An Application of the Submodular Principal Partition to Training Data Subset Selection

Hui Lin
Department of Electrical Engineering
University of Washington
Seattle, WA 98125
`hlin@ee.washington.edu`

Jeff Bilmes
Department of Electrical Engineering
University of Washington
Seattle, WA 98125
`bilmes@ee.washington.edu`

Abstract

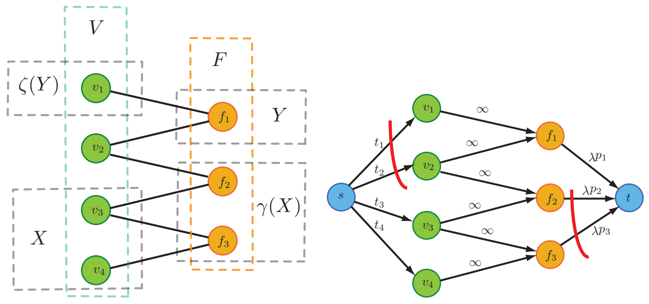
We address the problem of finding a subset of a large training data set (corpus) that is useful for accurately and rapidly prototyping novel and computationally expensive machine learning architectures. To solve this problem, we express it as an minimization problem over a weighted sum of modular functions and submodular functions. Quantities such as number of classes (or quality) in a set of samples, or quality of a bundle of classes are submodular functions which make finding the optimal solutions possible. We apply the principal partition to our problem such that solutions for all possible trade-offs between a modular function and a submodular function can be found efficiently. We show results for speech recognition on the Switchboard-I speech recognition corpus, demonstrating improved results over previous techniques for this purpose. We also demonstrate the variety of the resulting corpora that may be produced using our method.

1 Introduction

A challenge working with large machine learning data sets is determining how to quickly test a new algorithm on such data. In general, it is important to be able to test a novel idea quickly, without investing enormous amounts of time on the engineering effort to make the ideas perform well, and if a new idea ends up performing poorly, knowing this sooner rather than later will avoid futile work. Often, the more novel the idea, the more effort it takes to get it working on large data since there is less chance of potential implementation reuse from pre-existing systems.

One way to address this problem is to produce a smaller version of the data. Often, the complexity of an iteration of a machine learning system is linear in the number of samples but *polynomial* in the number of classes. For instance, in speech recognition, decoding using a trigram language model with size-N vocabulary (number of *classes* or *types*) has, in the worst case, a complexity of at least $O(N^3)$; in image segmentation, breaking an image into regions where each region corresponds to one of N objects (classes or types) using a MRF with non-submodular k-interaction potential functions, can have complexity as bad as $O(N^k)$.

Our goal therefore is to improve the turnaround time for developing and testing novel algorithms but still exploit the utility in large data. One approach is to select a subset of the data where the number of types is limited and the total number of samples is maximized. Ideally, we would like a process that can take a large corpus and produce a subset that satisfies a particular purpose. For example, when vocabulary size is the key attribute hindering the rapid evaluation of novel method, we might choose a limited vocabulary subset of data of maximal size. On the other hand, we may wish to correct for some other quality, such as imposing a bias against certain word forms. We moreover wish the results from the corpus to be an accurate reflection of results from the entire corpus.



(a) Bipartite graph (b) $s - t$ graph

Figure 1: In subfigure (a), $V = \{v_1, v_2, v_3, v_4\}$ and $F = \{f_1, f_2, f_3\}$. For $X = \{v_3, v_4\}$, $\gamma(X) = \{f_2, f_3\}$; for $Y = \{f_1\}$, $\zeta(Y) = \{v_1\}$. In (b), the s-t graph corresponding to Eq. 5. Also see [4].

We address this issue as a combinatorial optimization problem, and in this paper we formulate it as an optimization problem via the use of the principle partition of a submodular functions [1]. As we will see, this allows us to express the problem of corpus subset selection in a variety of flexible ways, suiting the needs of an individual novel system and its designer.

We demonstrate our approach for the task of large vocabulary spontaneous conversational speech recognition – one of the most challenging tasks in speech processing and one of the most computationally demanding in all machine learning. In recent times, very large amounts of transcribed speech data, with both many tokens and many types, have become available. Some corpora have vocabulary size as large as one million and as many as 230 billion tokens [2]. In the past, researchers have addressed this issue and have used methodology to create a small version of a large corpus [3]. As we will see, the approach herein outperforms these previous methods.

2 Problem Setup

We formulate our approach using a bipartite graph. Let V be a set of corpus utterances, and let F be the vocabulary (set of types) contained in these utterances. We define a bipartite graph $G = (V, F, E)$ where V are the left nodes, F are the right nodes, and $E \subseteq V \times F$ are the edges where each $(v, f) = e \in E$ is an edge between a node $v \in V$ and $f \in F$ if utterance v contains word f.

In [3], a simple greedy algorithm was used to find a vocabulary of any given size while selecting the corresponding in-vocabulary utterances from the larger corpus. The greedy heuristic was designed to maximize the amount of the data (number of tokens) selected. In particular, in each greedy step, a new type is added such that the number of tokens in the utterances that contains only the new vocabulary is maximized. Formally, the greedy algorithm in [3] attempts to select a set of types $Y \subseteq F$ such that a set function $f_{\text{svb}} : 2^F \to \mathbb{R}$ is maximized, where $f_{\text{svb}}(Y) \triangleq \sum_{i \in \zeta(Y)} t_i$, t_i is the number of tokens in utterance i, and $\zeta(Y)$ is the set of nodes $v \in V$ that have neighbors *only* in Y and not in $F \setminus Y$. That is

$$\zeta(Y) \triangleq \{v \in V : \exists f \in Y \text{ s.t. } (v, f) \in E \text{ and } (v, f') \notin E, \forall f' \in F \setminus Y\}$$

The greedy algorithm in [3] works by repeatedly taking an existing set Y, finding a $y \in F \setminus Y$ that maximizes $f_{\text{svb}}(Y \cup \{y\})$ and then updating Y by including that y. This repeats as long as a desired vocabulary size constraint $|Y| \le b$ is satisfied.

A set function $f : 2^V \to \mathbb{R}$ is *submodular* [1] if for any $A \subseteq B \subseteq V$ and $k \notin B$, we have

$$f(A \cup \{k\}) - f(A) \ge f(B \cup \{k\}) - f(B), \tag{1}$$

and f is *supermodular* if $-f$ is submodular. Now unfortunately for [3], f_{svb} is supermodular [4]. Therefore, the problem is essentially one of maximizing a supermodular function subject to a cardinality constraint. The greedy algorithm for this problem, as used in [3], has an unboundedly poor approximation factor. For instance, let $F = \{x, y, z\}, f(\{x\}) = 1, f(\{y\}) = f(\{z\}) = 0, f(\{x, y\}) = f(\{x, z\}) = 1$, $f(\{y, z\}) = p > 1$ and $b = 2$. Greedily maximizing f leads to a solution $\{x, y\}$ with objective function value 1, while the true optimal objective function value is p.

Since p is arbitrary, the approximation factor for this example is unbounded. This is not surprising, as it is known that greedy algorithm works near-optimally only when maximizing a *submodular* function subject to cardinality (knapsack) constraint [5, 6], and has been successfully applied to active learning [7] and document summarization [8, 9] in our previous work.

In this paper, we treat the corpus creation problem as finding a subset of sentences $X \subseteq V$ that simultaneously minimizes the vocabulary size and maximizes the total amount of data, measured as either the number of utterances, the number of tokens in the utterances, or duration of speech. We do this by finding X that maximizes the following expression

$$w(X) - \lambda\Gamma(X) \tag{2}$$

where $w(X)$ measures the amount of data contained in utterances X, $\Gamma(X)$ represents the vocabulary size associated with utterances X, and $\lambda \geq 0$ is a tradeoff coefficient.

We may have different w functions depending on our needs. For instance, when w is the cardinality function, i.e. $w(X) = |X|$, it measures the number of the utterances. We can also use $w(X) = \sum_{i \in X} t_i$: when t_i represents the number of tokens in utterance i, $w(X)$ is the total number of tokens in X; and when t_i represents the speech time-length of utterance i, $w(X)$ is the total speech duration of the utterances X.

We can also have different forms of $\Gamma(X)$. First, we define $\gamma(X)$ to be the set of nodes $f \in F$ that have a neighbor in X (i.e., words that appear in utterances X). That is

$$\gamma(X) \triangleq \{f \in F : \exists v \in X \text{ s.t. } (v, f) \in E\}. \tag{3}$$

Then several possible forms of $\Gamma(X)$ include:

- $\Gamma_1(X) = |\gamma(X)|$. This represents the collective vocabulary size of utterances in set X.
- $\Gamma_2(X) = m(\gamma(X)) = \sum_{i \in \gamma(X)} p_i$, where p_i indicates the unimportance of word i. A larger p_i states that word i is less important. This allows certain desirable properties of the vocabulary of the resultant corpus to be expressed (e.g., words with more syllables might be preferred). Note if $p_i = 1, \forall i$, then $\Gamma_2 = \Gamma_1$.
- In some case, incorporation of stop words (e.g., "about", "after", "all", "am", etc.) is less desirable compared to content words that convey the semantic meaning of an utterance. Let (F_s, F_f) be a partition of F into stopwords F_s and F_f non-stop (function) words $F_f = F \setminus F_s$. We can use the above modular function (i.e. m in Γ_2) to put a preference against stop words. I.e., for any $f \in F_s$ and $f' \in F_f$, we would have that $m(f) \geq m(f')$. So far (empirically) this has ended up performing poorly since it effectively reduces the vocabulary size down only to the function words.

  Alternatively, we can use a submodular function that has a preference away from stop words — the function is a mixture of a modular function over stop words and concave-submodular function over content words. I.e., we define

  $$\Gamma_3(X) = m(\gamma(X) \cap F_s) + \alpha\sqrt{m(\gamma(X) \cap F_f)} \tag{4}$$

  where α is a tradeoff coefficient. This function is such that once we start including non-stop words, they become cheaper but the stop words retain their expense.

Note that maximizing Eq. (2) is identical to finding X that minimizes

$$L(\lambda, X) \triangleq w(V \setminus X) + \lambda\Gamma(X). \tag{5}$$

For a given λ, if $L(\lambda, X)$ is a submodular function on X, then $\min_{X \subseteq V} L(\lambda, X)$ can solved exactly in polynomial time [10, 11]. Fortunately, all the aforementioned w functions are modular (both submodular and supermodular), and all the aforementioned Γ functions are submodular, making $L(\lambda, X)$ submodular in all our cases. Therefore we can solve our corpus creation problem optimally by leveraging submodular function minimization techniques.

To create a corpus with the desired property (e.g., a fixed upper limit on vocabulary size), different values of trade-off coefficient λ must be tried, where multiple calls of the optimization algorithm are required. That is, we do not have direct control on the vocabulary constraint, only indirect control via λ. In our case, however, *all* possible solutions for $\min_{X \subseteq V} L(\lambda, X)$ for *all* possible $\lambda \geq 0$ can be found in the same complexity as the complexity of solving $\min_{X \subseteq V} L(\lambda, X)$ for a *single* λ, and moreover there are only a finite (no more than $|V|$) number of distinct values of λ that makes a difference, all thanks to submodularity.

3 Principal Partition

Let $\mathcal{X}(\lambda)$ be the minimizer of $L(\lambda, X)$ and $\mathcal{L}(\lambda)$ be the corresponding optimal value. That is

$$\mathcal{X}(\lambda) \triangleq \{X \subseteq V : L(\lambda, X) \leq L(\lambda, X'), \forall X' \subseteq V\}, \tag{6}$$

$$\mathcal{L}(\lambda) \triangleq L(\lambda, X), X \in \mathcal{X}(\lambda) \tag{7}$$

and let $X^+(\lambda)$ be the unique maximal element in $\mathcal{X}(\lambda)$ and $X^-(\lambda)$ be the unique minimal element in $\mathcal{X}(\lambda)$. Formally,

$$X^+(\lambda) \triangleq \bigcup \{X \subseteq V : L(\lambda, X) \leq L(\lambda, X'), \forall X' \subseteq V\}$$

$$X^-(\lambda) \triangleq \bigcap \{X \subseteq V : L(\lambda, X) \leq L(\lambda, X'), \forall X' \subseteq V\}.$$

Then, we have the following theorem

Theorem 1 (Theorem 7.15 in [1])**.** *If Γ is non-decreasing submodular. Then there exists a sequence of real values $\lambda_1 < \lambda_2 < \cdots < \lambda_p$ (the critical points) where $p \leq |V|$ such that the distinct $\mathcal{X}(\lambda), \lambda \in \mathbb{R}_+$ are given by: $\mathcal{X}(\lambda_i), i = 1, 2, \cdots, p$; For any $\lambda_i < \lambda < \lambda_{i+1}$ we have $\mathcal{X}(\lambda) = \{X^-(\lambda_i)\} = \{X^+(\lambda_{i+1})\} \triangleq \{X_i\}$.*

$\mathcal{L}(\lambda)$ is a piece-wise linear function. Every intersection point of the line segments is called *critical points*, i.e. $\lambda_1, \cdots \lambda_p$. For every critical point λ_i, $\mathcal{X}(\lambda_i)$ has at least two elements, with maximal element X_{i-1} and minimal element X_i ($X_{i-1} \supset X_i$). For any $\lambda_i < \lambda < \lambda_{i+1}$, $\mathcal{X}(\lambda)$ has only one element, i.e. $\mathcal{L}(\lambda) = \{X_i\}$. Therefore, all possible solutions can be identified by finding $\lambda_i, i = 1, \cdots p$ which corresponding to the following nested sets:

$$V = X_1 \supset X_2 \supset \cdots \supset X_p = \emptyset \tag{8}$$

where $X_1 \setminus X_2, \cdots, X_{p-1} \setminus X_p$ consists a decomposition of the ground set, and is called *principal partition* of Eqn. (5). Therefore, if we can find all the critical points and the corresponding $X_i, i = 1, \cdots p$, then we obtain a *natural* partition of the original corpus, we can take any proper X_i as the newly created corpus with desired property.

A straightforward way of finding all the distinct λ values is by using a divide-and-conquer strategy that recursively examines a given interval of λ, which requires $O(|V|)$ calls of minimization of a submodular function. On the other hand, finding all distinct λ values (and minimizing sets) can be done using parametric submodular function minimization. When using a push-relabel framework [12], finding solutions for all λ requires only the same asymptotic running time as a *single* submodular function minimization. Note that the push-relabel framework was firstly introduced in [13] for network flow (graph cut) problem. If the submodular functions used in are graph-representable (e.g. Γ_1, Γ_2 in our case), we can convert our submodular minimization problem to the problem of finding minimum s-t cuts on a graph (part (b) of Figure 2 shows one example of the conversion), and therefore a fast parametric flow algorithm [13] can be used to find all the solutions for the corpus creation problem for all possible values trade-off coefficient λ, while only requiring the computational complexity of running a single minimum s-t cut, which can be solved very efficiently even on very large graphs. For submodular functions that are not graph-representable (e.g. Γ_3), we use minimum-norm-point algorithm for minimizing our submodular functions, which is shown to be more efficient than the combinatorial algorithms in practice [14].

4 Experiments

We tested our corpus creation approach on Switchboard I. To be comparable with [3], we followed the same experimental setup. In particular, each side of the long conversations in the Switchboard-I

Table 1: Corpus statistics on vocabulary size, average number of phones in the pronunciations of the corpus vocabulary, the total number of utterances, the total number of tokens in the utterances, and the duration of speech. $w_1(X)$ measures the total number of tokens in X. $w_2(X)$ measures the total duration in X. In corpus D,$\Gamma_2(X) = \sum_{i\in\gamma(X)} p_i = \sum_{i\in\gamma(X)} \frac{C}{q_i}$, where C is a constant, and q_i is the number of phonemes in the pronunciation of word i. In corpus E, $\Gamma_2(X) = \sum_{i\in\gamma(X)\cap F_s} 100 + \sum_{i\in\gamma(X)\cap F_f} 1$.

ID	Method	Vocab size	Ave. proun.	Utt #	Token #	Minutes
SVB	greedy	10	2.5	6775	7792	55.8
		25	2.72	9778	13324	85.2
		50	2.78	12442	20914	115.8
		100	3.12	14602	28611	148.8
		500	3.93	23670	89420	386.4
A	$\|V \setminus X\| + \lambda\Gamma_1(X)$	10	2.80	7615	8771	60.84
		25	2.72	10911	15952	92.55
		51	2.75	13506	23076	122.49
		115	3.19	16573	34213	169.32
		489	3.88	26165	96260	406.42
B	$w_1(V \setminus X) + \lambda\Gamma_1(X)$	9	3.00	7197	8600	59.18
		24	2.71	10625	15803	90.33
		50	2.70	13156	**23124**	120.69
		113	3.11	16230	35174	169.05
		459	3.86	25175	93785	393.52
C	$w_2(V \setminus X) + \lambda\Gamma_1(X)$	10	2.90	7586	9114	62.75
		26	2.96	11030	16330	95.68
		50	3.06	13365	22916	**122.26**
		95	3.28	15580	31415	157.40
		437	3.85	24851	89987	380.87
D	$w_2(V \setminus X) + \lambda\Gamma_2(X)$	10	3.50	7329	8778	60.95
		27	3.74	10755	15473	93.65
		50	**3.78**	12998	21576	118.55
		746	4.95	29029	128902	537.42
E	$w_1(V \setminus X) + \lambda\Gamma_2(X)$	50	3.96	6705	8435	59.10

corpus was divided into shorter segments. These segments were further divided into smaller utterances at every silence longer than 500ms. The resulting utterances were pruned by removing those contain disfluency and filler-model words.

We investigated several types of w and Γ function, and produced corpora with different properties. The statistics of these corpora (corpus A,B,C,D), along with the statistics of SVitchboard [3], our baseline, are illustrated in Table 1. For the baseline, the greedy algorithm did not perform as poorly as one might expect, presumably because the increment of the supermodular function corresponding to this task is not diminishing that fast as the set size decreases. Note that in our approach the vocabulary sizes of the resulting corpora are naturally determined by the principal partition rather than predefined, and we choose those that are as close as possible to the SVitchboard vocabulary sizes (10, 25, 50, 100, and 500) for comparison.

Corpus A was produced with objective function $|V \setminus X| + \lambda\Gamma_1(X)$: maximizing the number of utterances while restricting the vocabulary size. Results in Table 1 show that corpus A indeed includes more utterances compared to other corpora (created with objectives that are not maximizing the number of utterances) given the same vocabulary size. For instance, with vocabulary size 10, corpus A contains 7615 utterances while SVitchboard has 6775. It even contains more utterances (26165 vs. 23670) with a smaller vocabulary (489 vs. 500), compared to SVitchboard.

Corpora B and C are similar to A except that utterance weights are used. In particular, for corpus B, each utterance is measured by the number of tokens it contains (i.e. $w_1(X)$ measures the number of tokens in utterances X), while in corpus C, each utterance is measured by the minutes of the non-silence speech (i.e. $w_2(X)$ measures the speech duration in X). As we can see from Table 1, corpora B and C do have the desired property. For example, with vocabulary size 50, there are 23124 tokens in B, which is larger than those in SVitchboard, or in corpora C and D; there are about 122 minutes of speech in C, which is more than those in SVitchboard, or in corpora B and D.

In some situations, we may be concerned not only about the amount of data and the vocabulary size, but also wish for the resulting vocabulary to possess certain properties. Our method can handle such scenarios naturally and efficiently. For instance, a corpus with a limited vocabulary but rich phonetic variety could be useful for research on novel pronunciation modeling, and this is how we produced corpus D. We approximated the phonetic richness by the number of phoneme tokens in the pronunciation of a word. We used a Γ_2 function, where p_i was set to be the inverse of the number of phonemes in the pronunciation of word i. Words with more phones will have lower a weight and therefore a higher chance of being selected. Table 1 shows that corpus D is well balanced in terms of corpus size and vocabulary variety. In particular, compared to SVitchboard at vocabulary size 50, corpus D not only has a vocabulary with longer average pronunciation length (with words like "absolutely" and "definitely"), but also includes more tokens and more acoustic speech. We compare the complete vocabulary of SVitchboard-50 and D-50 using a Venn diagram in Figure 2, clearly showing that D-50 has a richer lexicon.

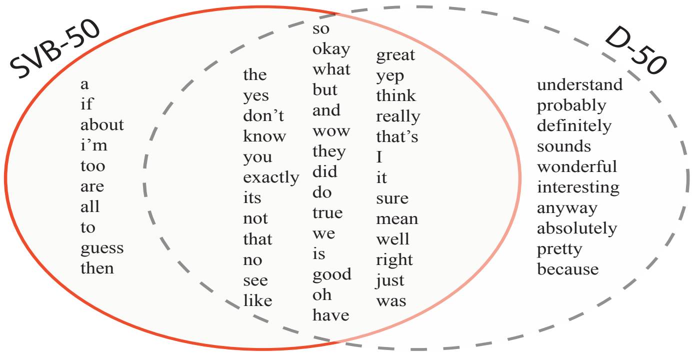


Figure 2: Venn diagram showing the vocabulary difference between SVitchboard-50 and D-50.

In corpus E, we used a $\Gamma_2(X) = \sum_{i \in \gamma(X) \cap F_s} C + \sum_{i \in \gamma(X) \cap F_f} 1$ such that non-stopwords (i.e., F_f) are preferred. The resulting corpus indeed includes more non-stopwords (note that function words usually contain more syllabels, and as we can see, the average pronunciation length of E vocabulary 50 outperforms all other corpora). Since most of the utterances in SWitchboard contain stopwords (i.e. it is a corpus on conversational speech), the size of a sub-corpus that contains less stopwords also significantly reduced. Nevertheless, we could adjust the constant C in Γ_2 to reach a preferred balance between corpus size and amount of stopwords.

References

[1] S. Fujishige, *Submodular functions and optimization*, Elsevier Science Ltd, 2005.

[2] C. Chelba, T. Brants, W. Neveitt, and P. Xu, "Study on interaction between entropy pruning and Kneser-Ney smoothing," in *Proc. of Interspeech*, September 2010.

[3] S. King, C. Bartels, and J. Bilmes, "SVitchboard 1: Small Vocabulary Tasks from Switchboard," in *Ninth European Conference on Speech Communication and Technology*. ISCA, 2005.

[4] H. Narayanan, *Submodular Functions and Electrical Networks*, North-Holland, Amsterdam, 1997.

[5] G.L. Nemhauser, L.A. Wolsey, and M.L. Fisher, "An analysis of approximations for maximizing submodular set functions I," *Mathematical Programming*, vol. 14, no. 1, pp. 265–294, 1978.

[6] S. Khuller, A. Moss, and J. Naor, "The budgeted maximum coverage problem," *Information Processing Letters*, vol. 70, no. 1, pp. 39–45, 1999.

[7] H. Lin and J. Bilmes, "How to select a good training-data subset for transcription: Submodular active selection for sequences," in *Proc. of Interspeech*, Brighton, UK, September 2009.

[8] H. Lin, J. Bilmes, and S. Xie, "Graph-based submodular selection for extractive summarization," in *Proc. IEEE Automatic Speech Recognition and Understanding (ASRU)*, Merano, Italy, December 2009.

[9] H. Lin and J. Bilmes, "Multi-document summarization via budgeted maximization of submodular functions," in *North American chapter of the Association for Computational Linguistics/Human Language Technology Conference (NAACL/HLT-2010)*, Los Angeles, CA, June 2010.

[10] A. Schrijver, "A combinatorial algorithm minimizing submodular functions in strongly polynomial time," *Journal of Combinatorial Theory, Series B*, vol. 80, no. 2, pp. 346–355, 2000.

[11] S. Iwata, L. Fleischer, and S. Fujishige, "A combinatorial strongly polynomial algorithm for minimizing submodular functions," *Journal of the ACM*, vol. 48, no. 4, pp. 761–777, 2001.

[12] L. Fleischer and S. Iwata, "A push-relabel framework for submodular function minimization and applications to parametric optimization," in *Symposium on Theory of Computing*, 2000, vol. 107, p. 116.

[13] G. Gallo, M. D. Grigoriadis, and R. E. Tarjan, "A fast parametric maximum flow algorithm and applications," *SIAM J. Comput.*, vol. 18, no. 1, pp. 30–55, 1989.

[14] S. Fujishige, T. Hayashi, and S. Isotani, "The minimum-norm-point algorithm applied to submodular function minimization and linear programming," *Research Institute for Mathematical Sciences Preprint RIMS-1571, Kyoto University, Kyoto Japan*, 2006.