An Efficient Algorithm for Large Scale Compressive Feature Learning

Hristo S. Paskov
Department of Computer Science
Stanford University

John C. Mitchell
Department of Computer Science
Stanford University

Trevor J. Hastie
Department of Statistics
Stanford University

Abstract

This paper focuses on large–scale unsupervised feature selection from text. We expand upon the recently proposed Compressive Feature Learning (CFL) framework, a method that uses dictionary-based compression to select a K-gram representation for a document corpus. We show that CFL is NP–Complete and provide a novel and efficient approximation algorithm based on a homotopy that transforms a convex relaxation of CFL into the original problem. Our algorithm allows CFL to scale to corpuses comprised of millions of documents because each step is linear in the corpus length and highly parallelizable. We use it to extract features from the BeerAdvocate dataset, a corpus of over 1.5 million beer reviews spanning 10 years. CFL uses two orders of magnitude fewer features than the full trigram space. It beats a standard unigram model in a number of prediction tasks and achieves nearly twice the accuracy on an author identification task.

1 Introduction

This paper focuses on large–scale learning tasks involving text: problems involving millions of examples and features, gigabytes of data, and thousands of classes. Processing speed and memory restrictions dominate the kinds of models we can train, and otherwise innocuous limitations, such as 32–bit memory addressing, become real issues. Of the many popular algorithms that fail in this regime, Kernel methods form an important class whose quadratic dependence on the number of training examples renders them unusable. As a result, many methods rely on simple linear models that explicitly represent examples as vectors $x \in \mathbb{R}^d$ and make predictions based on an inner product $x^T w$. The main issue, then, is to select an informative yet compact and sparse feature representation for each data point.

Selecting a good feature representation is a critical to any learning problem, irrespective of size; the feature set provides the lens through which a learning algorithm accesses the data. In the context of text, this problem often boils down to selecting which K–grams to use when forming a bag of K–grams representation for a document corpus. Popular heuristics filter K–grams by document frequency, chi–squared value, or information gain [19, 8] and are fast, but treat features independently and fail to account for interactions. More recently, methods based on L_1 regularization [17, 20] perform variable selection while learning, but they also require more processing effort.

Complicating the issue of feature selection is that, while a document corpus may be available, the particular learning task it will be used for may be unknown. This occurs when obtaining labeling information is slow, when a researcher wishes to explore the data without a specific goal in mind, or simply when the dataset is to be analyzed in multiple ways. Unsupervised feature selection methods are particularly attractive in these scenarios because they give the researcher freedom to decide on the learning task after features have been extracted. Importantly, features can be extracted a single time in an offline manner and used for all subsequent tasks, oftentimes allowing for rapid experimentation.

Dictionary–based compression offers a potential solution to these problems. It is based on the idea that a good feature set is given by the K–grams stored while compressing a corpus because they reflect inherent structure in the text. As the compression algorithm seeks to remove redundancy, K–grams must "win out" to be included in the dictionary and features are selected jointly. A number of papers [6, 1, 3, 10] report success when using off–the–shelf compression algorithms as kernels that compute the similarity of two documents according to their size when concatenated together and compressed. Moreover, [16] show that many of these off–the–shelf algorithms are true kernels and give their implicit feature spaces. Deriving an explicit representation from these algorithms remains difficult, however, because we must compress all pairs of documents or the entire corpus jointly. As discussed in [14],

Appearing in Proceedings of the 17^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2014, Reykjavik, Iceland. JMLR: W&CP volume 33.

these algorithms produce poor features in the latter case because they are sensitive to the order in which documents are concatenated before compression.

We focus on the Compressive Feature Learning (CFL) [14] framework, a recent dictionary–based compression scheme that is explicitly designed to find a feature space, i.e. set of K–grams, for a document corpus. Unlike off–the–shelf compressors, CFL is invariant to document order and produces features by compressing the entire document corpus. As shown in [14], CFL can achieve state of the art performance on supervised learning problems even though it selects features without any knowledge of the labels. It can also help elucidate structure in unsupervised learning scenarios. Importantly, [14] provide an approximation algorithm that can scale to larger datasets because each refinement step is linear in the corpus length and highly parallelizable.

1.1 Contributions

(1) We show that CFL is NP–Complete and that its related operations of encoding/decoding a document (once the dictionary is fixed) can be performed in $O(Kn)$ time, where n is the document length and K is the maximum K–gram length. **(2)** We introduce a new, approximate algorithm for CFL that is significantly faster than our previous iterative reweighting (IR) algorithm [14] and that produces solutions of similar quality. We demonstrate our algorithm's performance on several datasets. **(3)** Our algorithm is based on a novel homotopic approximation scheme for CFL that solves a sequence of increasingly more difficult optimization problems. The problems transform from the linear relaxation of CFL into a non–convex problem whose local optima are always binary. **(4)** We adapt the linearized Alternating Directions Method of Multipliers (ADMM) framework [13] to solve our homotopy and show how to tune its parameters. Each step of our algorithm requires $\Theta(Kn)$ operations and is faster than the $\Theta(K^2n)$ required for each ADMM step in the IR algorithm. **(5)** Finally, we use our algorithm to compress the BeerAdvocate [12] dataset, a large corpus of approximately 1.5 million beer reviews spanning 10 years. We demonstrate the superiority of CFL's features over unigrams in predicting a beer's ABV and rating (along various categories) from its review. We also use these features to identify authors based on samples of their reviews and achieve nearly *twice* the accuracy of unigram and full trigram models.

2 Compressive Feature Learning

We now describe the Compressive Feature Learning (CFL) scheme introduced in [14] for extracting features from a document $D = x_1 x_2 \dots x_n$ comprised of n words. At its core CFL is a lossless dictionary–based compression algorithm and is related to the popular LZ77 algorithm [21]. It finds an efficient way to represent D via a subset of its substrings, known as a *dictionary*, and a set of *pointers* that indicate where copies of each dictionary string should be placed in order to reconstruct D. The dictionary and pointer set are chosen so as to minimize the cost of storing all pointers and dictionary strings (in plaintext). We will refer to these costs as the pointer set and dictionary costs, respectively. Figure 1 demonstrates this representation scheme and associated storage costs on a toy example for three different pointer costs.

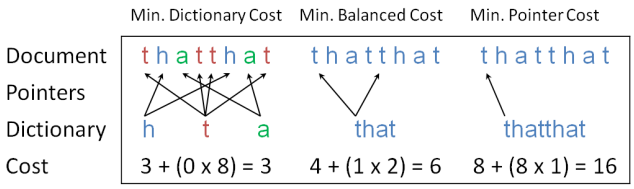

Figure 1: Three different CFL solutions on a toy example. Dictionary cost: number of characters in dictionary. Pointer set cost: $\lambda \times$ number of pointers. **Left:** dictionary cost only ($\lambda = 0$). **Right:** expensive pointer cost ($\lambda = 8$). **Center:** balanced dictionary and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

More formally, let the set of all unique K–grams [1] in $\mathcal{D}$ be $\mathcal{S} = \{x_i \dots x_{i+t-1} | 1 \le t \le K, 1 \le i \le n-t+1\}$ and let $\mathcal{P} = \left\{(s,l) \middle| s = x_l \dots x_{l+|s|-1}, |s| \le K\right\}$ be the set of all $m = |\mathcal{P}|$ potential pointers. Without loss of generality, we will think of $\mathcal{P}$ as an ordered set so that each unique pointer p_i has a unique index $i \in \{1,\dots,m\}$ associated with it. Define $J(s) \subset \{1,\dots,m\}$ to be the set of indices of all pointers that correspond to the same string s (at different locations in D).

Compressing D can be cast a binary linear minimization problem over a bit vector $w \in \{0,1\}^m$ that tells us which pointers to use in the compressed representation of D. Specifically, w *reconstructs* word x_j if for some $w_i = 1$ the corresponding pointer $p_i = (s,l)$ satisfies $l \le j < l + |s|$. In this case, p_i is used reconstruct part of D by pasting a copy of string s into location l. We can ensure that w reconstructs every word in D by requiring that $Xw \ge \mathbf{1}$, where $X \in \{0,1\}^{n \times m}$ indicates which words each $w_i = 1$ can reconstruct. The i-th column of X is zero everywhere except for a contiguous sequence of ones corresponding to the words which $w_i = 1$ reconstructs. We can also use w to implicitly define the dictionary (a subset of $\mathcal{S}$) by noting that s must be stored in the dictionary if $\|w_{J(s)}\|_\infty = 1$, i.e., some pointer using s is used in the compression of D. Assuming the pointer cost of setting $w_i = 1$ is given by $d_i \ge 0$ and the cost of storing any $s \in \mathcal{S}$ is $c(s)$, our lossless compression criterion is

$$\begin{aligned} \underset{w}{\text{minimize}} \quad & w^T d + \sum_{s \in \mathcal{S}} c(s) \|w_{J(s)}\|_\infty \\ \text{subject to} \quad & Xw \ge \mathbf{1}, \quad w \in \{0,1\}^m. \end{aligned} \tag{1}$$

[1] We use K–gram to mean word sequence *up to* length K.

We are particularly interested in the setting where the pointer cost is the same for all pointers, i.e., $d_i = \lambda$, 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 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.

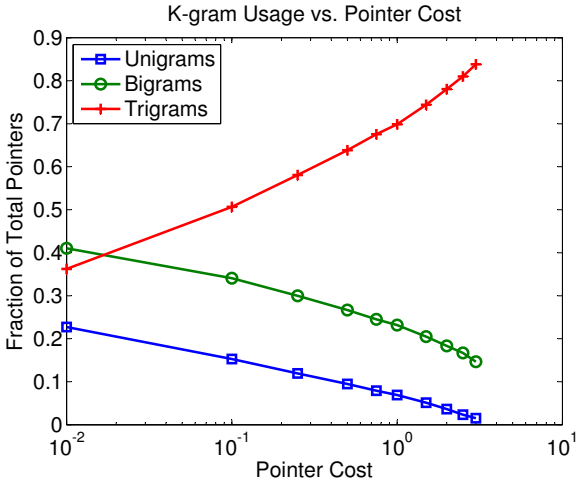

Figure 2: Fraction of pointers that correspond to unigrams, bigrams, and trigrams in the compressed representation of the BeerAdvocate dataset. We use the same pointer cost for all pointers and allow up to trigrams.

We can also compress multiple documents jointly. Given N documents $D_1, D_2, \ldots, D_N$, we form D by concatenating them into a long document $D = D_1 D_2 \ldots D_N$ so that $n = \sum_{i=1}^{N} n_i$ and disallowing any pointers that span document boundaries. Finally, we can extract features from a compressed document(s) by computing a bag of K–grams representation from the pointers used in its compressed representation, for further details see [14].

3 Complexity of Compression and Learning

This section discusses the computational complexity of CFL and the related operations of encoding and decoding a document. We give simple, linear–time algorithms for encoding and decoding a document once the dictionary is known and show that learning $\mathcal{S}$ is NP-Complete.

3.1 Decoding and Encoding

A document D is *encoded* in terms of dictionary $\mathcal{S}$ by finding the cheapest pointer set that reconstructs D. Here we assume that every occurrence of an $s \in \mathcal{S}$ in D is a potential pointer and that $\mathcal{S}$ can reconstruct D entirely. This problem is a special case of (1) obtained by setting $c(s) = 0$ (and disallowing all pointers for strings not in $\mathcal{S}$) and is applicable whenever $\mathcal{S}$ is decided upon ahead of time. It is important because it allows us to compress new documents without changing the dictionary and because (1) can be expressed as a nested problem in which the outer minimization selects a dictionary and the inner minimization encodes D. The inverse operation is *decoding* where, given a pointer encoding $U \subset \mathcal{P}$, we wish to reconstruct D. These operations can be performed independently for each document when D represents multiple documents, so we assume that D is a single document.

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.

Encoding Encoding can be performed in $O(Kn)$ time using dynamic programming. We assume that $\mathcal{S}$ contains all K–grams in D; we can disallow certain K–grams from being used by setting their pointers' costs to ∞. Define $R(i)$ to be the minimal cost of encoding D up to, but not past, position i so that $R(n)$ is the cost of the optimal encoding. The algorithm finds $R(n)$ by computing $R(i)$ from $i = 1$ up to n using the previous values of $R(j)$ for $1 \leq j < i$ to find $R(i)$ 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

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

As such, $R(i)$ can be found in $O(K)$ time given $R(0), \ldots, R(i-1)$ 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 $R(i)$ and working backwards once we compute $R(n)$. The overall running time of our procedure is therefore $O(Kn)$ and it is inherently online, requiring only a single pass through D.

3.2 Dictionary Learning

Given the simplicity of the encoding/decoding procedures, one would hope that an efficient procedure exists for solving CFL, i.e. learning $\mathcal{S}$ in addition to the document encoding. However, the addition of the dictionary cost term to

objective (1) significantly complicates the underlying problem. In fact, a simpler problem known as the *External Macro Data Compression* (EMDC) problem is discussed in [9] and shown to be NP-Complete via reduction from the Vertex Cover problem [9]. 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 (1) is itself a binary linear program, we conclude that CFL is NP-Complete.

4 Approximation Scheme

This section describes the homotopic scheme we employ to solve (1) approximately. Our scheme is an instance of an iterative binarization (IB) procedure that minimizes a sequence of interrelated problems until it arrives at a binary solution. In particular, IB 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, IB will arrive at a nearly optimal binary solution. High level pseudocode for IB 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.

4.1 Homotopy

In terms of the IB framework, homotopic methods rely on a homotopy parameter $\zeta \in [0,1]$ as their state variable and steadily increase ζ from 0 to 1. We use for $f_\zeta(x)$ 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 the convex relaxation of (1). 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 inital solution for each progressively harder problem. It traces out a solution path that starts at the continuous minimizer of (1) and ends at a high quality binary solution nearby.

We define our homotopy by first showing how to express (1) as a (non–convex) continuous problem whose minimizer must always be binary. Define $T(i) \subset \{1,\dots,m\}$ for $i = 1,\dots,n$ to be the index set which picks out 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 (1) is equivalent to

$$\begin{aligned} \underset{w}{\text{minimize}} \quad & w^T d + \sum_{s\in\mathcal{S}} c(s)\|w_{J(s)}\|_\infty \\ \text{subject to} \quad & \|w_{T(i)}\|_\infty = 1 \ \forall i = 1,\dots,n \\ & w \geq \mathbf{0}. \end{aligned} \tag{2}$$

The solution to (2) must be binary because the objective pushes w towards 0 and any $w_i < 1$ has no impact on the ∞–norm constraint.

We use the formulation in (2) to define our homotopy. Specifically, for $\zeta \in [0,1]$ we consider the problem

$$\begin{aligned} \underset{w}{\text{minimize}} \quad & w^T d + \sum_{s\in\mathcal{S}} c(s)\|w_{J(s)}\|_\infty \\ \text{subject to} \quad & \|w_{T(i)}\|_\infty \geq \zeta \ \forall i = 1,\dots,n \\ & Xw \geq \mathbf{1},\ w \geq \mathbf{0}. \end{aligned} \tag{3}$$

When $\zeta = 0$ the ∞–norm constraint cannot be active and the problem reduces to the linear relaxation of (1)[2]. However, when $\zeta = 1$ the ∞–norm constraint is more restrictive than the linear constraint and the problem reduces to (2). As a matter of interest, note that (3) 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 (1), the largest value of ζ at which (3) is still convex is given by $\zeta = \min_{i=1,\dots,n} \|w^*_{T(i)}\|_\infty$.

4.2 ADMM Formulation

We use linearized ADMM (LADMM) [13], a variant of the ADMM framework, to solve (3) approximately. While ADMM has traditionally been used for convex problems, it also converges (to a local optimum) for non-convex problems [2]. 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

$$\mathcal{L}_\rho(w,z,y) = h(w) + g(z) + y^T(Aw - z) + \frac{\rho}{2}\left\|Aw - z\right\|_2^2 \tag{4}$$

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

[2]Note that the linear relaxation requires $\mathbf{1} \geq w \geq \mathbf{0}$ but the upper bound is implied because no local optimum can have $w_i > 1$.

w while holding all other variables fixed, then z, and finally updates y according to [2]. 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 $A \neq I$. LADMM [13] remedies this by replacing the last term in (4) with its first order Taylor expansion around the previous value of w, $\bar{w}$, and a regularization term:

$$\frac{\rho}{2}\left\|Aw - z\right\|_2^2 \approx \rho(A\bar{w} - z)^T Aw + \frac{\mu}{2}\left\|w - \bar{w}\right\|_2^2. \quad (5)$$

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.

4.2.1 ADMM Split

In what follows, $I\{\bullet\}$ is the convex indicator function that is ∞ if the constraint inside the braces is not met and is 0 otherwise. We apply LADMM to (3) by splitting it into three distinct terms:

1. $h(w) = w^T d + \sum_{s\in\mathcal{S}} c(s)\|w_{J(s)}\|_\infty + I\{w \geq 0\}$
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), 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.

4.2.2 Minimization

Solving for w Simple algebra shows that the linearization of $\mathcal{L}_\rho$ is separable in each $s \in \mathcal{S}$. The supplementary material shows that each of these subproblems is of the form $\frac{\rho+\mu}{2}\|w_{J(s)} - q\|_2^2 + c(s)\|w_{J(s)}\|_\infty$ for appropriately defined q and can therefore be solved in $O(m_s)$ time, where m_s is the dimensionality of $w_{J(s)}$, using the procedure outlined in [14].

Solving for z We show in the supplementary material 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 3.1. 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

$$\left\|v - \theta\right\|_2^2 + g_2(\theta) \quad (6)$$

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 $\theta = \max(v, \zeta\psi)$.

4.2.3 Runtime Analysis

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 the supplementary material, the structure and sparsity of X allows us to perform this multiplication in $\Theta(m) = \Theta(Kn)$ operations, rather than $\Theta(nm)$ 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 $s \in \mathcal{S}$ and that the z and θ–updates, as well as multiplication by X or X^T, all parallelize across individual documents.

4.2.4 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 [13] 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 1. *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 show in the supplementary material 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 supplementary material also 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 [18] 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 [4] 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 [2] 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 (4), we measure convergence based on the quantities

$$r = \|Xw - z\|_\infty \quad u = \|z - \bar{z}\|_\infty \tag{7}$$

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.

4.3 Comparison with Iterative Reweighting

The IR algorithm in [14] uses an IB scheme where $f_\zeta(x)$ is a weighted linear relaxation of (1), with the state variable $\zeta \in \mathbb{R}_+^m$ 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 (1) 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.

5 Experiments

This section demonstrates the performance of our algorithm and CFL on the BeerAdvocate dataset [12], a document corpus consisting of $1,586,088$[3] 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.

5.1 Predictive Tasks

We compute a bag of trigrams [4] representation for each review by running our algorithm on the entire dataset with $K = 3$ and counting pointer frequencies as described in Section 2. We set each K–gram's dictionary storage cost to be its word length (between 1 and 3) and use a constant pointer cost of λ. Finally, we vary λ over a grid of 10 values ranging from 0.01 to 3; Figure 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 4.2.4. We increase ζ by increments of 0.1 whenever both convergence parameters in (7) 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 (7) 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

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

[4] Recall that this includes unigrams and bigrams.

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.

5.1.1 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 [7] and liblinear [5], 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 1 shows the testing accuracy of this approach when reviews are represented by their unigrams, (full) trigrams [5], 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 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 K–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 1: Testing Accuracy on Author Identification Task

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

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

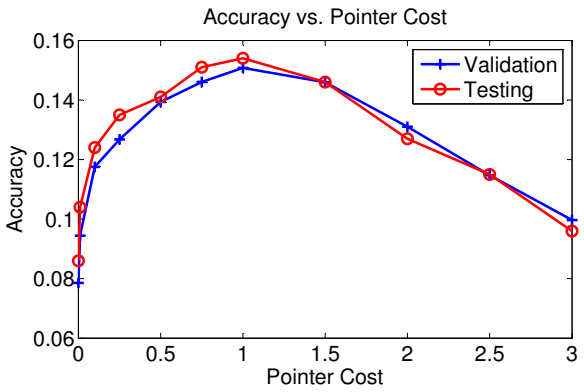

Figure 3: Testing and validation set accuracies of CFL features on author identification task as a function of the pointer cost. Unigrams correspond to $\lambda = 0$.

5.1.2 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 $100,000$, respectively, and the same splits are used for all tasks. We use glmnet to train an Elastic Net [7] on the training data and select all regularization parameters and the pointer cost for CFL through a grid search on the validation set. Table 2 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 2: 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**

5.2 Performance

We compare the performance of our algorithm to the IR scheme of [14] when used to compress subsets of the BeerAdvocate, 20 Newsgroups [15], and IMDb [11] datasets. All experiments are done on the same desktop computer[6]

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

with both algorithms coded in C. The code for the IR algorithm is the original used in [14].

Table 3 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.

Table 3: 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

Figure 4 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 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 4 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.

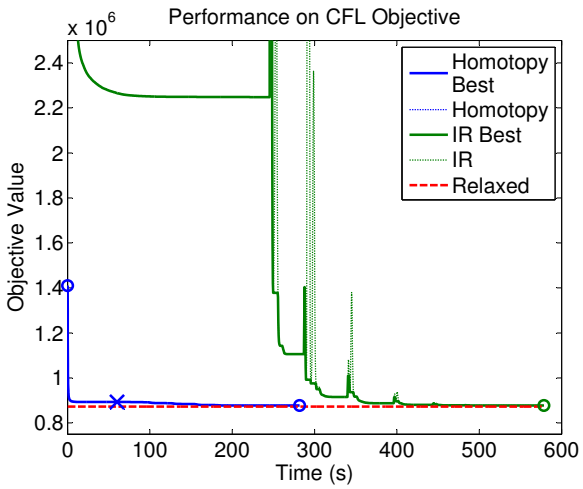

Figure 4: Performance trial comparing our homotopic method to the IR scheme of [14]. The red line denotes a lower bound for the objective. Circles indicate the starting/stopping points of the algorithms and the "X" marks the point at which we increase ζ from 0.

6 Conclusion

Feature selection is an essential problem in large and small–scale learning problems alike. The Compressive Feature Learning (CFL) framework offers a promising method for selecting a K–gram representation from a text corpus using compression. We provide a fast algorithm that allows this framework to handle large datasets comprised of millions of documents. While we prove that CFL is NP–Complete, we show that its related operations of decoding and encoding a document once the dictionary is fixed can be done in linear time. Our fast algorithm is based on a new homotopic scheme that finds an approximate solution for CFL by solving a sequence of interrelated problems. This homotopy produces a path of solutions that starts at the continuous minimizer of CFL and ends at a high quality binary solution nearby. The linear time encoding procedure plays an important role in this approximation scheme as it drives the homotopy and provides an effective way to round continuous solutions. We use the linearized ADMM framework to solve our homotopy efficiently and require $\Theta(Kn)$ operations per ADMM step. Finally, we use our algorithm to extract features from the BeerAdvocate dataset, a large corpus of text comprised of 1.5 million documents. The features CFL finds allow us to experiment with a variety of learning tasks and achieve nearly twice the accuracy of unigrams on a large author identification problem. This dataset also showcases the speed of our algorithm as it provides a significant performance improvement over the iterative reweighting scheme of [14] and finds solutions of similar quality.

References

[1] D. Benedetto, E. Caglioti, and V. Loreto. Language trees and zipping. *PRL*, 88(4):048702, 2002.

[2] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

[3] A. Bratko, B. Filipič, G. V. Cormack, T. R. Lynam, and B. Zupan. Spam filtering using statistical data compression models. *JMLR*, 7:2673–2698, 2006.

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

[5] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

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

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

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

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

[10] E. Keogh, S. Lonardi, and C. A. Ratanamahatana. Towards parameter-free data mining. In *KDD*, 2004.

[11] A. Maas, R. Daly, P. Pham, D. Huang, A. Y. Ng, and C. Potts. Learning word vectors for sentiment analysis. In *ACL*, 2011.

[12] J. J. McAuley and J. Leskovec. From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews. In *World Wide Web*, 2013.

[13] N. Parikh and S. Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 2013.

[14] H. Paskov, R. West, J. Mitchell, and T. Hastie. Compressive feature learning. In *NIPS*, 2013.

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

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

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

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

[19] Y. Yang and J. Pedersen. A comparative study on feature selection in text categorization. In *ICML*, 1997.

[20] J. Zhu, S. Rosset, T. Hastie, and R. Tibshirani. 1-norm support vector machines. In *NIPS*, 2004.

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