag{3.16}$$

where $I\{\bullet\}$ is the convex indicator function that is ∞ if the constraint inside the braces is not met and is 0 otherwise.

Minimizing w The dual of this problem is a quadratic knapsack problem solvable in linear expected time [10], we provide a similar algorithm that solves the primal formulation. We solve for each $w_{J(s)}$ separately since the optimization is separable in each block of variables. We show in Appendix A.1 that $w_{J(s)} = 0$ if $\|P^{-1}_{J(s)J(s)} q_{J(s)}\|_1 \leq c(s)$, where $q_{J(s)} = \max\left\{\rho z_{J(s)} - \tilde{c}_{J(s)} - y_{J(s)}, 0\right\}$ and the max operation is applied elementwise. Otherwise, $w_{J(s)}$ is non-zero and the L_∞ norm only affects the maximal coordinates of $P_{J(s)J(s)} w_{J(s)}$. For simplicity of exposition, we assume that the coefficients of $w_{J(s)}$ are sorted in decreasing order according to $P_{J(s)J(s)} q_{J(s)}$, i.e., $[P_{J(s)J(s)} q_{J(s)}]_j \geq [P_{J(s)J(s)} q_{J(s)}]_{j+1}$. This is always possible by permuting coordinates. We show in Appendix A.1 that, if $P_{J(s)J(s)} w_{J(s)}$ has r maximal coordinates, then

$$w_{J(s)_j} = P^{-1}_{J(s)_j J(s)_j} \min\left\{ P_{J(s)_j J(s)_j} q_{J(s)_j}, \frac{\sum_{v=1}^{r} P^{-1}_{J(s)_v J(s)_v} q_{J(s)_v} - c(s)}{\sum_{v=1}^{r} P^{-2}_{J(s)_v J(s)_v}} \right\}. \tag{3.17}$$

We can find r by searching for the smallest value of r for which exactly r coefficients in $P_{J(s)J(s)} w_J(s)$ are maximal when determined by the formula above. As discussed in Appendix A.1, an algorithm similar to the linear-time median-finding algorithm can be used to determine $w_{J(s)}$ in linear expected time.

Minimizing z Solving for z is tantamount to projecting a weighted combination of w and y onto the polyhderon given by $Xz \geq \mathbf{1}$ and is best solved by taking the dual. It can be shown (see Appendix A.1) that the dual optimization problem is

$$\underset{\alpha}{\text{minimize}} \quad \frac{1}{2}\alpha^T H \alpha - \alpha^T(\rho\mathbf{1} - X(y + \rho w)) \qquad \text{subject to} \quad \alpha \geq \mathbf{0} \tag{3.18}$$

where $\alpha \in \mathbb{R}^n_+$ is a dual variable enforcing $Xz \geq \mathbf{1}$ and $H = XX^T$. Strong duality obtains and z can be recovered via $z = \rho^{-1}(y + \rho w + X^T\alpha)$.

The matrix H has special structure when $\mathcal{C}^*$ is a set of k-grams no longer than K characters. In this case, Appendix A.1.3 shows that H is a $(K-1)$–banded positive definite matrix so we can find its Cholesky decomposition in $O(K^2 n)$. We then use an active-set Newton method [31] to solve (3.18) quickly in approximately 5 Cholesky decompositions. A second important property of H is that, if N documents $n_1, \ldots, n_N$ words long are compressed jointly and no k-gram spans two documents, then H is block-diagonal with block

i an $n_i \times n_i$ $(K-1)$–banded matrix. This allows us to solve (3.18) *separately* for each document. Since the majority of the time is spent solving for z, this property allows us to parallelize the algorithm and speed it up considerably.

3.5.2 $\ell_1 \to \ell_\infty$ Homotopy

Our second method is a homotopic method that relies on a homotopy parameter $\zeta \in [0,1]$ as its state variable and steadily increases ζ from 0 to 1. We use for $f_\zeta(x)$ in the iterative binarization procedure a function whose domain χ_ζ shrinks as ζ increases so that $\chi_{\zeta'} \subset \chi_\zeta$ for $\zeta' > \zeta$, i.e. the χ_ζ are nested and become more constrained with larger ζ. When $\zeta = 0$, $f_0(x)$ corresponds to CFL's linear programming relaxation of (3.14). As ζ increases, it steadily transforms into $f_1(x)$, a non–convex function whose local optima are always binary. This transformation relies on warm starts to provide a good initial solution for each progressively harder problem. It traces out a solution path that starts at the continuous minimizer of (3.14) and ends at a high quality binary solution nearby.

We define our homotopy by first showing how to express (3.14) as a (non–convex) continuous problem whose minimizer must always be binary. Define $T(i) \subset \mathcal{P}_\mathcal{C}$ for $i = 1, \ldots, n$ to be the set of all pointers that can be used to reconstruct position i, i.e. $T(i)$ is the index of all columns in X that are 1 at row i. Then (3.14) is equivalent to

$$\begin{aligned} \underset{w}{\text{minimize}} \quad & w^T c + \sum_{s \in \mathcal{C}^*} d_s \|w_{J(s)}\|_\infty \\ \text{subject to} \quad & \|w_{T(i)}\|_\infty = 1 \; \forall i = 1, \ldots, n \\ & w \geq \mathbf{0}. \end{aligned} \tag{3.19}$$

The solution to (3.19) must be binary because the objective pushes w towards 0 and any $w_p < 1$ has no impact on the ∞–norm constraint.

We use the formulation in (3.19) to define our homotopy. Specifically, for $\zeta \in [0,1]$ we consider the problem

$$\begin{aligned} \underset{w}{\text{minimize}} \quad & w^T c + \sum_{s \in \mathcal{C}^*} d_s \|w_{J(s)}\|_\infty \\ \text{subject to} \quad & \|w_{T(i)}\|_\infty \geq \zeta \; \forall i = 1, \ldots, n \\ & Xw \geq \mathbf{1}, \; w \geq \mathbf{0}. \end{aligned} \tag{3.20}$$

When $\zeta = 0$ the ∞–norm constraint cannot be active and the problem reduces to the linear relaxation of

(3.14). However, when $\zeta = 1$ the ∞–norm constraint is more restrictive than the linear constraint and the problem reduces to (3.19). As a matter of interest, note that when all N-grams are at most length K, (3.20) is convex even for values of $\zeta > 0$. For instance, because $Xw \geq \mathbf{1}$ is feasible only if $\|w_{T(i)}\|_\infty \geq \frac{1}{2K}$, setting $\zeta = \frac{1}{2K}$ preserves convexity. Moreover, given a unique solution w^* to the relaxation of (3.14), the largest value of ζ at which (3.20) is still convex is given by $\zeta = \min_{i=1,\ldots,n} \|w^*_{T(i)}\|_\infty$.

3.5.3 ADMM Formulation

We split (3.20) into three distinct terms and use linearized ADMM for efficiency:

1. $h(w) = w^T c + \sum_{s \in \mathcal{C}^*} d_s \|w_{J(s)}\|_\infty + I\{w \geq 0\}$
2. $g_1(z) = I\{z \geq 1\}$
3. $g_2(\theta) = I\left\{\|\theta_{T(i)}\|_\infty \geq \zeta \; \forall i = 1, \ldots, n\right\}$.

We also require that the equality constraints $z = Xw$ and $w = \theta$ hold upon convergence. Thus, h corresponds to the storage costs in (3.20), g_1 to the convex reconstruction constraint, and g_2 to the non-convex constraint that drives the homotopy. Notice that while the constraints imply $z = X\theta$, we do not enforce this equality constraint. Ignoring this constraint allows us to simply alternate between solving for w and (z, θ) simultaneously. Moreover, we only need to linearize the $\|Xw - z\|_2^2$ term in $\mathcal{L}_\rho$ when solving for w.

Minimization

Solving for w Simple algebra shows that the linearization of $\mathcal{L}_\rho$ is separable in each $s \in \mathcal{C}^*$. The Appendix A.2 shows that each of these subproblems is a quadratic knapsack problem is of the form $\frac{\rho+\mu}{2}\|w_{J(s)} - q\|_2^2 + d_s\|w_{J(s)}\|_\infty$ for appropriately defined q and can therefore be solved in $O(m_s)$ time, where m_s is the dimensionality of $w_{J(s)}$.

Solving for z We show in the Appendix A.2 that simple thresholding minimizes $\mathcal{L}_\rho$, i.e. $z = \max\left(Xw + \rho^{-1}y^{(z)}, 1\right)$. Here $y^{(z)}$ is the dual variable corresponding to the $z = Xw$ constraint and the max operation is applied elementwise.

Solving for θ Minimizing $\mathcal{L}_\rho$ with respect to θ yields a non-convex problem that can be solved *exactly* by casting it as an encoding problem and using the algorithm in Section 3.3.1. Letting $v = w + \rho^{-1}y^{(\theta)}$, where $y^{(\theta)}$ is the dual variable for $w = \theta$, the relevant parts of $\mathcal{L}_\rho$ are

$$\left\| v - \theta \right\|_2^2 + g_2(\theta) \tag{3.21}$$

Notice that θ_i helps satisfy g_2's constraint only if $\theta_i \geq \zeta$ and that this constraint is indifferent between $\theta_i = \zeta$ and $\theta_i > \zeta$. Since we are trying to find the closest point to v that satisfies g_2, it follows that $\theta_i = v_i$ if $v_i \geq \zeta$ and that $\theta_i = v_i$ or $\theta_i = \zeta$ otherwise. This is an encoding problem with pointer cost $\eta_i = \max(0, \zeta - v_i)^2$ whose solution, $\psi \in \{0,1\}^m$, determines θ via $\theta = \max(v, \zeta\psi)$.

Runtime Analysis

Assuming all N-grams up to length K are used, each pass of LADMM requires $\Theta(Kn)$ operations. In particular, finding w and z requires linear–time operations on vectors of size $O(Kn)$ formed by multiplying X or X^T by a vector. As discussed in Appendix A.2.1, the structure and sparsity of X allows us to perform this multiplication in $\Theta(|\mathcal{P}|) = \Theta(Kn)$ operations, rather than $\Theta(n|\mathcal{P}|)$ as would be the case with general multiplication. Similarly, the encoding problem to find θ only requires $O(Kn)$ operations and so each step of LADMM takes $\Theta(Kn)$. It is also important to note that the w–update parallelizes across individual substrings $s \in \mathcal{C}^*$ and that the z and θ–updates, as well as multiplication by X or X^T, all parallelize across individual documents.

Parameter Tuning

Linearized ADMM relies critically on the parameters μ and ρ to converge. Roughly speaking, μ controls how far w deviates from $\bar{w}$ and therefore depends on how well $\|Aw - z\|_2^2$ is approximated linearly. On the other hand, ρ controls how much each w–update focuses on minimizing h versus satisfying the reconstruction constraints and has a significant impact on the number of steps necessary for convergence.

Selecting μ It can be shown [49] that linearized ADMM converges if $\mu \geq \rho\|X\|_2^2$, with convergence being fastest when equality holds. The following theorem shows how to select μ for our problem.

Theorem 4. *A tight upper bound for $\|X\|_2^2$ is given by*

$$\sigma = \frac{K^3}{3} + \frac{K^2}{2} + \frac{K}{6}.$$

Proof. We assume that N documents are compressed jointly so that $n = \sum_{i=1}^{N} n_i$. We know that XX^T is an $n \times n$ block diagonal matrix with block $B^{(i)} \in \mathbb{Z}_+^{n_i \times n_i}$ corresponding to document i and hence $\|X\|_2^2 = \max_{i=1,\ldots,N} \gamma_i$ where $\gamma_i = \|B^{(i)}\|_2$. The result in the Appendix A.2.1 shows that the first and last $K-1$ rows of $B^{(i)}$ have row sum less than σ and that the submatrix formed by deleting these rows is Toeplitz with row sum equal to σ. Thus, Geršgorin's disc theorem [67] gives the bound $\gamma_i \leq \sigma$.

To show that this bound is tight, we consider what happens to γ_i as n_i increases. Application of a standard result from eigenvalue perturbation theory [17] reveals that

$$|\gamma_i - \sigma| \leq \left\| \tfrac{1}{\sqrt{n_i}} B^{(i)} \mathbf{1} - \tfrac{\sigma}{\sqrt{n_i}} \mathbf{1} \right\|_2 \leq \sqrt{\frac{2K-2}{n_i}}$$

and so γ_i approaches σ when n_i is large relative to K. □

Tuning ρ Selecting ρ remains an open problem for general ADMM [7] and we found that, for our problem, convergence is fastest when ρ starts small and is aggressively increased based upon the progress of the algorithm. Focusing only on variables for the convex part of (3.10), we measure convergence based on the quantities

$$r = \|Xw - z\|_\infty \quad u = \|z - \bar{z}\|_\infty \tag{3.22}$$

where $\bar{z}$ is the previous value of z. We start with $\rho = 1/K$ and update $\rho = 1.5\rho$ if the average value of r/u over the last 15 iterations is greater than 5 and ρ has not been updated in as many iterations. Compared to traditional schemes which increase/decrease ρ if $\max(\frac{r}{u}, \frac{u}{r}) \geq 10$, ours is more aggressive but also utilizes a smoother estimate. Finally, our scheme only increases ρ because it starts out small and, for our problem, erroneously decreasing ρ slows convergence considerably.

3.5.4 Comparison with Iterative Reweighting

The IR algorithm uses an IB scheme where $f_\zeta(x)$ is a weighted linear relaxation of (3.14), with the state variable $\zeta \in \mathbb{R}_+^{|\mathcal{P}|}$ providing weights for each pointer's indicator variable. It uses ADMM to solve each

relaxation and requires $\Theta(K^2 n)$ operations per ADMM step. The brunt of the work is spent projecting the current solution to satisfy the reconstruction constraint $Xw \geq 1$. As ADMM progresses, the algorithm keeps track of the projection which achieves the lowest storage cost when all non-zero entries are set to 1. This "best projection" is used to calculate the weights for the next weighted problem.

Our homotopic algorithm was designed to address several drawbacks of the IR algorithm. The latter uses a simple rounding scheme — it simply sets all non-zero entries to 0 — and does not achieve a reasonable binary solution, i.e. a compression without redundant pointers, until it has nearly converged. In contrast, setting the homotopy parameter to 1 in our homotopic method immediately starts generating binary solutions that have no redundant pointers because of the encoding algorithm.

More fundamentally, each ADMM step of the IR algorithm is slower than the homotopic scheme's because of the projection step, a procedure that the IR binarization scheme critically relies on. Indeed tracking is necessary because the "best projection" rarely corresponds to the continuous minimizer of each weighted relaxation and using this point to calculate weights leads to poor binary solutions. Thus, even though our fast LADMM algorithm solves the linear relaxation of (3.14) when the homotopy parameter is 0, it is not useful for the IR scheme because its solution does not satisfy the reconstruction constraint until convergence, so each step would require the expensive $\Theta(K^2 n)$ projection.

Performance Benchmarks

We compare the performance of our algorithm to the IR scheme of [50] when used to compress subsets of the BeerAdvocate, 20 Newsgroups [53], and IMDb [41] datasets. All experiments are done on the same desktop computer[7] with both algorithms coded in C. The code for the IR algorithm is the original used in [50].

Table 3.1 shows the running times of both algorithms when used to find $K = 3$ and $K = 5$ grams. In all cases the algorithms found comparable solutions with similar objective values that were between 1.005 and 1.008 times larger than the lower bounds given by the solutions of the continuous relaxations. The homotopic procedure is consistently faster than the IR algorithm.

Figure 3.5 shows the performance of both algorithms in more detail on the BeerAdvocate dataset with $K = 3$. It plots the CFL objective value as a function of time for both algorithms and compares them to a lower bound provided by CFL's convex relaxation. Since neither algorithm is monotonic, we track the

[7] Intel Core i970 processor with 24GB of RAM

Table 3.1: Time Trials for Homotopic (H) and IR Algorithms

Dataset	Homotopy (s)	IR (s)
Beer $K = 3$	277	638
Beer $K = 5$	455	766
20 News $K = 3$	70	201
20 News $K = 5$	109	331
IMDb $K = 3$	119	518
IMDb $K = 5$	208	575

objective value of the current and best *binary* solutions. The circles mark the starting and ending points for both algorithms (IR's starting value runs off the y–axis) and both algorithms converge to an optimum that is 1.006 times larger than the relaxation. For reference, our algorithm requires 1748 iterations until convergence whereas IR necessitates 1729 steps for 8 rounds of reweighting. The "X" marks the point at which our algorithm has solved the LP relaxation to sufficient accuracy and starts increasing ζ. This transitions occurs at iteration 546 and shows that steps when $\zeta > 0$ are approximately twice as expensive as steps when $\zeta = 0$ – this is expected because we must encode the document corpus at every step where $\zeta > 0$. Nonetheless, our algorithm is twice as fast as the IR method.

Figure 3.5 depicts the large fluctuations the IR algorithm exhibits at the beginning of every reweighting round. IR relies critically on solution tracking to provide good weights for each round; poorly chosen weights cause the fluctuations to increase and prevent convergence. The graph also demonstrates how long the IR algorithm takes to find a reasonable binary solution. Its method for rounding a continuous solution sets all non–zero indicators to 1 and tends to create many redundant pointers. In contrast, our homotopic scheme relies on the encoding algorithm to round its solutions so it never creates redundant pointers. Our algorithm finds a good approximate solution in several seconds and refines it thereafter; it takes the IR algorithm over 350 seconds to find a comparable solution.

3.6 Experiments

We now present experiments investigating the usefulness of Dracula and CFL's features. We start with the shallow criterion as we were able to apply it to larger datasets using the binarization algorithms discussed Section 3.5. We explore datasets with both schemes, although the $\ell_1 \to \ell_\infty$ homotopy algorithm allowed

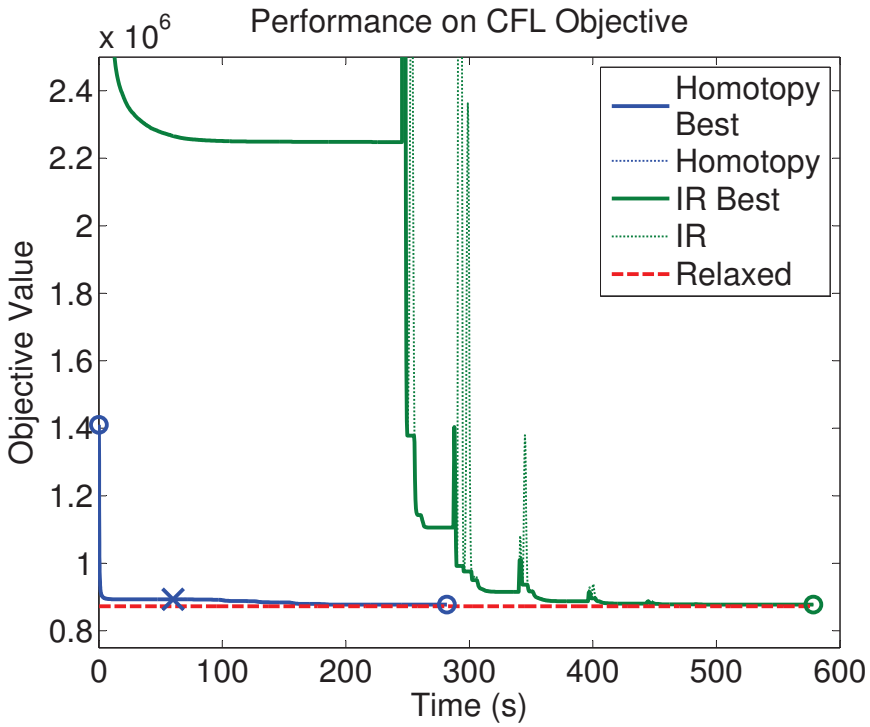


Figure 3.5: Performance trial comparing our homotopic method to the IR scheme of [50]. The red line denotes a lower bound for the objective. Circles indicate the starting/stopping points of the algorithms and the "X" marks the point at which we increase ζ from 0.

us to perform the largest scale experiments. In all of these experiments we fix all *document pointer costs* uniformly to a value λ, set all other costs identically to 1, and operate at the word level of granularity so that words are treated as characters. The majority of these shallow experiments are based on the following datasets:

20 Newsgroups Dataset The 20 Newsgroups dataset [34, 53] is a collection of about 19,000 messages approximately evenly split among 20 different newsgroups. Since each newsgroup discusses a different topic, some more closely related than others, we investigate our compressed features' ability to elucidate class structure in supervised and unsupervised learning scenarios. We use the "by-date" 60%/40% training/testing split described in [53] for all classification tasks. This split makes our results comparable to the existing literature and makes the task more difficult by removing correlations from messages that are responses to one another.

BeerAdvocate Dataset The BeerAdvocate dataset introduced in Section 2.6.1 consisting of $1,586,088$[8] beer reviews from $33,387$ users over 10 years. These reviews require over 1 gigabyte of memory to store in plaintext. Included with each review is a tag identifying its author; individual ratings (between 0 and 5) of the beer's appearance, aroma, palate, taste, and overall performance; and a timestamp.

3.6.1 Iterative Reweighting

All experiments were limited to using 5-grams as the maximum N-gram length. Each substring's dictionary cost was its word length and the pointer cost was uniformly set to $0 \leq \lambda \leq 5$. We found that an overly large λ hurts accuracy more than an overly small value since the former produces long, infrequent substrings, while the latter tends to a unigram representation. It is also worthwhile to note that the storage cost (i.e., the value of the objective function) of the binary solution was never more than 1.006 times the storage cost of the relaxed solution, indicating that we consistently found a good local optimum.

Finally, all classification tasks use an Elastic-Net–regularized logistic regression classifier implemented by `glmnet` [20]. Since this regularizer is a mix of L_1 and L_2 penalties, it is useful for feature selection but can also be used as a simple L_2 ridge penalty. Before training, we normalize each document by its L_1 norm and then normalize features by their standard deviation. We use this scheme so as to prevent overly long documents from dominating the feature normalization.

LZ77 Comparison Our first experiment demonstrates LZ77's sensitivity to document ordering on a simple binary classification task of predicting whether a document is from the alt.atheism (A) or comp.graphics (G) newsgroup. Features were computed by concatenating documents in different orders: (1) by class, i.e., all documents in A before those in G, or G before A; (2) randomly; (3) by alternating the class every other document. Figure 3.6 shows the testing error compared to features computed from our criterion. Error bars were estimated by bootstrapping the testing set 100 times, and all regularization parameters were chosen to minimize testing error while λ was fixed at 0.03. As discussed earlier, document ordering has a marked impact on performance, with the by-class and random orders performing significantly worse than the alternating ordering. Moreover, order invariance and the ability to tune the pointer cost lets our criterion select a better set of 5-grams.

[8] The original dataset has $1,586,259$ reviews but we threw away all reviews containing fewer than 10 words.

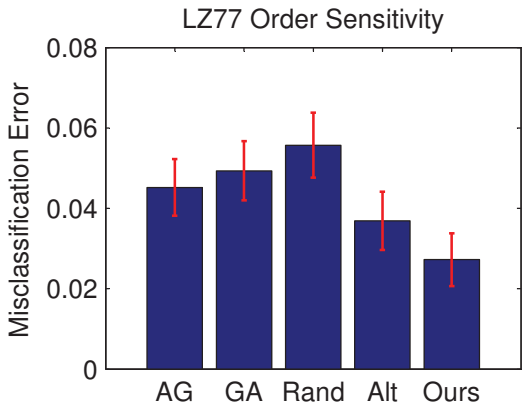


Figure 3.6: Misclassification error and standard error bars when classifying alt.atheism (A) vs. comp.graphics (G) from 20 Newsgroups. The four leftmost results are on features from running LZ77 on documents ordered by class (AG, GA), randomly (Rand), or by alternating classes (Alt); the rightmost is on our compressed features.

PCA Next, we investigate our features in a typical exploratory analysis scenario: a researcher looking for interesting structure by plotting all pairs of the top 10 principal components of the data. In particular, we verify PCA's ability to recover binary class structure for the A and G newsgroups, as well as multiclass structure for the A, comp.sys.ibm.pc.hardware (PC), rec.motorcycles (M), sci.space (S), and talk.politics.mideast (PM) newsgroups. Figure 3.7 plots the pair of principal components that best exemplifies class structure using (1) compressed features and (2) all 5-grams. For the sake of fairness, the components were picked by training a logistic regression on every pair of the top 10 principal components and selecting the pair with the lowest training error. In both the binary and multiclass scenarios, PCA is inundated by millions of features when using all 5-grams and cannot display good class structure. In contrast, compression reduces the feature set to tens of thousands (by two orders of magnitude) and clearly shows class structure. The star pattern of the five classes stands out even when class labels are hidden.

Table 3.2: Classification accuracy on the 20 Newsgroups and IMDb datasets

Method	20 Newsgroups	IMDb
Discriminative RBM [35]	76.2	—
Bag-of-Words SVM [33, 41]	80.8	88.2
Naïve Bayes [38]	81.8	—
Word Vectors [41]	—	88.9
All 5-grams	82.8	90.6
Compressed (our method)	83.0	90.4

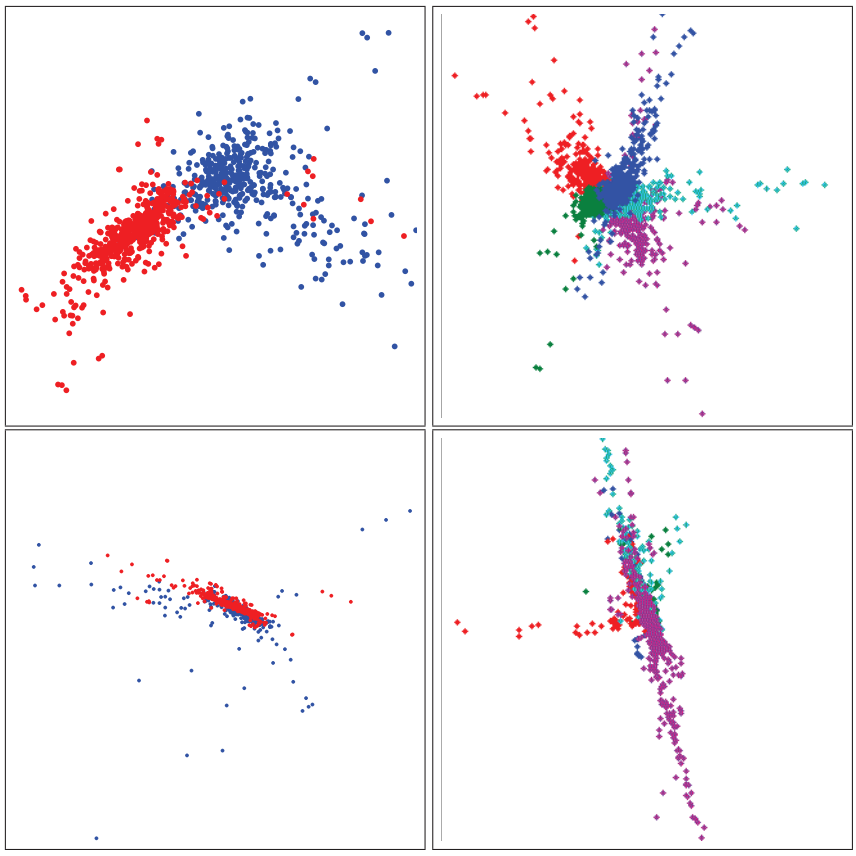
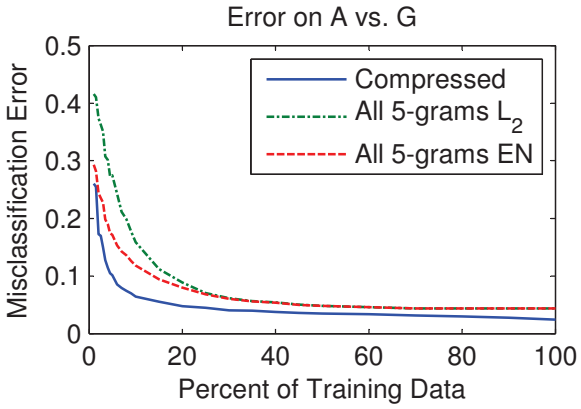


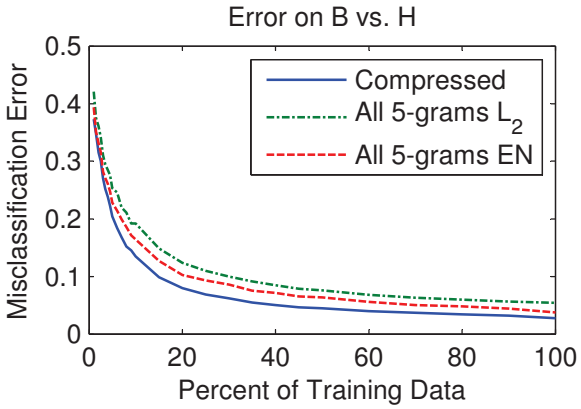
Figure 3.7: PCA plots for 20 Newsgroups. **Left:** alt.atheism (blue), comp.graphics (red). **Right:** alt.atheism (blue), comp.sys.ibm.pc.hardware (green), rec.motorcycles (red), sci.space (cyan), talk.politics.mideast (magenta). **Top:** compressed features (our method). **Bottom:** all 5-grams.

Classification Tasks Table 3.2 compares the performance of compressed features with all 5-grams on two tasks: (1) categorizing posts from the 20 Newsgroups corpus into one of 20 classes; (2) categorizing movie reviews collected from IMDb [41] into one of two classes (there are 25,000 training and 25,000 testing examples evenly split between the classes). For completeness, we include comparisons with previous work for 20 Newsgroups [35, 33, 38] and IMDb [41]. All regularization parameters, including λ, were chosen through 10-fold cross validation on the training set. We also did not L_1-normalize documents in the binary task because it was found to be counterproductive on the training set.

Our classification performance is state of the art in both tasks, with the compressed and all-5-gram features tied in performance. Since both datasets feature copious amounts of labeled data, we expect the 5-gram



(a)



(b)

Figure 3.8: Classification accuracy as the training set size varies for two classification tasks from 20 Newsgroups: **(a)** alt.atheism (A) vs. comp.graphics (G); **(b)** rec.sport.baseball (B) vs. rec.sport.hockey (H). To demonstrate the effects of feature selection, L_2 indicates L_2-regularization while EN indicates elastic-net regularization.

features to do well because of the power of the Elastic-Net regularizer. What is remarkable is that the compression retains useful features without using any label information. There are tens of millions of 5-grams, but compression reduces them to hundreds of thousands (by two orders of magnitude). This has a particularly noticeable impact on training time for the 20 Newsgroups dataset. Cross-validation takes 1 hour with compressed features and 8–16 hours for all 5-grams on our reference computer depending on the sparsity of the resulting classifier.

Training-Set Size Our final experiment explores the impact of training-set size on binary-classification accuracy for the A vs. G and rec.sport.baseball (B) vs. rec.sport.hockey (H) newsgroups. Figure 3.8 plots

testing error as the amount of training data varies, comparing compressed features to full 5-grams; we explore the latter with and without feature selection enabled (i.e., Elastic Net vs. L_2 regularizer). We resampled the training set 100 times for each training-set size and report the average accuracy. All regularization parameters were chosen to minimize the testing error (so as to eliminate effects from imperfect tuning) and $\lambda = 0.03$ in both tasks. For the A–G task, the compressed features require substantially less data than the full 5-grams to come close to their best testing error. The B–H task is harder and all three classifiers benefit from more training data, although the gap between compressed features and all 5-grams is widest when less than half of the training data is available. In all cases, the compressed features outperform the full 5-grams, indicating that that latter may benefit from even more training data. In future work it will be interesting to investigate the efficacy of compressed features on more intelligent sampling schemes such as active learning.

3.6.2 $\ell_1 \to \ell_\infty$ Homotopy

We compute a bag of trigrams representation for each review by running our algorithm on the entire dataset with a maximum N-gram length of 3. We vary λ over a grid of 10 values ranging from 0.01 to 3; Figure 3.2 shows the fraction of pointers that correspond to unigrams, bigrams, and trigrams for each λ. There are $45,408,597$ distinct features in the full trigram space and CFL produces a feature space two orders of magnitude smaller.

We use the following criteria when running our algorithm. The ADMM parameters ρ and μ are tuned as outlined in Section 3.5.3. We increase ζ by increments of 0.1 whenever both convergence parameters in (3.22) are below $\rho 10^{-3}$ or more than 150 steps have gone by since increasing ζ. However, when $\zeta = 0$, we always wait until the parameters in (3.22) reach our threshold so that the linear program is solved to reasonable accuracy. This configuration yields a reasonable balance between running time and finding a good local optimum: each value of λ takes about 10 hours to compute. It is worth noting that the relative duality gap, the ratio of the objective values of the binary and relaxed solutions, was always less than 1.01 which indicates that the algorithm is finding a good local optimum.

Author Identification

Our first task uses the author tags associated with each review as labels for an author identification task. We only allow authors with 10 or more reviews to participate, leaving 10,702 users. Three posts are randomly

selected from each author and set aside as a testing set; the remainder are used for training. The reviews in this testing set are further split by selecting 1,000 authors to act as a validation set that we use to tune the pointer cost λ.

This author identification task is a difficult multiclass classification problem with 10,702 classes: a random baseline achieves 0.009% accuracy. There are so many classes that both, glmnet [20] and liblinear [18], fail because of memory issues. We therefore use a simple 1–Nearest Neighbor classifier that represents each author as the centroid of his/her posts. An unknown author is classified by averaging the three samples of his/her writing and finding the nearest centroid. We use the validation set to select among our 10 compressed feature representations and normalize all features by their inverse document frequency.

Table 3.6 shows the testing accuracy of this approach when reviews are represented by their unigrams, (full) trigrams [9], and compressed features. CFL achieves nearly twice the accuracy of the unigram model, and the full trigram model performs the worst because it is inundated with spurious features. In addition, Figure 3.9 plots the testing and validation set accuracies for the CFL features as a function of λ. We also include unigrams in this graph since they correspond to $\lambda = 0$. The curve shows a clear preference for $\lambda = 1$ and its shape is akin to the regularization paths obtained from L_1/L_2 regularization. A possible explanation for this is that as λ increases, CFL favors larger N-grams that are more likely to be specific to (related) sets of documents when compared to their constituent unigrams. However, when λ is too large, CFL uses too many infrequent trigrams and documents become nearly incomparable because they have few features in common.

Table 3.3: Testing Accuracy on Author Identification Task

Baseline	**Unigrams**	**Trigrams**	**Compressed**
0.009 %	7.85 %	7.13 %	**15.1 %**

Rating and ABV Prediction

We also use the BeerAdvocate dataset to predict a beer's ABV and rating (along various criteria) from the text of its review. We treat all tasks as regression problems although ratings are always between 0 and 5. Reviews are randomly split into testing, training, and validation sets of sizes $500,000$; $986,259$; and

[9] This representation took over 12 hours to test.

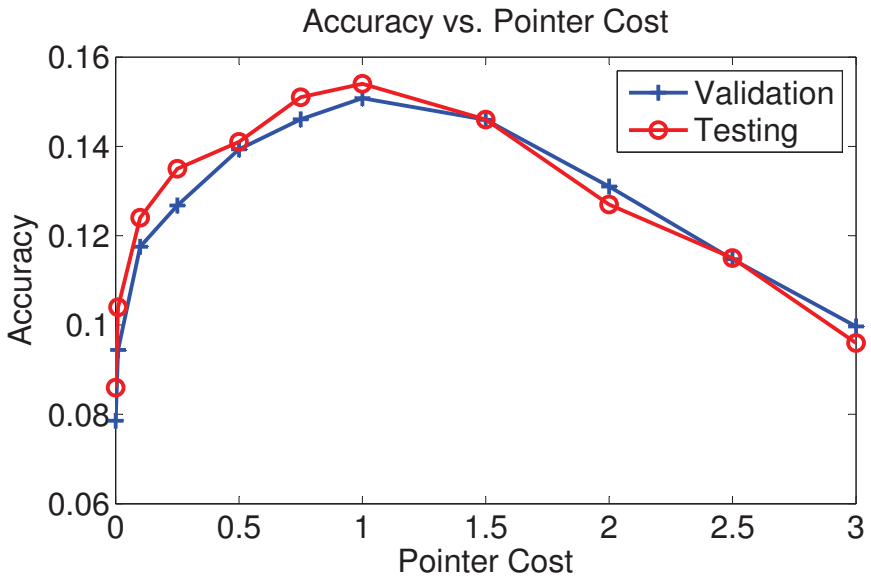


Figure 3.9: Testing and validation set accuracies of CFL features on author identification task as a function of the pointer cost. Unigrams correspond to $\lambda = 0$.

$100,000$, respectively, and the same splits are used for all tasks. We use glmnet to train an Elastic Net [20] on the training data and select all regularization parameters and the pointer cost for CFL through a grid search on the validation set. Table 3.4 compares using unigram or CFL features with a simple baseline that uses the mean of its training labels as a prediction. Text features clearly improve upon the baseline and CFL features outperform unigrams slightly, reducing the testing error between 4-8%.

Table 3.4: MSE when Predicting Rating or ABV

Task	Baseline	Unigrams	Compressed
Overall	0.52	0.29	**0.268**
Appearance	0.379	0.233	**0.221**
Aroma	0.486	0.264	**0.252**
Palate	0.467	0.263	**0.25**
Taste	0.536	0.261	**0.24**
ABV	5.393	2.397	**2.294**

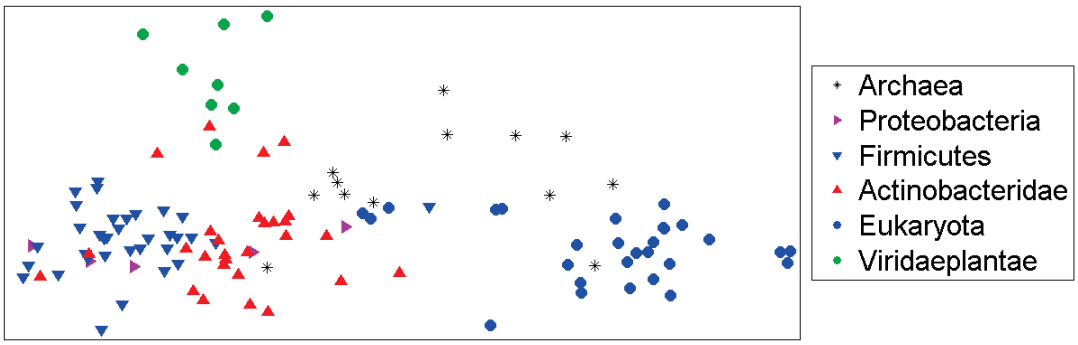


Figure 3.10: Proteins represented using the 4^{th} and 5^{th} singular vectors of Top features from Dracula.

3.6.3 Deep Compression

This section presents experiments comparing traditional BoN features with features derived from Dracula and CFL. Our primary goal is investigate whether deep compression can provide better features for learning than shallow compression or the traditional "fully redundant" BoN representation (using all N-grams up to a maximum length). Since any of these representations can be obtained from Dracula using an appropriate cost scheme, positive evidence for the deep compression implies Dracula is uncovering hierarchical structure which is simultaneously useful for compression and learning. We also provide a measure of compressed size that counts the number of pointers used by each representation, i.e. the result of evaluating each compression with a "common sense" space objective where all costs are 1. We use **Top** to indicate BoN features counting only document pointers (X in previous section), **Flat** for dictionary diffusion features (i.e. $\hat{X}$), **CFL** for BoN features from CFL, and **All** for traditional BoN features using all N-grams considered by Dracula.

We used Gurobi [23] to solve the refined LP relaxation of Dracula for all of our experiments. While Gurobi can solve impressively large LP's, encoding Dracula for a general-purpose solver is inefficient and limited the scale of our experiments. Dedicated algorithms that utilize problem structure, such as the network flow interpretation of the reconstruction modules, are the subject of follow-up work and will allow Dracula to scale to large-scale datasets. We limited our parameter tuning to the dictionary pointer cost λ (discussed in the solution path section) as this had the largest effect on performance. Experiments were performed with $\tau = 0$, $\alpha = 1$, a maximum N-gram length, and only on N-grams that appear at least twice in each corpus.

Protein Data We ran Dracula using 7-grams and $\lambda = 1$ on 131 protein sequences that are labeled with the kingdom and phylum of their organism of origin [2]. Bacterial proteins (73) dominate this dataset, 68 of

Table 3.5: Bacteria Identification Accuracy using Protein Data

SVD Rank	5	10	15	20	All	# Pointers
All	59.5	77.7	83.3	77.6	81.1	4.54×10^5
CFL	**89.7**	85.0	76.9	74.5	74.0	2.69×10^4
Top	87.5	**91.2**	**89.0**	83.3	84.3	1.76×10^4

which evenly come from Actinobacteria (A) and Fermicutes (F). The first 5 singular values (SV's) of the Top features show a clear separation from the remaining SV's and Figure 3.10 plots the proteins when represented by their 4^{th} and 5^{th} principle components. They are labeled by kingdom and, in more interesting cases, by phylum. Note the clear separation of the kingdoms, the two main bacterial phyla, and the cluster of plants separated from the other eukaryotes. Table 3.5 shows the average accuracy of two binary classification tasks in which bacteria are positive and we hold out either phylum A or F, along with other randomly sampled phyla for negative cases, as a testing set. We compare All features to Top features from Dracula and CFL using an ℓ_2-regularized SVM with $C = 1$. Since there are many more features than training examples we plot the effect of using the top K principle components of each feature matrix. Flat features did not help and performance strictly decreased if we limited the N-gram length for All features, indicating that long N-grams contain essential information. Both compression criteria perform well, but using a deep dictionary seems to help as Dracula's profile is more stable than CFL's.

Stylometry We extracted 100 sentences from each of the training and testing splits of the Reuters dataset [39] for 10 authors, i.e. $2,000$ total sentences, and replaced their words with part-of-speech tags. The goal of this task is to predict the author of a given set of writing samples (that all come from the same author). We make predictions by representing each author by the centroid of her 100 training sentences, averaging together the unknown writing samples, and reporting the nearest author centroid to the sample centroid. We ran Dracula on this representation with 10-grams and normalized centroids by their ℓ_1 norm and features by their standard deviation. Table 3.6 compares the performance of All features to Top features derived from various λ's for various testing sentence sample sizes. We report the average of $1,000$ trials, where each trial tested every author once and randomly selected a set of sample sentences from the testing split sentences. As in the protein data, neither Flat nor shorter N-gram features helped, indicating that higher order features

Table 3.6: Author Identification Accuracy

# Samples	5	10	25	50	75	# Pointers
All	36.0	47.9	67.9	80.6	86.4	5.01×10^5
CFL $\lambda = 20$	**39.6**	50.5	73.8	87.5	91.4	3.33×10^4
Top $\lambda = 1$	35.1	46.2	68.6	85.3	93.7	2.39×10^4
Top $\lambda = 10$	**39.6**	**51.0**	**75.0**	88.9	93.7	3.00×10^4
Top $\lambda = 20$	37.7	49.4	73.8	**91.5**	**97.8**	3.32×10^4

contain vital information. CFL with $\lambda = 20$ strictly dominated every other CFL representation and is the only one included for brevity. Dracula with $\lambda = 10$ or $\lambda = 20$ shows a clear separation from the other schemes, indicating that the deep compression finds useful structure.

Sentiment Prediction We use a dataset of $10,662$ movie review sentences [48] labeled as having positive or negative sentiment. Bigrams achieve state-of-the-art accuracy on this dataset and unigrams perform nearly as well [70], so enough information is stored in low order N-grams that the variance from longer N-grams hurts prediction. We ran Dracula using 5-grams to highlight the utility of Flat features, which focus the classifier onto lower order features. Following [70], Table 3.7 compares the 10-fold CV accuracy of a multinomial naïve-Bayes (NB) classifier using Top or Flat features with one using all N-grams up to a maximum length. The dictionary diffusion process successfully highlights relevant low order features and allows the Flat representation to be competitive with bigrams (the expected best performer). The table also plots the mean N-gram length (**MNL**) used by document pointers as a function of λ. The MNL decreases as λ increases and this eventually pushes the Top features to behave like a mix of bigrams and unigrams. Finally, we also show the performance of ℓ_2 or ℓ_1-regularized support vector machines for which we tuned the regularization parameter to minimize CV error (to avoid issues with parameter tuning). It is known that NB performs surprisingly well relative to SVMs on a variety of sentiment prediction tasks, so the dropoff in performance is expected. Both SVMs achieve their best accuracy with bigrams; the regularizers are unable to fully remove the spurious features introduced by using overly long N-grams. In contrast, Flat achieves its best performance with larger MNLs which suggests that Dracula performs a different kind of feature selection than is possible with direct ℓ_1/ℓ_2 regularization. Moreover, tuning λ combines feature selection with NB or any kind of classifier, irrespective of whether it natively performs feature selection.

Table 3.7: Sentiment Classification Accuracy

λ:	MNL	# Pointers	Top	Flat	N-gram Len.	NB All	SVM ℓ_1 All	SVM ℓ_2 All
0.25	**4.02**	1.79×10^5	73.9	78.2	**5**	77.9	76.6	76.9
0.5	**3.78**	1.75×10^5	75.1	**78.8**	**4**	77.9	76.8	77.0
1	**3.19**	1.71×10^5	76.6	78.2	**3**	78.4	77.0	77.2
2	**2.51**	1.71×10^5	78.0	78.1	**2**	**78.8**	77.2	77.5
5	**1.96**	1.86×10^5	78.0	78.0	**1**	78.0	76.3	76.5

Chapter 4

Conclusion

We have shown that learning with long N-grams on large text corpora is tractable because of the rich structure of suffix trees. The resulting matrix-vector multiplication algorithm is not only theoretically more efficient than sparse multiplication, but also practically substantially faster and more memory efficient. Our algorithm has broad implications for text based machine learning systems owing to the ubiquity of multiplication in machine learning; it can speed up any learning system based on any of the popular multiplication based algorithms like gradient descent. These computational savings also imply statistical efficiency by pruning out redundant features and equipping the resulting machine learning systems with a plurality of feature screening tools. The persistent storage framework that we have developed can viably store massive text repositories in a "machine learning optimized" format that facilitates rapid model development for arbitrary learning tasks. In view of the successes N-gram models have recently had in bioinformatics, it will be particularly interesting to apply our framework to the massive amounts of genomic information generated by next-generation sequencing methods. The computational savings of our routine enable a fundamentally new class of machine learning models for this domain, and it will be exciting to explore the "biological code" that defines life using these techniques.

We have also shown that compressed representations that minimize on disk storage space can directly provide good feature representations. Our proposed dictionary-based compressor, Dracula, is a novel combinatorial form of Deep Learning that can be expressed as an NP-Complete binary linear program and thereby analyzed by classical results from optimization and computer science. We use these perspectives to reveal that

Dracula is comprised of two easy combinatorial subproblems and then use these results to devise approximation algorithms based on iterative reweighting and homotopy techniques. Dracula's problem structure also allows us to gain statistical insights into the behavior of its solutions when the storage cost model it optimizes for changes. Critically, its solutions can be parameterized to change in predictable ways that are dictated by the combinatorial structure of its constraint polyhedron. This rich problem structure indicates that there is considerable potential to construct algorithms that will allow Dracula to scale to massive datasets. As this framework is built to uncover hierarchical structure, it is well suited to uncover motifs and combinatorial patterns in DNA and other largely compressible text sequences.

Our translation of string algorithms and combinatorial programming into the machine learning realm elucidates further connections between classical ideas from computer science and machine learning. Indeed, our advances can be viewed as bringing machine learning closer to a database technology. The resemblance is striking; data is initially preprocessed — be it by suffix tree or compression — to build an index whereupon the query language, i.e. learning paradigm, can be used to rapidly query the learning system for models. In this sense computer science provides guidance not only for how to rapidly learn, but also for how to structure machine learning systems. We believe that this database view of learning will become more relevant as the demands placed on machine learning grow.

Appendix A

Derivations for ADMM Subproblems

A.1 ADMM Derivations for Iterative Reweighting

We wish to minimize

$$\begin{aligned} \underset{w}{\text{minimize}} \quad & c^T w + \sum_{s \in \mathcal{C}^*} d_s \|P_{J(s)} w_{J(s)}\|_\infty \\ \text{subject to} \quad & Xw \geq \mathbf{1}, \quad w \geq \mathbf{0} \end{aligned} \tag{A.1}$$

For the sake of brevity, we take $P_{J(s)}$ to mean $P_{J(s)J(s)}$. With this in mind, the augmented Lagrangian of (A.1) is

$$c^T w + \sum_{s \in \mathcal{C}^*} d_s \|P_{J(s)} w_{J(s)}\|_\infty + I(w \geq \mathbf{0}) + I(Xz - \mathbf{1} \geq \mathbf{0}) + y^T(w - z) + \frac{\rho}{2}\|w - z\|_2^2 \tag{A.2}$$

A.1.1 Solving for w

The relevant parts of (A.2) for w are

$$c^T w + \sum_{s \in \mathcal{C}^*} d_s \|P_{J(s)} w_{J(s)}\|_\infty + I(w \geq \mathbf{0}) + y^T w + \frac{\rho}{2}\|w - z\|_2^2 \tag{A.3}$$

Notice that this separates out with respect to $w_{J(s)}$ so we can focus on each group separately. We therefore

drop subscripts and use w to refer to $w_{J(s)}$ and P to $P_{J(s)}$. The problem can be restated as

$$\begin{aligned} \underset{w}{\text{minimize}} \quad & c^T w + ct + y^T w + \frac{\rho}{2}\|w - z\|_2^2 \\ \text{subject to} \quad & w \geq \mathbf{0}, \quad Pw \leq t\mathbf{1} \end{aligned} \tag{A.4}$$

We have replaced the L_∞-norm via an epigraph variable transform. Note that we don't need a $t \geq 0$ constraint because it is implied by the existing ones. The Lagrangian is give by

$$\mathcal{L}(w, t, \alpha, \gamma) = c^T w + ct + y^T w + \frac{\rho}{2}\|w - z\|_2^2 - \gamma^T w + \alpha^T(Pw - t\mathbf{1}) \tag{A.5}$$

We have introduced dual variables α and γ to enforce the non-negativity and $Pw \leq t\mathbf{1}$ constraints, respectively. Taking derivatives w.r.t. t yields

$$\frac{\delta \mathcal{L}}{\delta t} = c - \alpha^T \mathbf{1} \tag{A.6}$$

If $c - \alpha^T \mathbf{1} \neq 0$ then we can set t such that the above is arbitrarily negative. Therefore, we assume that $c = \alpha^T 1$. This leads to the problem

$$\mathcal{L}(w, \alpha, \gamma) = c^T w + y^T w + \frac{\rho}{2}\|w - z\|_2^2 - \gamma^T w + \alpha^T P w \text{ subject to } c = \alpha^T 1 \tag{A.7}$$

Next, the derivative w.r.t. w is

$$\frac{\delta \mathcal{L}}{\delta w} = c + y + \rho(w - z) - \gamma + P\alpha \tag{A.8}$$

which implies

$$w = \rho^{-1}(\rho z - c - y + \gamma - P\alpha) \tag{A.9}$$

Strong duality holds and the KKT conditions imply that $\gamma, \alpha \geq 0$ with $w^T \gamma = 0$ and $\alpha_i = 0$ if $P_i w_i < \|Pw\|_\infty$. Thus, unless $w = 0$, $\gamma^T \alpha = 0$. To check whether $w = 0$, we plug into (A.9) and check whether

$$0 = \rho z - c - y + \gamma - P\alpha \tag{A.10}$$

with $c = \alpha^T 1$. Let $q = (\rho z - c - y)_+$ where $(x)_+ = \max(0, x)$ applies element-wise. Since γ can add

arbitrarily positive amounts, (A.10) is equivalent to

$$0 = q - P\alpha \tag{A.11}$$

With the looser restriction $c \geq \alpha^T 1$. Thus, $P^{-1}q = \alpha$ and so

$$w = 0 \Leftrightarrow 1^T P^{-1} q \leq c \tag{A.12}$$

Next, assume that $w \neq 0$. Then to find α, let $f = Pq$ and suppose that f is sorted in *decreasing* order and that w, P, α are also sorted so that indices match up. This is always possible by permuting the vectors. Then w_1 is maximal iff

$$P_{11} w_1 = f_1 - P_{11}^2 c \geq f_2 = P_{22} w_2 \tag{A.13}$$

If w_1, w_2 are maximal, then

$$f_1 - P_{11}^2 \alpha_1 = f_2 - P_{22}^2 (c - \alpha_1) \geq f_3 \tag{A.14}$$

Solving for α_1 we find

$$\alpha_1 = \frac{f_1 - f2 + P_{22}^2 c}{P_{11}^2 + P_{22}^2} \tag{A.15}$$

And hence

$$f_1 - P_{11}^2 \frac{f_1 - f_2 + P_{22}^2 c}{P_{11}^2 + P_{22}^2} = \frac{P_{22}^2 f_1 + P_{11}^2 f_2 + P_{11}^2 P_{22}^2 c}{P_{11}^2 + P_{22}^2} \geq f_3 \tag{A.16}$$

Continuing on, suppose that w_1, w_2, w_3 are maximal so that

$$f_1 - P_{11}^2 \alpha_1 = f_2 - P_{22}^2 \alpha_2 = f_3 - P_{33}^2 (c - \alpha_1 - \alpha_2) \geq f_4 \tag{A.17}$$

Solving that for α_1 we find

$$\alpha_1 = \frac{f_1 - f_2 + P_{22}^2 \alpha_2}{P_{11}^2} \tag{A.18}$$

And then for α_2

$$\alpha_2 = \frac{P_{11}^2(f_2 - f_3) - P_{33}^2(f_1 - f_2) + P_{11}^2 P_{33}^2 c}{P_{11}^2 P_{22}^2 + P_{11}^2 P_{33}^2 + P_{22}^2 P_{33}^2} \tag{A.19}$$

Thus, the maximal elements are given by

$$f_2 - P_{22}^2 \alpha_2 = \frac{P_{11}^{-2} f_1 + P_{22}^{-2} f_2 + P_{33}^{-2} f_3 - c}{P_{11}^{-2} + P_{22}^{-2} + P_{33}^{-2}} \tag{A.20}$$

It can be shown by induction that there are k maximal elements only if

$$\frac{\sum_{i=1}^{k} P_{ii}^{-1} q_i - c}{\sum_{i=1}^{k} P_{ii}^{-2}} \geq P_{k+1,k+1} q_{k+1} \tag{A.21}$$

In order to recover w, we set $w_i = q_i$ if $P_{ii} w_i$ is not maximal, and if it is, we set $w_i = P_{ii}^{-1} \frac{\sum_{j=1}^{k} P_{jj}^{-1} q_j - c}{\sum_{j=1}^{k} P_{jj}^{-2}}$.

Next, we show that it is possible to quickly find k in linear time (i.e. without sorting). Define $m(k) = \frac{\sum_{i=1}^{k} P_{ii}^{-2} f_i - c}{\sum_{i=1}^{k} P_{ii}^{-2}}$ and suppose that there are k true maximal elements so that

$$m(k) > f_{k+1} \tag{A.22}$$

We show that $m(t) \geq f_{t+1}$, when $t > k$. Using the fact that $m(k) > f_{t+1}$,

$$m(k) > f_{t+1} \Leftrightarrow \sum_{i=1}^{k} P_{ii}^{-2} f_i - c > \left(\sum_{i=1}^{k} P_{ii}^{-2} \right) f_{t+1} \tag{A.23}$$

$$\Leftrightarrow \sum_{i=1}^{k} P_{ii}^{-2} f_i - c + \sum_{i=k+1}^{t} P_{ii}^{-2} f_i > \left(\sum_{i=1}^{k} P_{ii}^{-2} \right) f_{t+1} + \left(\sum_{i=k+1}^{t} P_{ii}^{-2} \right) f_{t+1} \tag{A.24}$$

$$\Leftrightarrow \sum_{i=1}^{t} P_{ii}^{-2} f_i - c > \left(\sum_{i=1}^{t} P_{ii}^{-2} \right) f_{t+1} \Leftrightarrow m(t) > f_{t+1} \tag{A.25}$$

Thus $m(t) \leq f_{t+1}$ for $t < k$ and $m(t) > f_{t+1}$ for $t \geq k$.

We can use this as a search criteria to develop an algorithm akin to the linear time median finding algorithm. This allows us to find k in linear time without requiring that f be sorted.

A.1.2 Solving for z

For z the relevant parts are

$$\begin{aligned} \underset{z}{\text{minimize}} \quad & -y^T z + \frac{\rho}{2}\|w - z\|_2^2 \\ \text{subject to} \quad & Xz \geq \mathbf{1} \end{aligned} \tag{A.26}$$

This is easiest to solve by taking the dual. The Lagrangian is given by

$$\mathcal{L}(z, \alpha) = -y^T z + \frac{\rho}{2}\|w - z\|_2^2 + \alpha^T(\mathbf{1} - Xz) \tag{A.27}$$

Solving for z we find

$$\frac{\delta \mathcal{L}}{\delta z} = -y - \rho w + \rho z - X^T \alpha = 0 \tag{A.28}$$

$$z = \rho^{-1}(y + \rho w + X^T \alpha) \tag{A.29}$$

Strong duality obtains, so plugging (A.29) into the Lagrangian yields the dual optimization problem

$$\begin{aligned} \underset{\alpha}{\text{minimize}} \quad & -\left(\rho 1 - X(y + \rho w)\right)^T \alpha + \frac{1}{2}\alpha^T H \alpha \\ \text{subject to} \quad & \alpha \geq \mathbf{0} \end{aligned} \tag{A.30}$$

where $H = XX^T$.

A.1.3 Matrix Entries

This section explores the structure of $H = XX^T$. We assume that N documents are compressed jointly, each of size n_i, and that pointers respect document boundaries. We show that H is a $(k-1)$-banded matrix and that it is block diagonal with N blocks, each of size $n_i \times n_i$ and corresponding to document i. This structure occurs when we assume a specific ordering for the set of potential pointers $\mathcal{P}$. In particular, pointers

are ordered lexicographically according to the document they pertain to, then the length of their substring, and finally the location in which they insert their substring.

Recall that column j of X corresponds to pointer $p_j \in \mathcal{P}$ and that this column only has 1's at locations corresponding to words that p_j can reconstruct. Let $m_i = \sum_{t=1}^{k} n_i - t + 1$ be the total number of pointers pertaining to document i. Since pointers respect document boundaries, our ordering implies that X is a block diagonal matrix in which columns $1, \ldots, m_1$ can only have $1's$ in rows $1, \ldots, n_1$; columns $m_1+1, \ldots, m_1+m_2$ can only have 1's in rows $n_1+1, \ldots, n_1+n_2$; and so on. This immediately implies that H is also a block diagonal matrix comprised of N blocks, each of size $n_i \times n_i$ with the i^{th} block corresponding to document i.

Next, to show that H is $(k-1)$-banded, notice that each column of X has a contiguous sequence of at most k ones and is 0 everywhere else. The outer product $XX^T = \sum_{i=1}^{m} X_i X_i^T$ where X_i is the i^{th} column of X is therefore formed by adding together a series of rank one matrices, each of which is $(k-1)$-banded. This implies that H must itself be $(k-1)$-banded.

A.2 Linearized ADMM Derivations for $\ell_1 \to \ell_\infty$ Homotopy

We show how to minimize the augmented Lagrangian

$$\mathcal{L}_\rho(w, \gamma, y) = h(w) + g(\gamma) + y^T(Aw - \gamma) + \frac{\rho}{2}\|Aw - \gamma\|_2^2 \tag{A.31}$$

Note that $\gamma = \begin{bmatrix} z \\ \theta \end{bmatrix}$ and hence $y = \begin{bmatrix} y^{(z)} \\ y^{(\theta)} \end{bmatrix}$ are partitioned variables. We use the following definitions for h, g, and A:

$$\begin{aligned} h(w) &= w^T c + \sum_{s \in \mathcal{C}^*} d_s \|w_{J(s)}\|_\infty + I\{w \geq 0\} \\ g(z, \theta) &= I\{z \geq 1\} + I\left\{\left\|\theta_{T(i)}\right\|_\infty \geq \zeta \; \forall i = 1, \ldots, n\right\} \\ A &= \begin{bmatrix} X \\ I_m \end{bmatrix} \end{aligned} \tag{A.32}$$

We use the notation $I\{\bullet\}$ as an indicator function that is 0 if the condition insides the braces is met and is ∞ otherwise. Starting with w, recall that we use the linearized form of the Lagrangian. The relevant parts of the

optimization problem are

$$w^T c + \sum_{s \in \mathcal{C}^*} d_s \left\| w_{J(s)} \right\|_\infty + y^{(z)T} X w + y^{(\theta)T} w + \rho (X\bar{w} - z)^T X w + \frac{\rho}{2} \left\| w - \theta \right\|_2^2 + \frac{\mu}{2} \left\| w - \bar{w} \right\|_2^2 \quad \text{(A.33)}$$

$$\text{subject to } w \geq 0$$

We collect all linear terms and complete the square to obtain the equivalent formulation

$$\sum_{s \in \mathcal{C}^*} d_s \left\| w_{J(s)} \right\|_\infty + \frac{\mu + \rho}{2} \left\| w - q \right\|_2^2 \quad \text{(A.34)}$$

$$q = -\frac{1}{\mu + \rho} \left(c + y^{(z)T} X + y^{(\theta)} + \rho X^T (X\bar{w} - z) - \mu \bar{w} - \rho \theta \right) \quad \text{(A.35)}$$

Next, the relevant terms for γ are

$$- \begin{bmatrix} z^T & \theta^T \end{bmatrix} \begin{bmatrix} y^{(z)} \\ y^{(\theta)} \end{bmatrix} + \frac{\rho}{2} \left\| \begin{bmatrix} Xw \\ w \end{bmatrix} - \begin{bmatrix} z \\ \theta \end{bmatrix} \right\|_2^2 \quad \text{(A.36)}$$

$$\text{subject to } z \geq 1$$

$$\left\| \theta_{T(i)} \right\|_\infty \geq \zeta \; \forall i = 1, \ldots, n \quad \text{(A.37)}$$

This function is clearly separable in z and θ. It simply projects $Xw + \rho^{-1} y^{(z)}$ to have all entries ≥ 1 and hence $z = \max(Xw + \rho^{-1} y^{(z)}, 1)$. The solution for θ is discussed in the paper.

A.2.1 Additional Structure in X

We assume that a collection of N documents of sizes $n_1, \ldots, n_N$ is being compressed and define $n = \sum_{i=1}^N n_i$. Assuming that we allow all N-grams that respect document boundaries as potential pointers, X has special structure. Note that there are $m_j = \sum_{i=1}^K (n_j - i + 1)$ potential pointers for document j and that $X \in \{0, 1\}^{n \times m}$ where $m = \sum_{i=1}^N m_i$. Each column in this matrix corresponds to a particular potential pointer in $\mathcal{P}$ and we are free to select how to order the pointers and hence columns of X. An efficient way to do this is to let the first m_1 columns correspond to the pointers for document 1, the next m_2 columns to the

pointers for document 2, and so on. X then becomes a block diagonal matrix

$$X = \begin{bmatrix} X^{(1)} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & X^{(N)} \end{bmatrix} \tag{A.38}$$

(where $X^{(i)} \in \{0,1\}^{n_i \times m_i}$ corresponds to D_i) because the pointers for document i cannot be used to reconstruct any other documents (because they respect document boundaries). It is easy to see that with this order,

$$XX^T = \begin{bmatrix} X^{(1)}X^{(1)T} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & X^{(N)}X^{(N)T} \end{bmatrix} \tag{A.39}$$

is a block diagonal positive semidefinite matrix.

We further order the columns within each $X^{(i)}$ by ordering the pointers according to size first and then starting location. Thus, column j for $1 \leq j \leq n_i$ corresponds to the pointer for the unigram at position j. Column j for $n_i + 1 \leq j \leq 2n_i - 1$ corresponds to the pointer for the bi-gram that starts at position $j - n_i$, and so on. When $K = 2$, for example, $X^{(i)}$ looks like

$$X^{(i)} = \begin{bmatrix} I_{n_i} & A^{(i2)} \end{bmatrix} \tag{A.40}$$

where I_{n_i} is the $n_i \times n_i$ identity and $A^{(i2)} \in \{0,1\}^{n_i \times (n_i - 1)}$ with column k a vector of all 0's except for two 1's in positions k and $k + 1$. If we continue this line of reasoning, we see that $X^{(i)}$ can be expressed as K block matrices

$$X^{(i)} = \begin{bmatrix} A^{(i1)} & \dots & A^{(iK)} \end{bmatrix} \tag{A.41}$$

where $A^{(ik)} \in \{0,1\}^{n_i \times (n_i - k + 1)}$. Column j of $A^{(ik)}$ is all 0's except for a contiguous string of k 1's starting at position j. We will use the notation $e^{(jk)}$ to denote column j in $A^{(ik)}$ (we have dropped reference to the document number for brevity). An immediate consequence of this representation is that vector multiplication

by $X^{(i)}$ and $X^{(i)T}$ is very fast. For instance, the product

$$X^{(i)T}w = \sum_{k=1}^{K} A^{(ik)T}w \tag{A.42}$$

and multiplying by $A^{(ik)T}$ amounts to convolving w with a signal of k 1's and is easily performed in $\Theta(n_i)$ operations. The overall running time is therefore $O(Kn_i)$ for matrix-vector multiplication by $X^{(i)T}$ and a similar $O(Kn_i)$ algorithm can be obtained for multiplication by $X^{(i)}$.

We can now express the product $B^{(i)} = X^{(i)}X^{(i)T}$ as

$$B^{(i)} = \sum_{k=1}^{K} \sum_{i=1}^{n_i-k+1} e^{(jk)}e^{(jk)T} \tag{A.43}$$

or, equivalently, as sum of squares of 1's of side lengths $1, \ldots, K$ whose upper left corner is positioned along the main diagonal. We will call these squares k-squares.

To start, $B^{(i)}$ is symmetric because it is a covariance matrix so we only consider its upper triangle. We start with the "middle" entries, assuming that $n_i > 2k-2$. Then $B^{(i)}_{st}$ for $t \geq s$ and $s \geq k$ can be expressed as an appropriate sum of 1's. Note that if $t = s+1$, a 1-square cannot contribute to the entry. Extending this reasoning to the general case, we see that if $z = t-s$, then only $z+1, \ldots, K$ squares can contribute to $B^{(i)}_{st}$. This implies that $B^{(i)}_{st} = 0$ if $t \geq s+K$, i.e. each $B^{(i)}$ and hence XX^T is $K-1$ banded and symmetric.

Next, assuming $k \in z+1, \ldots, K$, a k-square whose upper left corner is in row j can only contribute if it is non-zero at position (s,t). This happens when $s-j+1 \leq k$ and $t-j+1 \leq k$, i.e. $j \geq s-k+1$ and $j \geq t-k+1$. Since $t \geq s$, we only need to check the second inequality. Finally, we also know that $j \leq s$, and so our entry can be expressed as

$$B^{(i)}_{st} = \sum_{k=z+1}^{K} \sum_{i=t-k+1}^{s} 1 = \sum_{k=z+1}^{K} (k-z) = \sum_{k=1}^{K-z} (k+z-z) = \frac{(K-z)(K-z+1)}{2} \tag{A.44}$$

Next, suppose that $t \geq s$ and $1 \leq s < k$. The outer summation stays the same, but the inner one must account for when $t-k+1 < 1$. In those cases, the inner summation contributes only s instead of $k-z$.

This situation happens when $k > t$, so we divide the summation into

$$B_{st}^{(i)} = \sum_{k=z+1}^{t'} (k-z) + \sum_{k=t'+1}^{K} s = \sum_{k=1}^{t'-z} k + s(K-t') = s(K-t') = \frac{(t'-z)(t'-z+1)}{2} \tag{A.45}$$

where $t' = \min(K, t)$. Finally, our matrix is not only symmetric but also symmetric with respect to its minor diagonal. This can be seen from redoing all of our formulas using the bottom right of each square rather than the top left.

It is easy to see now that $B^{(i)}$ is nearly Toeplitz. Indeed, if we chop off the top and bottom $K-1$ rows, this is the case. The sum of each row of this Toeplitz matrix can be expressed as

$$\frac{K(K+1)}{2} + 2\sum_{z=1}^{K-1} \frac{(K-z)(K-z+1)}{2} = \frac{K^3}{3} + \frac{K^2}{2} + \frac{K}{6} \tag{A.46}$$

In addition, it is easy to see that each of the top and bottom rows we removed must sum to an integer less than $\frac{K^3}{3} + \frac{K^2}{2} + \frac{K}{6}$ since each entry in these rows has fewer k-squares added to it than the rows in the middle.

A sample 12×12 matrix $B^{(i)}$ with $K = 5$ is shown below:

$$\begin{matrix}
5 & 4 & 3 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 9 & 7 & 5 & 3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 7 & 12 & 9 & 6 & 3 & 1 & 0 & 0 & 0 & 0 & 0 \\
2 & 5 & 9 & 14 & 10 & 6 & 3 & 1 & 0 & 0 & 0 & 0 \\
1 & 3 & 6 & 10 & 15 & 10 & 6 & 3 & 1 & 0 & 0 & 0 \\
0 & 1 & 3 & 6 & 10 & 15 & 10 & 6 & 3 & 1 & 0 & 0 \\
0 & 0 & 1 & 3 & 6 & 10 & 15 & 10 & 6 & 3 & 1 & 0 \\
0 & 0 & 0 & 1 & 3 & 6 & 10 & 15 & 10 & 6 & 3 & 1 \\
0 & 0 & 0 & 0 & 1 & 3 & 6 & 10 & 14 & 9 & 5 & 2 \\
0 & 0 & 0 & 0 & 0 & 1 & 3 & 6 & 9 & 12 & 7 & 3 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 3 & 5 & 7 & 9 & 4 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 3 & 4 & 5
\end{matrix} \tag{A.47}$$

Bibliography

[1] Memex human trafficking - deepdive. `http://deepdive.stanford.edu/showcase/memex`. Accessed: 2016-10-30.

[2] Protein classification benchmark collection. http://hydra.icgeb.trieste.it/benchmark/index.php?page=00.

[3] Mohamed Ibrahim Abouelhoda, Stefan Kurtz, and Enno Ohlebusch. Replacing suffix trees with enhanced suffix arrays. *J. of Discrete Algorithms*, 2(1):53–86, March 2004.

[4] Manda Banerji, Ofer Lahav, Chris J Lintott, Filipe B Abdalla, Kevin Schawinski, Steven P Bamford, Dan Andreescu, Phil Murray, M Jordan Raddick, Anze Slosar, et al. Galaxy zoo: reproducing galaxy morphologies via machine learning. *Monthly Notices of the Royal Astronomical Society*, 406(1):342–353, 2010.

[5] Dario Benedetto, Emanuele Caglioti, and Vittorio Loreto. Language trees and zipping. *PRL*, 88(4):048702, 2002.

[6] Dimitris Bertsimas and Robert Weismantel. *Optimization over integers.* Athena Scientific, 2005.

[7] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

[8] Andrej Bratko, Bogdan Filipič, Gordon V. Cormack, Thomas R. Lynam, and Blaž Zupan. Spam filtering using statistical data compression models. *JMLR*, 7:2673–2698, 2006.

[9] Eric Brown, Eddie Epstein, J William Murdock, and Tong-Haing Fin. Tools and methods for building watson.

[10] Peter Brucker. An $O(n)$ algorithm for quadratic knapsack problems. *Operations Research Letters*, 3(3):163–166, 1984.

[11] Emmanuel Candes, Michael Wakin, and Stephen Boyd. Enhancing sparsity by reweighted ℓ_1 minimization. *J Fourier Analysis and Applications*, 14(5-6):877–905, 2008.

[12] William Cavnar and John Trenkle. N-gram-based text categorization. In *In Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 161–175, 1994.

[13] John D Chodera, Nina Singhal, Vijay S Pande, Ken A Dill, and William C Swope. Automatic discovery of metastable states for the construction of markov models of macromolecular conformational dynamics. *The Journal of chemical physics*, 126(15):155101, 2007.

[14] Rudi Cilibrasi and Paul MB Vitányi. Clustering by compression. *TIT*, 51(4):1523–1545, 2005.

[15] F.H. Clarke. *Optimization and nonsmooth analysis*. Canadian Mathematical Society series of monographs and advanced texts. Wiley, 1983.

[16] Sajal K Das, Diane J Cook, Amiya Battacharya, Edwin O Heierman, and Tze-Yun Lin. The role of prediction algorithms in the mavhome smart home architecture. *IEEE Wireless Communications*, 9(6):77–84, 2002.

[17] J.W. Demmel. *Applied Numerical Linear Algebra*. Miscellaneous Bks. Society for Industrial and Applied Mathematics, 1997.

[18] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

[19] E. Frank, C. Chui, and I.H. Witten. Text categorization using compression models. Technical Report 00/02, University of Waikato, Department of Computer Science, 2000.

[20] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. *J Stat Softw*, 33(1):1–22, 2010.

[21] Evgeniy Gabrilovich and Shaul Markovitch. Text categorization with many redundant features: Using aggressive feature selection to make SVMs competitive with C4.5. In *ICML*, 2004.

[22] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[23] Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2015.

[24] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997.

[25] Trevor Hastie, Robert Tibshirani, and J. H. Friedman. *The elements of statistical learning: data mining, inference, and prediction: with 200 full-color illustrations*. New York: Springer-Verlag, 2001.

[26] John Houvardas and Efstathios Stamatatos. N-gram feature selection for authorship identification. In *Proceedings of the 12th International Conference on Artificial Intelligence: Methodology, Systems, and Applications*, AIMSA'06, pages 77–86, Berlin, Heidelberg, 2006. Springer-Verlag.

[27] Laurent Hyafil and Ronald L Rivest. Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, 5(1):15–17, 1976.

[28] Juhani Kähärä and Harri Lähdesmäki. Evaluating a linear k-mer model for protein-dna interactions using high-throughput selex data. *BMC Bioinformatics*, 14(S-10):S2, 2013.

[29] Toru Kasai, Hiroki Arimura, and Setsuo Arikawa. Efficient substring traversal with suffix arrays.

[30] Eamonn Keogh, Stefano Lonardi, and Chotirat Ann Ratanamahatana. Towards parameter-free data mining. In *KDD*, 2004.

[31] Dongmin Kim, Suvrit Sra, and Inderjit S Dhillon. Tackling box-constrained optimization via a new projected quasi-newton approach. *SIAM Journal on Scientific Computing*, 32(6):3548–3563, 2010.

[32] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[33] Man Lan, ChewLim Tan, and HweeBoon Low. Proposing a new term weighting scheme for text categorization. In *AAAI*, 2006.

[34] Ken Lang. Newsweeder: Learning to filter netnews. In *ICML*, 1995.

[35] Hugo Larochelle and Yoshua Bengio. Classification using discriminative restricted Boltzmann machines. In *ICML*, 2008.

[36] Yann LeCun, Sumit Chopra, Raia Hadsell, Marc'Aurelio Ranzato, and Fu-Jie Huang. A tutorial on energy-based learning. In G. Bakir, T. Hofman, B. Schölkopf, A. Smola, and B. Taskar, editors, *Predicting Structured Data*. MIT Press, 2006.

[37] Honglak Lee, Peter Pham, Yan Largman, and Andrew Y Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In *Advances in neural information processing systems*, pages 1096–1104, 2009.

[38] Baoli Li and Carl Vogel. Improving multiclass text classification with error-correcting output coding and sub-class partitions. In *Can Conf Adv Art Int*, 2010.

[39] Zhi Liu. Reuter 50 50 data set.

[40] Shu Lu and Stephen M Robinson. Normal fans of polyhedral convex sets. *Set-Valued Analysis*, 16(2-3):281–305, 2008.

[41] Andrew Maas, Raymond Daly, Peter Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *ACL*, 2011.

[42] Julian McAuley and Jure Leskovec. From amateurs to connoisseurs: Modeling the evolution of user expertise through online reviews. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 897–908, Republic and Canton of Geneva, Switzerland, 2013. International World Wide Web Conferences Steering Committee.

[43] Julian McAuley and Jure Leskovec. Hidden factors and hidden topics: understanding rating dimensions with review text. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 165–172. ACM, 2013.

[44] Michael Montemerlo, Jan Becker, Suhrid Bhat, Hendrik Dahlkamp, Dmitri Dolgov, Scott Ettinger, Dirk Haehnel, Tim Hilden, Gabe Hoffmann, Burkhard Huhnke, et al. Junior: The stanford entry in the urban challenge. *Journal of field Robotics*, 25(9):569–597, 2008.

[45] Yuta Mori. sais, https://sites.google.com/site/yuta256/sais, 2015.

[46] U.S. National Library of Medicine. The cancer genome atlas, 2015.

[47] U.S. National Library of Medicine. National center for biotechnology information, 2015.

[48] Bo Pang and Lillian Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 115–124. Association for Computational Linguistics, 2005.

[49] Neal Parikh and Stephen Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 2013.

[50] Hristo Paskov, Robert West, John Mitchell, and Trevor Hastie. Compressive feature learning. In *NIPS*, 2013.

[51] 1000 Genomes Project. 1000 genomes project, a deep catalog of human genetic variation., 2008.

[52] Carl Edward Rasmussen. Gaussian processes for machine learning. 2006.

[53] Jason Rennie. 20 Newsgroups dataset, 2008. `http://qwone.com/~jason/20Newsgroups` (accessed May 31, 2013).

[54] Konrad Rieck and Pavel Laskov. Linear-time computation of similarity measures for sequential data. *The Journal of Machine Learning Research*, 9:23–48, 2008.

[55] Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[56] Ruslan Salakhutdinov. Learning deep generative models, 2009.

[57] Bernhard Scholkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. 2001.

[58] A. Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency*. Springer, 2003.

[59] D. Sculley and Carla E. Brodley. Compression and machine learning: A new perspective on feature space vectors. In *DCC*, 2006.

[60] Margaret A Shipp, Ken N Ross, Pablo Tamayo, Andrew P Weng, Jeffery L Kutok, Ricardo CT Aguiar, Michelle Gaasenbeek, Michael Angelo, Michael Reich, Geraldine S Pinkus, et al. Diffuse large b-cell lymphoma outcome prediction by gene-expression profiling and supervised machine learning. *Nature medicine*, 8(1):68–74, 2002.

[61] Richard Socher and Christopher D. Manning. Deep learning for NLP (without magic). In *Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings*, pages 1–3, 2013.

[62] Matt Taddy. Document classification by inversion of distributed language representations. *arXiv preprint arXiv:1504.07295*, 2015.

[63] Adi L Tarca, Vincent J Carey, Xue-wen Chen, Roberto Romero, and Sorin Drăghici. Machine learning and its applications to biology. *PLoS Comput Biol*, 3(6):e116, 2007.

[64] Choon Hui Teo and S. V. N. Vishwanathan. Fast and space efficient string kernels using suffix arrays. In *In Proceedings, 23rd ICMP*, pages 929–936. ACM Press, 2006.

[65] Robert Tibshirani. Regression shrinkage and selection via the lasso. *J. R. Stat. Soc. B*, 58(1):267–288, 1996.

[66] Robert Tibshirani, Jacob Bien, Jerome Friedman, Trevor Hastie, Noah Simon, Jonathan Taylor, and Ryan J. Tibshirani. Strong rules for discarding predictors in lasso-type problems., 2010.

[67] R. S. Varga. *Gershgorin and His Circles*. ser. Springer Series in Computational Mathematics. Berlin, Germany: Springer-Verlag, 2004, no. 36.

[68] Kévin Vervier, Pierre Mahé, Maud Tournoud, Jean-Baptiste Veyrieras, and Jean-Philippe Vert. Large-scale machine learning for metagenomics sequence classification. *Bioinformatics*, 32(7):1023–1032, 2016.

[69] SVN Vishwanathan and Alexander Johannes Smola. Fast kernels for string and tree matching. *Kernel methods in computational biology*, pages 113–130, 2004.

[70] Sida Wang and Christopher D Manning. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2*, pages 90–94. Association for Computational Linguistics, 2012.

[71] Frank Wood, Jan Gasthaus, Cédric Archambeau, Lancelot James, and Yee Whye Teh. The sequence memoizer. *Communications of the ACM*, 54(2):91–98, 2011.

[72] Dell Zhang. Extracting key-substring-group features for text classification. In *In Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD 06*, pages 474–483. ACM Press, 2006.

[73] Gunter M. Ziegler. *Lectures on Polytopes*. Graduate texts in mathematics. Springer-Verlag, 1995.

[74] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *TIT*, 23(3):337–343, 1977.

[75] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding, 1978.