Parallel Training of CRFs: A Practical Approach to Build Large-Scale Prediction Models for Sequence Data

H.X. Phan[1], M.L. Nguyen[1], S. Horiguchi[2], Y. Inoguchi[1], and B.T. Ho[1]

[1] Japan Advanced Institute of Science and Technology
1–1, Asahidai, Tatsunokuchi, Ishikawa, 923–1211, Japan
{hieuxuan, nguyenml, inoguchi, bao}@jaist.ac.jp
[2] Tohoku University
Aoba 6–3–09 Sendai, 980–8579, Japan
susumu@ecei.tohoku.ac.jp

Abstract. Conditional random fields (CRFs) have been successfully applied to various applications of predicting and labeling structured data, such as natural language tagging & parsing, image segmentation & object recognition, and protein secondary structure prediction. The key advantages of CRFs are the ability to encode a variety of overlapping, non-independent features from empirical data as well as the capability of reaching the global normalization and optimization. However, estimating parameters for CRFs is very time-consuming due to an intensive forward-backward computation needed to estimate the likelihood function and its gradient during training. This paper presents a high-performance training of CRFs on massively parallel processing systems that allows us to handle huge datasets with hundreds of thousand data sequences and millions of features. We performed the experiments on an important natural language processing task (phrase chunking) on large-scale corpora and achieved significant results in terms of both the reduction of computational time and the improvement of prediction accuracy.

1 Introduction

CRF, a conditionally trained Markov random field model, together with its variants have been successfully applied to various applications of predicting and labeling structured data, such as information extraction [1, 2], natural language tagging & parsing [3, 4], pattern recognition & computer vision [5, 7, 6, 8], and protein secondary structure prediction [9, 10]. The key advantages of CRFs are the ability to encode a variety of overlapping, non-independent features from empirical data as well as the capability of reaching the global normalization and optimization.

However, training CRFs, i.e., estimating parameters for CRF models, is very expensive due to a heavy forward-backward computation needed to estimate the likelihood function and its gradient during the training process. The computational time of CRFs is even larger when they are trained on large-scale datasets or using higher-order Markov dependencies among states. Thus, most previous work either evaluated CRFs on moderate datasets or used the first-order Markov

CRFs (i.e., the simplest configuration in which the current state only depends on one previous state). Obviously, this difficulty prevents us to explore the limit of the prediction power of high-order Markov CRFs as well as to deal with large-scale structured prediction problems.

In this paper, we present a high-performance training of CRFs on massively parallel processing systems that allows to handle huge datasets with hundreds of thousand data sequences and millions of features. Our major motivation behind this work is threefold:

• Today, (semi-)structured data (e.g., text, image, video, protein sequences) can be easily gathered from different sources, such as online documents, sensors, cameras, and biological experiments & medical tests. Thus, the need for analyzing, e.g., segmentation and prediction, those kinds of data is increasing rapidly. Building high-performance prediction models on distributed processing systems is an appropriate strategy to deal with such huge real-world datasets.

• CRF has been known as a powerful probabilistic graphical model, and already applied successfully to many learning tasks. However, there is no thoroughly empirical study on this model on large datasets to confirm its actual limit of learning capability. Our work also aims at exploring this limit in the viewpoint of empirical evaluation.

• Also, we expect to examine the extent to which CRFs with the global normalization and optimization could do better than other classifiers when performing structured prediction on large-scale datasets. And from that we want to determine whether or not the prediction accuracy of CRFs should compensate its large computational cost.

The rest of the paper is organized as follows. Section 2 gives the background of CRFs. Section 3 presents the parallel training of CRFs. Section 4 presents the empirical evaluation. And some conclusions are given in Section 5.

2 Conditional Random Fields

The task of predicting a label sequence to an observation sequence arises in many fields, including bioinformatics, computational linguistics, and speech recognition. For example, consider the natural language processing task of predicting the part-of-speech (POS) tag sequence for an input text sentence as follows: "*Rolls-Royce*_NNP *Motor*_NNP *Cars*_NNPS *Inc.*_NNP *said*_VBD *it*_PRP *expects*_VBZ *its*_PRP$ *U.S.*_NNP *sales*_NNS *to*_TO *remain*_VB *steady*_JJ *at*_IN *about*_IN *1,200*_CD *cars*_NNS *in*_IN *1990*_CD ._"
Here, "*Rolls-Royce Motor Cars Inc. said* ..." and "NNP NNP NNPS NNP VBD ..." can be seen as the input data observation sequence and the output label sequence, respectively. The problem of labeling sequence data is to predict the most likely label sequence of an input data observation sequence. CRFs [11] was deliberately designed to deal with such kind of problem.

Let $\mathbf{o} = (o_1, \ldots, o_T)$ be some input data observation sequence. Let $\mathcal{S}$ be a finite set of states, each associated with a label l ($\in \mathcal{L} = \{l_1, \ldots, l_Q\}$). Let $\mathbf{s} = (s_1, \ldots, s_T)$ be some state sequence. CRFs are defined as the conditional probability of a state sequence given an observation sequence as,

$$p_\theta(\mathbf{s}|\mathbf{o}) = \frac{1}{Z(\mathbf{o})}\exp\left(\sum_{t=1}^{T} \mathbf{F}(\mathbf{s}, \mathbf{o}, t)\right), \tag{1}$$

where $Z(\mathbf{o}) = \sum_{\mathbf{S'}} \exp\left(\sum_{t=1}^{T} \mathbf{F}(\mathbf{s'}, \mathbf{o}, t)\right)$ is a normalization factor summing over all label sequences. $\mathbf{F}(\mathbf{s}, \mathbf{o}, t)$ is the sum of CRF features at time position t,

$$\mathbf{F}(\mathbf{s}, \mathbf{o}, t) = \sum_i \lambda_i f_i(s_{t-1}, s_t) + \sum_j \lambda_j g_j(\mathbf{o}, s_t) \tag{2}$$

where f_i and g_j are *edge* and *state* features, respectively; λ_i and λ_j are the feature weights associated with f_i and f_j. Edge and state features are defined as binary functions as follows,

$$f_i(s_{t-1}, s_t) \equiv [s_{t-1} = l^{'}][s_t = l]$$
$$g_j(\mathbf{o}, s_t) \equiv [x_j(\mathbf{o}, t)][s_t = l]$$

where $[s_t = l]$ equals 1 if the label associated with state s_t is l, and 0 otherwise (the same for $[s_{t-1} = l^{'}]$). $x_i(\mathbf{o}, t)$ is a logical context predicate that indicates whether the observation sequence $\mathbf{o}$ (at time t) holds a particular property. $[x_i(\mathbf{o}, t)]$ is equal to 1 if $x_i(\mathbf{o}, t)$ is *true*, and 0 otherwise. Intuitively, an edge feature encodes a sequential dependency or causal relationship between two consecutive states, e.g., "the label of the previous word is JJ (adjective) and the label of the current word is NN (noun)". And, a state feature indicates how a particular property of the data observation influences the prediction of the label, e.g., "the current word ends with *-tion* and its label is NN (noun)".

2.1 Inference in Conditional Random Fields

Inference in CRFs is to find the most likely state sequence $\mathbf{s}^*$ given the input observation sequence $\mathbf{o}$,

$$\mathbf{s}^* = \text{argmax}_{\mathbf{S}}\ p_\theta(\mathbf{s}|\mathbf{o}) = \text{argmax}_{\mathbf{S}} \left\{\exp\left(\sum_{t=1}^{T} \mathbf{F}(\mathbf{s}, \mathbf{o}, t)\right)\right\} \tag{3}$$

In order to find $\mathbf{s}^*$, one can apply a dynamic programming technique with a slightly modified version of the original Viterbi algorithm for HMMs [12]. To avoid an exponential–time search over all possible settings of $\mathbf{s}$, Viterbi stores the probability of the most likely path up to time t which accounts for the first t observations and ends in state s_i. We denote this probability to be $\varphi_t(s_i)$ $(0 \leq t \leq T-1)$ and $\varphi_0(s_i)$ to be the probability of starting in each state s_i. The recursion is given by:

$$\varphi_{t+1}(s_i) = \max_{s_j} \{\varphi_t(s_j)\exp\mathbf{F}(\mathbf{s}, \mathbf{o}, t+1)\} \tag{4}$$

The recursion stops when $t = T$ - 1 and the biggest unnormalized probability is $p_\theta^* = \text{argmax}_i[\varphi_T(s_i)]$. At this time, we can backtrack through the stored information to find the most likely sequence $\mathbf{s}^*$.

2.2 Training Conditional Random Fields

CRFs are trained by setting the set of weights $\theta = \{\lambda_1, \lambda_2, \ldots\}$ to maximize the log–likelihood, L, of a given training data set $\mathcal{D} = \{(\mathbf{o}^{(j)}, \mathbf{l}^{(j)})\}_{j=1}^{N}$:

$$L = \sum_{j=1}^{N} \log\left(p_\theta(\mathbf{l}^{(j)}|\mathbf{o}^{(j)})\right) = \sum_{j=1}^{N}\sum_{t=1}^{T} \mathbf{F}(\mathbf{l}^{(j)}, \mathbf{o}^{(j)}, t) - \sum_{j=1}^{N} \log Z(\mathbf{o}^{(j)}) \quad (5)$$

When the label sequences in the training dataset is complete, the likelihood function in exponential models such as CRFs is convex, thus searching the global optimum is guaranteed. However, the optimum can not be found analytically. Parameter estimation for CRFs requires an iterative procedure. It has been shown that quasi–Newton methods, such as L–BFGS [13], are most efficient [4]. This method can avoid the explicit estimation of the Hessian matrix of the log–likelihood by building up an approximation of it using successive evaluations of the gradient. L–BFGS is a limited–memory quasi–Newton procedure for unconstrained convex optimization that requires the value and gradient vector of the function to be optimized. The log–likelihood gradient component of λ_k is

$$\begin{aligned}\frac{\delta L}{\delta \lambda_k} &= \sum_{j=1}^{N}\left[\widetilde{C}_k(\mathbf{l}^{(j)}, \mathbf{o}^{(j)}) - \sum_{\mathbf{s}} p_\theta(\mathbf{s}|\mathbf{o}^{(j)}) C_k(\mathbf{s}, \mathbf{o}^{(j)})\right] \\ &= \sum_{j=1}^{N}\left[\widetilde{C}_k(\mathbf{l}^{(j)}, \mathbf{o}^{(j)}) - \mathrm{E}_{p_\theta} C_k(\mathbf{s}, \mathbf{o}^{(j)})\right] \end{aligned} \quad (6)$$

where $\widetilde{C}_k(\mathbf{l}^{(j)}, \mathbf{o}^{(j)}) = \sum_{t=1}^{T} f_k(\mathbf{l}_{t-1}^{(j)}, \mathbf{l}_t^{(j)})$ if λ_k is associated with an edge feature f_k and $= \sum_{t=1}^{T} g_k(\mathbf{o}^{(j)}, \mathbf{l}_t^{(j)})$ if λ_k is associated with a state feature g_k. Intuitively, it is the expectation (i.e., the count) of feature f_k (or g_k) with respect to the j^{th} training sequence of the empirical data $\mathcal{D}$. And $\mathrm{E}_{p_\theta} C_k(\mathbf{s}, \mathbf{o}^{(j)})$ is the expectation (i.e., the count) of feature f_k (or g_k) with respect to the CRF model p_θ.

The training process for CRFs requires to evaluate the log-likelihood function L and gradient vector $\{\frac{\delta L}{\delta \lambda_1}, \frac{\delta L}{\delta \lambda_2}, \ldots\}$ at each training iteration. This is very time-consuming because estimating the partition function $Z(o^{(j)})$ and the expected value $\mathrm{E}_{p_\theta} C_k(\mathbf{s}, \mathbf{o}^{(j)})$ needs an intensive forward-backward computation. This computation manipulates on the transition matrix M_t at every time position t of each data sequence. M_t is defined as follows,

$$M_t[l^{'}][l] = \exp\, \mathbf{F}(\mathbf{s}, \mathbf{o}, t) = \exp\, (\sum_i \lambda_i f_i(s_{t-1}, s_t) + \sum_j \lambda_j g_j(\mathbf{o}, s_t)) \quad (7)$$

To compute the partition function $Z(o^{(j)})$ and the expected value $\mathrm{E}_{p_\theta} C_k(\mathbf{s}, \mathbf{o}^{(j)})$, we need forward and backward vector variables α_t and β_t defined as follows,

$$\alpha_t = \begin{cases} \alpha_{t-1} M_t & 0 < t \leq T \\ \mathbf{1} & t = 0 \end{cases} \quad (8)$$

$$\beta_t^\top = \begin{cases} M_{t+1} \beta_{t+1}^\top & 1 \leq t < T \\ \mathbf{1} & t = T \end{cases} \quad (9)$$

$$Z(\mathbf{o}^{(j)}) = \alpha_T \mathbf{1}^\top \tag{10}$$

$$\mathrm{E}_{p_\theta} C_k(\mathbf{s}, \mathbf{o}^{(j)}) = \sum_{t=1}^{T} \frac{\alpha_{t-1}(f_k * M_t)\beta_t^\top}{Z(o^{(j)})} \tag{11}$$

3 Training CRFs on Multiprocessor Systems

3.1 The Need of Parallel Training of CRFs

In the sequential algorithm for training CRFs computing log-likelihood L and its gradient $\{\frac{\delta L}{\delta \lambda_1}, \frac{\delta L}{\delta \lambda_2}, \ldots\}$ is most time-consuming due to the heavy forward-backward computation on transition matrices. The L-BFGS update is very fast even if the log-likelihood function is very high dimensional. Therefore, the computational complexity of the training algorithm is mainly estimated from the former step.

The time complexity for calculating the transition matrix M_t in (7) is $\mathcal{O}(\bar{n}|\mathcal{L}|^2)$ where $|\mathcal{L}|$ is the number of class labels and $\bar{n}$ is the average number of *active* features at a time position in a data sequence. Thus, the time complexity to the partition function $Z(\mathbf{o}^{(j)})$ according to (8) and (10) is $\mathcal{O}(\bar{n}|\mathcal{L}|^2 T)$, in which T is the length of the observation sequence $\mathbf{o}^{(j)}$. And, the time complexity for computing the feature expectation $\mathrm{E}_{p_\theta} C_k(\mathbf{s}, \mathbf{o}^{(j)})$ is also $\mathcal{O}(\bar{n}|\mathcal{L}|^2 T)$. As a result, the time complexity for evaluating the log-likelihood function and its gradient vector is $\mathcal{O}(N\bar{n}|\mathcal{L}|^2 \bar{T})$, in which N is the number of training data sequences and T is now replaced by $\bar{T}$ - the average length of training data sequences. Because we train the CRF model m iterations, the final computational complexity of the serial training algorithm is $\mathcal{O}(mN\bar{n}|\mathcal{L}|^2 \bar{T})$. This computational complexity is for first-order Markov CRFs. If we use the second-order Markov CRFs in which the label of the current state depends on two labels of two previous states, the complexity is now proportional to $|\mathcal{L}|^4$, i.e., $\mathcal{O}(mN\bar{n}|\mathcal{L}|^4 \bar{T})$.

Although the training complexity of CRFs is polynomial with respect to all input parameters, the training process on large-scale datasets is still prohibitively expensive. In practical implementation, the computational time for training CRFs is even larger than what we can estimate from the theoretical complexity; this is because many other operations need to be performed during training, such as feature scanning, mapping between different data formats, numerical scaling (to avoid numerical problems), and smoothing. For example, training a first-order Markov CRF model for POS tagging ($|\mathcal{L}| = 45$) on about 1 million words (i.e., $N\bar{T} \simeq 1,000,000$) from the Wall Street Journal corpus (Penn TreeBank) took approximately 100 hours, i.e., more than 4 days.

In the point of view of machine learning, speeding up the training of CRFs is motivated by a couple of reasons. First, there are more large-scale annotated datasets in NLP and Bioinformatics. Further, unlike natural language sentences, biological data sequences are much longer, and thus need more time for training and inference. Second, generative models like CRFs can incorporate millions of

features. However, not all features are relevant. Feature selection for choosing most important and useful features from a huge set of candidates sometimes requires the model to be re-trained over and over again. Third, another challenge is that in many new application domains, the lack of labeled training data is very critical. Building large annotated datasets requires a lot of human resources. Semi-supervised learning is a way to build accurate prediction models using a small set of labeled data as well as a large set of unlabeled data because unlabeled data are widely available and easy to obtain. There are several approaches in semi-supervised learning like self- and co-training that also need the models to be trained again and again. Finally, building an accurate prediction model needs a repeated refinement because the learning performance of a model like CRF depends on different parameter settings. This means that we have to train the model several times using different values for input parameters and/or under different experimental setups till it reaches a desired output. Thus, accelerating the training process can save time for practitioners significantly.

3.2 The Parallel Training of CRFs

Input:
- Training data: $\mathcal{D} = \{(\mathbf{o}^{(j)}, \mathbf{l}^{(j)})\}_{j=1}^{N}$
- The number of parallel processes: P;
- The number of training iterations: m

Output:
- Optimal feature weights: $\theta^* = \{\lambda_1^*, \lambda_2^*, \ldots\}$

Initial Step:
- Generate features with initial weights $\theta = \{\lambda_1, \lambda_2, \ldots\}$
- Each process loads its own data partition $\mathcal{D}_i$

Parallel Training (each training iteration):
1. The root process broadcasts θ to all parallel processes
2. Each process P_i computes the local log-likelihood L_i and local gradient vector $\{\frac{\delta L}{\delta \lambda_1}, \frac{\delta L}{\delta \lambda_2}, \ldots\}_i$ on $\mathcal{D}_i$
3. The root process gathers and sums all L_i and $\{\frac{\delta L}{\delta \lambda_1}, \frac{\delta L}{\delta \lambda_2}, \ldots\}_i$ to obtain the global L and $\{\frac{\delta L}{\delta \lambda_1}, \frac{\delta L}{\delta \lambda_2}, \ldots\}$
4. The root process performs L-BFGS optimization search to update the new feature weights θ
5. If #iterations $< m$ then goto step 1, stop otherwise

Table 1. Parallel algorithm for training CRFs

As we can see from (5) and (6), the log-likelihood function and its gradient vector with respect to training dataset $\mathcal{D}$ are computed by summing over all training data sequences. This *nature sum* allows us to divide the training dataset into different partitions and evaluate the log-likelihood function and its gradient on each partition independently. As a result, the parallelization of the training process is quite straightforward.

How the Parallel Algorithm Works The parallel algorithm is shown in Table 1. The algorithm follows the master-slave strategy. In this algorithm, the training dataset $\mathcal{D}$ is randomly divided into P equal partitions: $\mathcal{D}_1, \ldots, \mathcal{D}_P$. At

the initialization step, each data partition is loaded into the internal memory of its corresponding process. Also, every process maintains the same vector of feature weights θ in its internal memory.

At the beginning of each training iteration, the vector of feature weights on each process will be updated by communicating with the master process. Then, the local log-likelihood L_i and gradient vector $\{\frac{\delta L}{\delta \lambda_1}, \frac{\delta L}{\delta \lambda_2}, \ldots\}_i$ are evaluated in parallel on distributed processes; the master process then gathers and sums those values to obtain the global log-likelihood L and gradient vector $\{\frac{\delta L}{\delta \lambda_1}, \frac{\delta L}{\delta \lambda_2}, \ldots\}$; the new setting of feature weights is updated on the master process using L-BFGS optimization. The algorithm will check for some terminating criteria to whether stop or perform the next iteration. The output of the training process is the optimal vector of feature weights $\theta^* = \{\lambda_1^*, \lambda_2^*, \ldots\}$. Although the stopping criteria can be difference in likelihood or in the norm of the parameter vector between two consecutive iterations, we usually use the iteration count because training CRFs according to those criteria might take too many iterations and of course suffer from overfitting problem.

Data Communication and Synchronization In each training iteration, the master process has to communicate with each slave process twice: (1) broadcasting the vector of feature weights and (2) gathering the local log-likelihood and gradient vector. These operations are performed using message passing mechanism. Let n be the number of feature weights and weights are encoded with "double" data type, the total amount of data needs to be transferred between the master and each slave is $8(2n+1)$. If, for example, $n = 1,500,000$, the amount of data is approximately 23Mb. This is very small in comparison with high-speed links among computing nodes on massively parallel processing systems. A barrier synchronization is needed at each training iteration to wait for all processes complete their estimation of local log-likelihood and gradient vector.

Data Partitioning and Load Balancing Load balancing is important to parallel programs for performance reasons. Because all tasks are subject to a barrier synchronization point at each training iteration, the slowest process will determine the overall performance. In order to keep a good load balance among processes, i.e., to reduce the total idle time of computing processes as much as possible, we attempt to divide data into partitions as equally as possible. Let $M = \sum_{j=1}^{N} |\mathbf{o}^{(j)}|$ be the total number of data observations in training dataset $\mathcal{D}$. Ideally, each data partition $\mathcal{D}_i$ consists of N_i data sequences having exactly $\frac{M}{P}$ data observations. However, this ideal partitioning is not always easy to find because the lengths of data sequences are different. To simplify the partitioning step, we accept an approximate solution as follows. Let δ be some integer number, we attempt to find a partitioning in which the number of data observations in each data partition belongs to the interval $[\frac{M}{P} - \delta, \frac{M}{P} + \delta]$. To search for the first acceptable solution, we follow the round-robin partitioning policy in which longer data sequences are considered first. δ starts from some small value and will be gradually increased until the first solution is satisfied.

4 Empirical Evaluation

We performed two important natural language processing tasks, text noun phrase chunking and all-phrase chunking, on large-scale datasets to demonstrate two main points: (1) the large reduction in computational time of the parallel training of CRFs on massively parallel computers in comparison with the serial training; (2) when being trained on large-scale datasets, CRFs tends to achieve higher prediction accuracy in comparison with the previous applied learning methods.

4.1 Experimental Environment

The experiments were carried out using our C/C++ implementation[3] of second-order Markov CRFs. It was designed to deal with hundreds of thousand data sequences and millions of features. It can be compiled and run on any parallel system supporting message passing interface (MPI). We used a Cray XT3 system (Linux OS, 180 AMD Opteron 2.4GHz processors, 8GB RAM per each, high-speed (7.6GB/s) interconnection among processors) for the experiments.

4.2 Text Chunking

Text chunking[4], an intermediate step towards full parsing of natural language, recognizes phrase types (e.g., noun phrase, verb phrase, etc.) in input text sentences. Here is a sample sentence with phrase marking: "[NP *Rolls-Royce Motor Cars Inc.*] [VP *said*] [NP *it*] [VP *expects*] [NP *its U.S. sales*] [VP *to remain*] [ADJP *steady*] [PP *at*] [NP *about 1,200 cars*] [PP *in*] [NP *1990*]."

4.3 Text Chunking Data and Evaluation Metric

We evaluated NP chunking and chunking on two datasets: (**1**) **CoNLL2000-L**: the training dataset consists of 39,832 sentences of sections from 02 to 21 of the Wall Street Journal (WSJ) corpus and the testing set includes 1,921 sentences of section 00 of WSJ; and (**2**) **25-fold CV Test**: 25-fold cross-validation test on all 25 sections of WSJ. For each fold, we took one section of WSJ as the testing set and all the others as training set.

Label representation for phrases is either IOB2 or IOE2. B indicates the beginning of a phrase, I is the inside of a phrase, E marks the end of a phrase, and O is outside of all phrases. The label path in IOB2 of the sample sentence is "B-NP I-NP I-NP I-NP B-VP B-NP B-VP B-NP I-NP I-NP B-VP I-VP B-ADJP B-PP B-NP I-NP I-NP B-PP B-NP O".

Evaluation metrics are precision ($pre. = \frac{a}{b}$), recall ($rec. = \frac{a}{c}$), and $F_{\beta=1} = 2 \times (pre. \times rec.)/(pre. + rec.)$; in which a is the number of *correctly* recognized phrases (by model), b is is the number of recognized phrases (by model), and c is the the number of actual phrases (by humans).

[3] PCRFs: http://www.jaist.ac.jp/~hieuxuan/flexcrfs/flexcrfs.html
[4] See the CoNLL-2000 shared task: http://www.cnts.ua.ac.be/conll2000/chunking

4.4 Feature Selection for Text Chunking

To achieve high prediction accuracy on these tasks, we train CRF model using the second-order Markov dependency. This means that the label of the current state depends on the labels of the two previous states. As a result, we have four feature types as follows rather than only two types in first-order Markov CRFs.

$$f_i(s_{t-1}, s_t) \equiv [s_{t-1} = l^{'}][s_t = l]$$
$$g_j(\mathbf{o}, s_t) \equiv [x_i(\mathbf{o}, t)][s_t = l]$$
$$f_k(s_{t-2}, s_{t-1}, s_t) \equiv [s_{t-2} = l^{''}][s_{t-1} = l^{'}][s_t = l]$$
$$g_h(\mathbf{o}, s_{t-1}, s_t) \equiv [x_h(\mathbf{o}, t)][s_{t-1} = l^{'}][s_t = l]$$

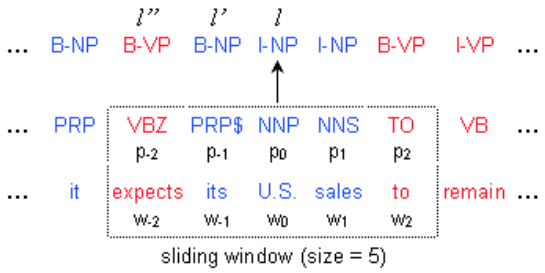

Fig. 1. An example of a data sequence

$w_{-2},\ w^*_{-1},\ w^*_0,\ w_1,\ w_2,\ w_{-1}w^*_0,\ w_0w_1$
$p_{-2},\ p^*_{-1},\ p^*_0,\ p_1,\ p_2,\ p_{-2}p_{-1},\ p_{-1}p^*_0,\ p_0p_1,\ p_1p_2$
$p_{-2}p_{-1}p_0,\ p_{-1}p_0p_1,\ p_0p_1p_2,\ p_{-1}w^*_{-1},\ p_0w^*_0$
$p_{-1}p_0w^*_{-1},\ p_{-1}p_0w^*_0,\ p_{-1}w_{-1}w^*_0,\ p_0w_{-1}w^*_0,\ p_{-1}p_0p_1w_0$

Table 2. Context predicate templates for text chunking

Figure 1 shows a sample training data sequence for text chunking. The top half is the label sequence and the bottom half is the observation sequence including tokens (words or punctuation marks) and their POS tags. Table 2 describes the context predicate templates for text chunking. Here w denotes a token; p denotes a POS tag. A predicate template can be a single token (e.g., the current word: w_0), a single POS tag (e.g., the POS tag of the previous word: p_{-1}), or a combination of them (e.g., the combination of the POS tag of the previous word, the POS tag of the current word, and the current word: $p_{-1}p_0w_0$). Context predicate templates with asterisk ($*$) are used for both state feature type 1 (i.e., g_j) and state feature type 2 (i.e., g_h). We also apply rare (cut-off) thresholds for both context predicates and state features (the threshold for edge features is zero). Those predicates and features whose occurrence frequency is smaller than 2 will be removed from our models to reduce overfitting.

4.5 Experimental Results of Text Chunking

Methods	NP chunking $F_{\beta=1}$	All-phrase chunking $F_{\beta=1}$
Ours (majority voting among 16 CRFs)	**96.74**	**96.33**
Ours (CRFs, about 1.3M - 1.5M features)	**96.59**	**96.18**
Kudo & Matsumoto 2001 (voting SVMs)	95.77	–
Kudo & Matsumoto 2001 (SVMs)	95.34	–
Sang 2000 (system combination)	94.90	–

Table 3. Accuracy comparison of NP and all-phrase chunking on the CoNLL2000-L

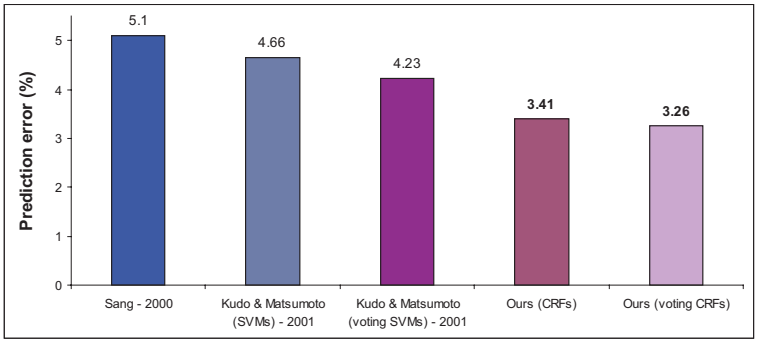

Fig. 2. Error rate comparison for noun phrase chunking on CoNLL2000-L dataset

Datasets	n_{00}	n_{01}	n_{10}	n_{11}	χ^2	χ	Null hypothesis
CoNLL2000-L	304	336	525	45286	41.0499	6.4070	REJECT

Table 4. Statistical (McNemar) test for comparing prediction models: between our second-order Markov **CRFs** and **SVMs** (Kudo & Matsumoto 2001)

Table 3 shows the comparison of F_1-scores of NP and all-phrase chunking tasks on the CoNLL2000-L dataset among state-of-the-art chunking systems. Figure 2 shows our improvement in accuracy in comparison with the previous work in a more visual way. Our model reduces error by 22.93% on NP chunking relative to the previous best system, i.e., Kudo & Matsumoto's work.

We performed McNemar's test for noun phrase chunking on the CoNLL2000-L dataset mentioned above. The statistical test was done between our second-order Markov CRFs and SVMs (Kudo & Matsumoto 2001) [17]. Table 4 shows

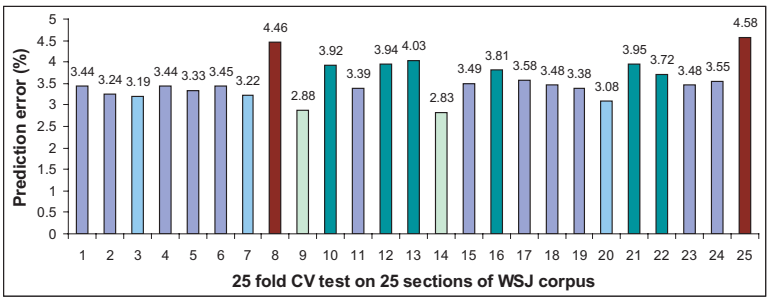

Fig. 3. 25-fold cross-validation test of NP chunking on the whole 25 sections of WSJ

all information about the McNemar's tests: n_{00} be the number of examples (i.e., words) in the testing set that are misclassified by both our CRF and SVM models; n_{01} be the number of examples misclassified by our CRF but not by SVM; n_{10} be the number of examples misclassified by SVM but not by our CRF; and n_{11} be the number of misclassified by neither our CRF nor SVM. Where $n = n_{00} + n_{01} + n_{10} + n_{11}$ is the total number of examples in the testing set. This test takes into account the number of examples misclassified by our CRF model when compared to the SVM model of Kudo & Matsumoto and vice-versa. The null hypothesis (i.e., H_0) is that both prediction models have equal error rate, i.e., $n_{01} = n_{10}$. We have the chi-square statistic with one degree of freedom,

$$\chi^2 = \frac{(|n_{01} - n_{10}| - 1)^2}{n_{01} + n_{10}}$$

McNemar's test accepts the null hypothesis at significance level α if $\chi^2 \leq \chi^2_{\alpha,1}$. The typical value for α is 0.05 and $\chi^2_{0.05,1} = 3.84$. Since the null hypothesis is rejected and n_{01} is greater than n_{10}, we conclude that our CRF models outperform the SVM models of Kudo & Matsumoto on this task.

In order to investigate chunking performance on the whole WSJ, we performed a 25-fold CV test on all 25 sections. We trained totally 50 CRF models for 25 folds for NP chunking using two label styles IOB2, IOE2 and only one initial value of θ (= .00). The number of features of these models are approximately 1.5 million. Figure 3 shows the lowest error rates of those 25 sections.

4.6 Computational Time Measure and Analysis

We also measured the computational time of the CRF models the Cray XT3 system. For example, training 130 iterations of NP chunking task on CoNLL2000-L dataset using a single process took 38h57' while it took only 56' on 45 parallel processes. Similarly, each fold of the 25-fold CV test of NP chunking took an average training time of 1h21' on 45 processes while it took approximately 56h on one process. All-phrase chunking is much more time-consuming. This is because the number of class labels is $|\mathcal{L}| = 23$ on CoNLL2000-L. For example, serial training on the CoNLL2000-L requires about 1348h for 200 iterations (i.e., about

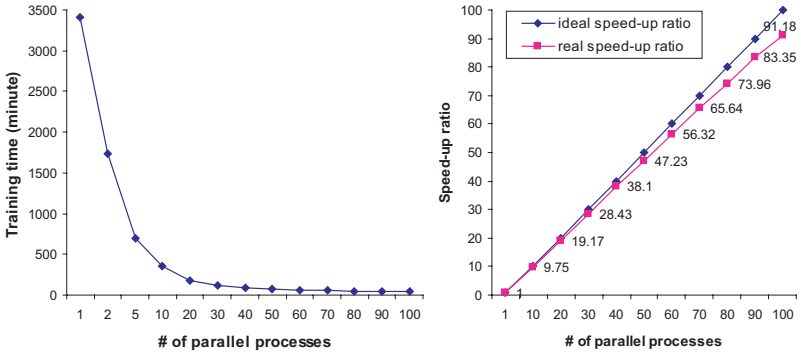

Fig. 4. The computational time of parallel training and the speed-up ratio of the first fold (using IOB2) of 25-fold CV test on WSJ

56 days) whereas it took only 17h46' on 90 parallel processes. Figure 4 depicts the computational time and the speed-up ratio of the parallel training CRFs on the Cray XT3 system.

5 Conclusions

We have presented a high-performance training of CRFs on large-scale datasets using massively parallel computers. And the empirical evaluation on text chunking with different data sizes and parameter configurations shows that second-order Markov CRFs can achieved a significantly higher accuracy in comparison with the previous results, particularly when being provided enough computing power and training data. And, the parallel training algorithm for CRFs helps reduce computational time dramatically, allowing us to deal with large-scale problems not limited to natural language processing.

References

1. Pinto, D. McCallum, A, Wei, X., and Croft, B. (2003). Table extraction using conditional random fields. The 26th ACM SIGIR.
2. Kristjansson, T., Culotta, A., Viola, P., and McCallum, A. (2004). Interactive information extraction with constrained conditional random fields. The 19th AAAI.
3. Cohn, T., Smith, A., and Osborne, M. (2005). Scaling conditional random fields using error-correcting codes. The 43th ACL.
4. Sha, F. and Pereira, F. (2003). Shallow parsing with conditional random fields. HLT/NAACL.
5. Kumar, S. and Hebert, M. (2003). Discriminative random fields: a discriminative framework for contextual interaction in classification. The IEEE CVPR.
6. Quattoni, A., Collins, M., and Darrell, T. (2004) "Conditional random fields for object recognition", The 18th NIPS.

7. Torralba, A., Murphy, K., and Freeman, F. (2004) "Contextual models for object detection using boosted random fields", The 18th NIPS.
8. He, X., Zemel, R.S., and Carreira-Perpinan, M.A. (2004) "Multiscale conditional random fields for image labeling", The IEEE CVPR.
9. Lafferty, J., Zhu, X., and Liu, Y. (2004) "Kernel conditional random fields: representation and clique selection", The 20th ICML.
10. Liu, Y., Carbonell, J., Weigele, P., and Gopalakrishnan, V. (2005) "Segmentation conditional random fields (SCRFs): a new approach for protein fold recognition", The 9th RECOMB.
11. Lafferty, J., McCallum, A., and Pereira, F. (2001) "Conditional random fields: probabilistic models for segmenting and labeling sequence data", The 18th ICML.
12. Rabiner, L. (1989) "A tutorial on hidden markov models and selected applications in speech recognition", Proc. of IEEE, vol.77, no.2, pp. 257-286.
13. Liu, D. and Nocedal, J. (1989) "On the limited memory bfgs method for large-scale optimization", Mathematical Programming, vol.45, pp. 503-528.
14. Pietra, S.D., Pietra, V.D., and Lafferty, J. (1997) "Inducing features of random fields", IEEE PAMI, 19(4):380–393.
15. McCallum, A. (2003) "Efficiently inducing features of conditional random fields", The 19th UAI.
16. Sang, E. (2000) "Noun phrase representation by system combination", The ANLP/NAACL.
17. Kudo, T. and Matsumoto, Y. (2001) "Chunking with support vector machines", The NAACL.
18. Chen, S. and Rosenfeld, R. (1999) "A gaussian prior for smoothing maximum entropy models", Technical Report CS-99-108, CMU.

A Parallel Feature Selection Algorithm from Random Subsets

Daniel J. Garcia, Lawrence O. Hall, Dmitry B. Goldgof, and Kurt Kramer

Department of Computer Science and Engineering
4202 E. Fowler Ave. ENB118
University of South Florida
Tampa, FL 33620, USA
(djgarcia,hall,goldgof,kkramer)@csee.usf.edu

Abstract. Feature selection methods are used to find the set of features that yield the best classification accuracy for a given data set. This results in lower training and classification time for a classifier, a support vector machine here, and better classification accuracy. Feature selection, however, may be a time consuming process unfit for real time application. In this paper, we explore a feature selection algorithm based on support vector machine training time. It is compared with the Wrapper algorithm. Our approach can be run on all available processors in parallel. Our feature selection approach is ideal if new features need to be selected during data acquisition, where a fast, approximate approach may be advantageous. Experimental results indicate that the training time based method yields feature sets almost as good as the Wrapper method, while requiring considerably less computation time.

Keywords: Feature Selection, Parallelism, Random Subsets, Wrappers, SVM.

1 Introduction

Support vector machines (SVMs)[1] are learning algorithms which result in a model that can be used to classify data. The details of the inner workings of SVMs are beyond the scope of this paper. For our purposes, SVMs use labeled data to construct a classifier, and then use it to classify unknown data. In this paper, the data being analyzed are plankton images obtained from a device called SIPPER (Shadow Image Particle Profiling Evaluation Recorder) [2]. In order for a support vector machine to be able to classify these images, features are extracted from them. These features are used by the SVM to create the support points that will differentiate between images. These features can be numerous and include characteristics such as height, weight, shape, length, transparency, and texture.

The use of all the available features does not guarantee the best accuracy, training, and classification time. It is possible for a subset of features to have better accuracy, training, and classification time. Also of importance is the fact

that the process of training a SVM is faster if fewer features are used. It is for this reason that feature selection processes are necessary in order to effectively train a classifier. Feature selection is the process through which an "optimal" group of features for a given data set is found. This process may be a time consuming one, and is typically not well suited for real time applications. The goal is to create a feature selection algorithm that is able to complete its execution in a short enough amount of time to allow it to be implemented in the field during data acquisition, while retaining high classification accuracy. Our hypothesis is that sets of features which result in faster SVM training times can be exploited to create overall feature sets which can be used to build a high accuracy classifier. It is expected that less training time will be required to find the boundaries with features that will enable higher accuracy classifiers to be built.

There are many feature selection methods. In fact, there are a number which are relatively specific for support vector machines [3, 4]. The recursive feature elimination approach has been used with success with SVM's [4]. Space limitations prevent us from doing detailed comparisons, but we do present an alternative, feature selection approach that works with SVM's.

2 Random Feature Selection

Feature selection methods are applied to all the features describing a data set to find a subset of features that best describe that data set. The feature selection method proposed in this paper can be divided into two stages. The first stage consists of generating a number of feature sets of fixed size, then running a 10 fold cross validation using only the features found in these sets, to determine how well they are able to classify the data. It is important to emphasize that the features in these sets are randomly selected out of the pool of all features, and thus these sets are generated in a very short amount of time. The sets of features are then sorted by a given criteria, such as training time (here) or the number of support vectors generated, and the best of these randomly generated sets are selected for the second stage of the algorithm.

For the second stage of the method we have a number of ranked feature sets. Using these sets, a new set composed of the union of the features found in the selected sets is created. At this point, the classifier is trained using the newly created feature sets, and then it is tested against a previously unseen test set to see how well it performs. The number of feature sets selected for the second stage of the method can vary from 2 to the number of sets generated during the first stage of the process. Figure 1 shows a flowchart of the random sets feature selection method. The algorithm had minimal sensitivity to increasing the number of feature sets. The choice of numbers of feature sets to union needs more exploration with the goal of smaller numbers in the union resulting in fewer chosen features.

One very distinct characteristic of the random sets approach is that the random sets are all independent of each other. Feature selection algorithms usually go through a large number of possible combinations of features in order to find

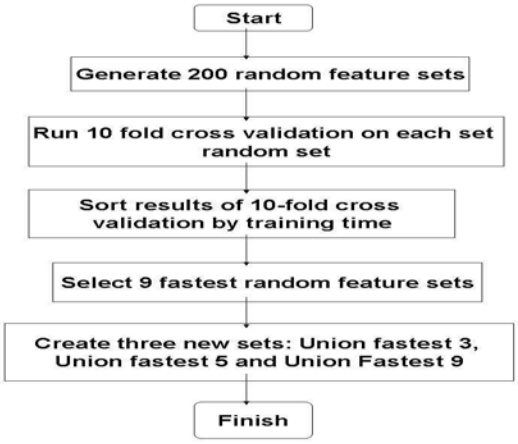

Fig. 1. Random Sets Flow Chart

the best one. However, since the number of features could possibly reach the hundreds, the number of possible combinations grows at a very fast rate. It is for this reason that existing feature selection algorithms do not search for possible combinations blindly, instead they do it intelligently. This means that there is some logic guiding the search, and the next step in the search process is partly based on previous steps. The implications of this characteristic is that these feature selection methods cannot fully take advantage of parallel processing because future steps in the process need to wait for previous steps to finish. The random sets method, on the other hand, does allow parallel processing. The random feature sets created are completely independent from each other, and all of them are evaluated during the same step in the algorithm. For this reason, it is possible for every single random set created to be evaluated in parallel; greatly reducing the time it takes for this feature selection method to finish its task. In this work, timings are reported with all training done on 1 processor. One could divide by approximately the number of random feature sets evaluated (there will be some overhead) to look at the parallel computing time advantage. The reader will see the speed-ups are quite impressive even without parallelism.

3 Wrappers

An alternative algorithm for feature selection is also presented. Feature selection methods usually work by trying combinations of features from the original pool of features and then choosing the combination that yields the best results. One such method consists of organizing the feature combinations in a tree structure. In this organization, the nodes of the tree are simply a given combination of features. This is the approach taken by the Wrapper feature selection method [5].

A given combination of features is passed to a learning algorithm for evaluation and then the results are obtained and kept for later comparison. The results from the learning algorithm are then used to intelligently search the tree structure.

To illustrate this approach, let us assume we have n features. The root of the tree is the set containing all n features. This set is analyzed first and the accuracy is stored. The next step in the algorithm is to choose the best stored results and then analyze every combination of the number of features in that set minus one. In the current case, only the results of analyzing one set have been obtained, so the next step would be to analyze every combination of n -1 features and store the results. At this point, a best first search is done, which means that the best case, gauged by the classification accuracy, is selected in order to repeat the process of searching for every combination of features of length s - 1, where s is the number of features in the most recently selected feature set. Thus, the next step is to look at all (n - 1) - 1 = n - 2 subsets of the best n - 1 grouping.

The stopping criterion was the analysis of a fixed number of new feature sets without finding a new set with clearly higher accuracy [6]. We did allow sub-optimal (5^{th} best) sets to be examined with a low probability.

This feature selection method can't take full advantage of parallel processing. This is so because the search has to wait for the results of all the processed combinations before it can select the best next case. Suppose combinations of size s are being analyzed, all of these combinations come from a parent of size s + 1. Theoretically, every combination of size s in this case can be processed at the same time if enough processors are present. However, the result from these sets will be considered for the next best case, which forces this method to wait until every set of size s is evaluated before it can continue.

4 Data Set and Parameters

The results presented in this paper were obtained from experiments using a data set made up of plankton images obtained from the SIPPER device. The data set includes 5 different classes and consists of 8440 images total, with 1688 images per class. This image set was split into three smaller subsets for the purpose of the experiments. Two of the subsets contain 1000 images, with 200 images per class; and the remaining one contains 6440 images, with 1288 images per class. Feature selection is done on one of the sets with 1000 images; the Test Set is the other set with 1000 images; and the Training Subset had 6440 images. The data is these files has been stratified for 10 fold cross validations. The SVM used as the classifier is a modified version of libSMV [7], the parameters for the SVM are C = 16, Gamma = 0.04096, and the Gaussian radial basis function (RBF) was used as the kernel. The sequential minimal optimization (SMO) algorithm was the optimization algorithm used. For more information regarding the parameter tuning process and the RBF kernel, please refer to [8].

5 Experiments and Results

The procedure for the experiments was the following. First, from the original pool of 47 features extracted from the plankton images, 200 random subsets with 10 features each were created. We did some empirical tests and found these numbers to be the best of a range of reasonably equivalent choices. Clearly, the selection will make a difference.

A ten fold cross validation was done on the Feature Selection data set using each one of these sets independently, and the time it took to train the SVM using these sets was recorded. Next, the 9 feature sets associated with the shortest training times were selected for the second stage of our method. Three new sets were created by using the union of the features found in the selected sets: the union of the 3 sets, the union of the 5 sets, and the union of the 9 sets, respectively associated with the ordered shortest training time. Finally, the Feature Selection data set and the Training Subset data set are put together into a joint data set and a classifier is trained on this joint set using the three new feature sets. Then the classifier is tested against the unseen Test Set to obtain the final results.

The whole procedure is repeated five times with five different randomly chosen stratified sets of data. The results reported are the average values of the five experiments.

For the Wrapper approach, the procedure was the following. First, a search was performed on the Feature Selection set using a 5 fold cross validation to evaluate the feature sets. This will yield an accuracy value for each level in the search tree, thus we get an accuracy value for the best sets of n features, where n goes from 1 feature to all features. Next, the union of the Training Subset and the Feature Selection set are used to train the classifier using the best set of features at a specific level in the tree, and then the classifier is tested against the Test Set to determine how accurate it is on unseen data.

As with the random sets approach, this whole procedure is repeated five times over the same data sets as the random sets method and the results reported are the average values of the five different experiments.

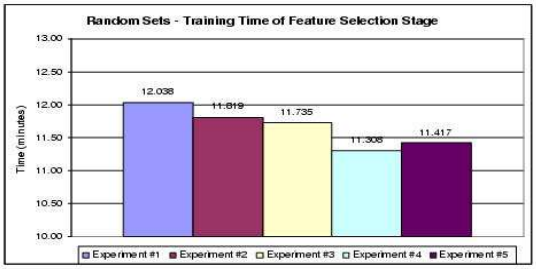

Fig. 2. Random Sets Average Training Time of Feature Selection Stage

The most important aspect of the random sets method is how fast it is. Figures 2 and 3 are graphs of the average training time of the feature selection stage for both the random sets approach, and the Wrappers approach for the five experiments. The training time of the feature selection stage of the random sets approach consists of the time it takes to train on the n random feature sets of fixed size, in this particular case, 200. The training time of the feature selection stage of the Wrapper method consists of the time it takes to train all of the combinations of features the Wrapper method tries while looking for an "optimal" set of features.

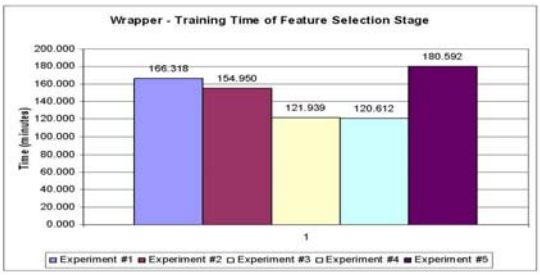

Fig. 3. Wrapper Average Training Time of Feature Selection Stage

There is a significant time difference between these two algorithms. The difference in time may be attributed to several factors, but chief among them, is the amount of work that each algorithm must do before finding their best feature sets. A good indicator of the amount of work each algorithm performs is the number of feature combinations evaluated during the search. The Wrapper approach consists of intelligently searching combinations of features starting with all features and reducing the number of features in the combinations as it progresses. The average number of combinations evaluated by the 5 iterations of the Wrapper method done here was 9372. Meanwhile, by the very definition of the random sets method, a fixed number of random feature sets needs to be evaluated. For the experiments carried out in this paper, only 200 random combinations were attempted for each of the five iterations, thus the average number of combinations across the five repetitions of the experiment is 200.

Specifically, the random sets approach shows a distinctly shorter training time during feature selection, and moderately shorter evaluation time when compared to the Wrapper. The average training time of the 200 random sets across the 5 individual experiments is approximately 700 seconds, or 11 minutes and 40 seconds. On the other hand, the average training time of all the combinations tried by the Wrapper approach across the 5 iterations of the experiment is approximately 9000 seconds, or 150 minutes. Thus, on average, there is a difference of approximately 2 hours and 19 minutes in time between the two feature selection methods.

The difference in evaluation time is not as great as the difference between the training times. Across the 5 experiments, the average evaluation time for the 200 randomly created sets is 43 seconds. The Wrapper, on the other hand, spent on average 2498 seconds, or 41 minutes and 38 seconds, on evaluation time. Adding the training time and the evaluation time together we get the total CPU time spent on each method. The random sets method spent an average of 744 seconds, or roughly 12 minutes to complete; on the other hand, the Wrapper method spent an average of 11,431 seconds, roughly 191 minutes, or 3 hours and 11 minutes, to finish.

Figure 4 shows a graph of the average accuracy for the union of the 3 fastest sets found by the random sets method. Because the set resulting from the union of these three sets contained 23 features, the accuracy is being compared with the average accuracy obtained by the Wrapper approach at 23 features. The remaining accuracy in Figure 4 is the best observed accuracy across all experiments, that is, taking into consideration both the random sets approach and the Wrapper approach. This accuracy was achieved by the Wrapper approach using the best 42 features found during the search and it is provided as a measure of how close the individual methods get to the best possible accuracy. Figure 5 provides training time measurements for these sets.

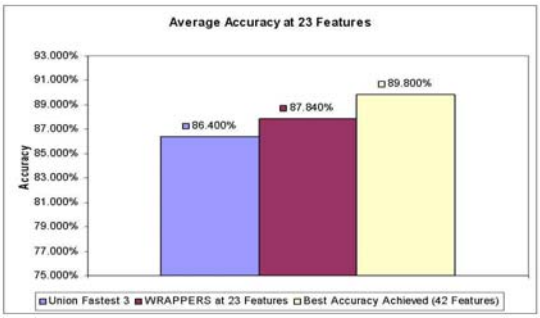

Fig. 4. Average Accuracy at 23 Features

At 23 features, the Wrapper approach was more accurate than the random sets approach by 1.44%. In turn, the best accuracy achieved was superior to the union of the fastest 3 random sets by 3.4%, and superior to Wrappers at 23 features by 1.96%. Figure 5 shows the average training times for the sets shown in Figure 4. Not surprisingly, the 23 features found by the Wrapper method are, on average, faster for training than both the 23 features found by the random sets, due to the Wrappers' ability to likely find the best 23 features, and the set of all features. The feature set found by the random sets is also faster for training than the set of all the features.

Figure 6 is a graph of the average accuracy values for the experiments involving the union of the 5 random feature sets that resulted in the fastest training

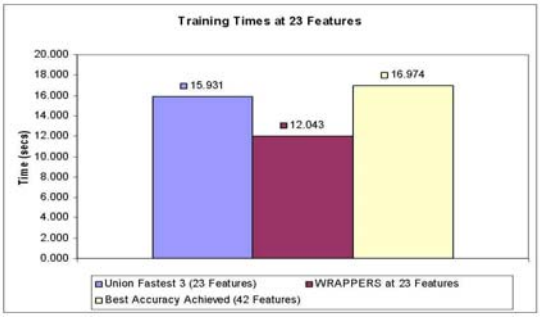

Fig. 5. Training Time at 23 Features

times. Using the fastest 5 random feature sets to train , the number of features in the union of these sets has increased to 31. For this reason, the accuracy of the union of the 5 fastest sets is being compared to the average accuracy of the Wrapper at 31 features. The best achieved accuracy is also provided for comparison purposes. Figure 7 shows the training time information for these sets.

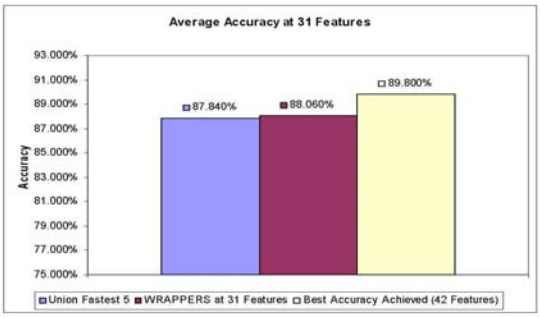

Fig. 6. Average Accuracy at 31 Features

As Figure 6 shows, the union of the 5 fastest to train sets, using 31 features, is inferior to Wrappers at the same number of features by only 0.22%. At this point, the best accuracy achieved is only 1.96% more than the random sets method, and 1.74% above the Wrappers. Figure 7 shows the training times for these three sets. Once again, the features found by the Wrappers were faster than the features found by the random sets.

Figure 8 is a graph of the final step of the random sets approach, when all the selected feature sets, in this case 9, were taken together to form a set consisting of the union of all the features in these sets. This new set contains 40 features; its

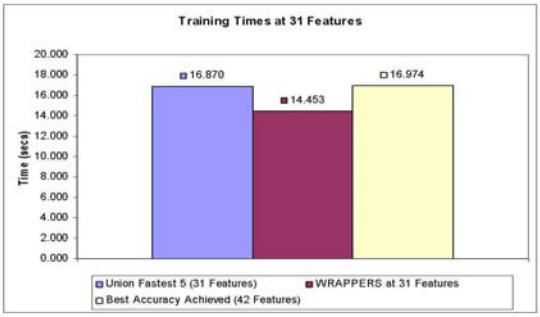

Fig. 7. Training Time at 31 Features

average accuracy is being compared with the average accuracy of the Wrappers at 42 features, and with the best accuracy achieved. Figure 9 is a graph of the average training times of the relevant feature sets.

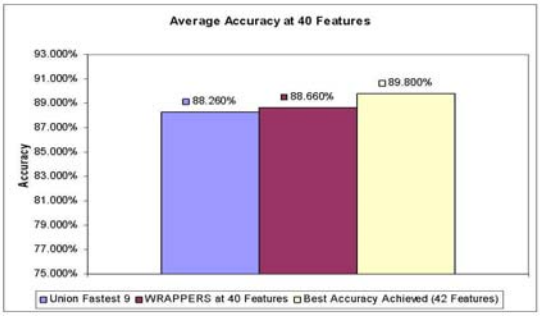

Fig. 8. Average Accuracy at 42 Features

As can be seen in Figure 8, with 40 features, the Wrapper method is only 1.14% less accurate than the best achieved accuracy. The random sets method, using the 40 features it found, is only 1.54% less accurate than the best accuracy obtained. Figure 9 shows a more interesting result, with 2 fewer features, the union of the fastest 9 sets has a higher training time than Wrappers at 42 features, that is, the best set of features found throughout the experiments. The reason for this behavior, as was stated previously, is that some features, in actuality, hinder the training process by making less clear the boundary between classes of images.

Figures 4, 6 and 8 show an interesting trend where the Wrapper approach performs slightly better than the random sets approach, and the sets of features that the Wrapper method produces are slightly faster for the training process

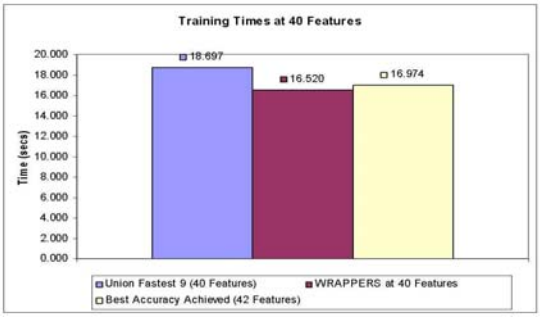

Fig. 9. Training Time at 42 Features

also. The reasoning is that the Wrapper method is a deeper, logically driven search, while our approach has a random element. This means that for any particular n, the Wrapper method should have approximated the best set consisting of n features, while the random sets could have found those features, it is not highly likely that it did. The advantage of the random sets method is that it finds sets of features that almost mirror the performance of the sets found by the Wrapper, but it does so in **considerably less time**. Time saving is the greatest asset of the random set method.

The random sets method is based on the hypothesis that the features that allow a SVM to train faster on a specific set of data are, in fact, the features better suited for that particular set of data. To test this hypothesis, the inverse of the hypothesis was used as the basis of the random set method and applied to one of the five data sets created for the random sets experiments. Using the inverse of the hypothesis implies selecting the "best" feature sets based on the fact they take the longest time to train; thus instead of selecting the fastest, to train, 9 sets to take to the second stage of the random sets method, the slowest 9 sets were selected. The results obtained from this experiment are compared to the result obtained from using the previously described random sets method on the same data set.

Figure 10 shows the accuracy for the union of 3 sets, the union of 5 sets, and the union of 9 sets as we use the fastest 9 random sets and the slowest 9 random sets. As can be seen in Figure 10, when the union operation is performed on the fastest sets, the accuracy is significantly higher in all three cases. The superiority of the features is shown in Figure 11, which gives the number of features in the relevant sets. Notice how the union of the fastest, to train, 3 sets actually has 3 fewer features than the union of the slowest 3 sets, however it is 10.6% more accurate. The accuracy continues to be higher for the union of the fastest 5 sets, and the union of the fastest 9 sets, however, the difference in accuracy becomes smaller as the number of features involved increases.

Figure 12 is a graph of the average accuracies of the random sets method and the Wrapper Method vs. the number of features in each of the sets. The accuracy

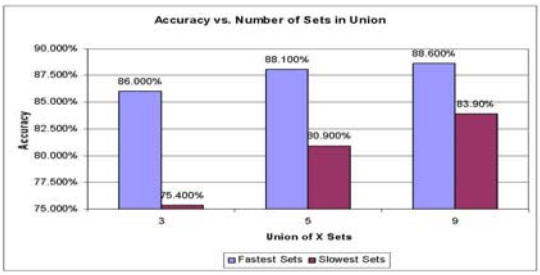

Fig. 10. Accuracy of Union vs. Number of Sets in Union

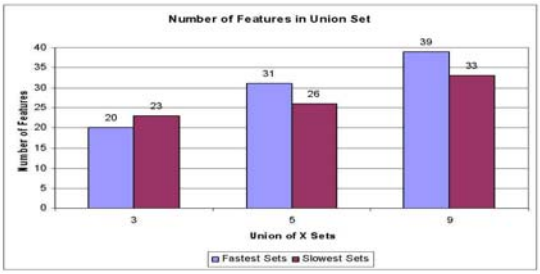

Fig. 11. Number of features in set vs. Number of Sets in Union

curve for the Wrapper method shows an increase as the number of features increases, reaching the highest average accuracy at 40 features. The random sets method is represented by three points, each representing the average accuracy of the union of the fastest 3, fastest 5, and fastest 9 random feature sets. Figure 12 clearly shows that the random sets method is able to find feature sets which can be used to create classifiers of comparable accuracy to those found by the Wrapper method, with the advantage that it does so in much less time.

6 Conclusion

As has been shown, using random feature sets as a feature selection tool provides benefits for learning algorithms. Real time application is one of the greatest benefits, perhaps allowing a limited feature selection algorithm to be run as new data is gathered. The random set approach is fast, can result in a very accurate classifier, and takes great advantage of available parallel processing. Each feature set can be evaluated in parallel. The Wrapper approach, on the other hand, was much slower but consistently more accurate. If accuracy is of the utmost importance, and feature selection time is no issue, the Wrapper method should be used; however, if time is critical, the random sets approach provides competitive accuracy while taking much less time.

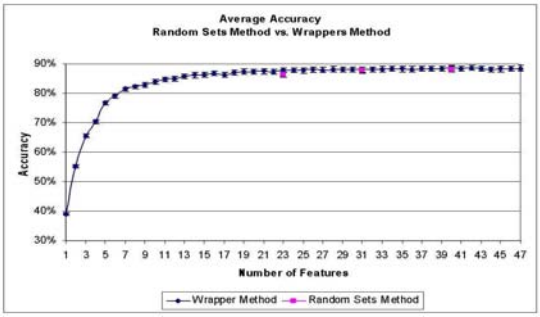

Fig. 12. Average Accuracy Random Sets Method vs. Wrapper Method

A real time application of the random set approach is the analysis of plankton on a cruise. The random set approach allows fast feature selection as different organisms are encountered. It is true that the accuracy will likely be slightly less than the best possible, but the difference in accuracy does not appear to be significant and it does allow for near real time optimization.

Acknowledgements: This research was partially supported by the United States Navy, Office of Naval Research, under grant number N00014-02-1-0266, the NSF under grant EIA-0130768 and by the Department of Energy through the Advanced Strategic Computing Initiative (ASCI) Visual Interactive Environment for Weapons Simulation (VIEWS) Data Discovery Program Contract number: DEAC04-76DO00789.

References

1. Tong Luo, Kurt Kramer, Dmitry B. Goldgof, Scott Samson Lawrence O. Hall, Andrew Remsen, and Thomas Hopkins. Recognizing Plankton from Shadow Image Particle Evaluation Recorder. *IEEE trans. on system, man and cybernetics-part B:cybernetics*, 34(4), 2004.
2. S. Samson, T. Hopkins, A. Remsen, L. Langebrake, T. Sutton, and J. Patten. A System for High Resolution Zooplankton Imaging. *IEEE Journal of Oceanic Engineering*, 26(4):671–676, 2001.
3. I. Guyon, S. Gunn, M. Nikravesh, and L.A. Zadeh, editors. *Feature Extraction Foundations and Applications*. Springer, 2006.
4. I. Guyon, J. Weston, and S. Barnhill. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1-3):389–422, 2002.
5. Ron Kohavi and George H. John. Wrappers for Feature Subset Selection. *Artificial Intelligence Archive*, 97:273–324, 1997.
6. Kurt A. Kramer. Identifying Plankton from Grayscale Silhouette Images. Master's thesis, University of South Florida, 2005.
7. Chih-Chung Chang and Chih-Jen Lin. A Library for Support Vector Machines, libsvm. http://www.csie.ntu.edu.tw/-cjlin/libsvm.
8. T. Luo, K. Kramer, D. Goldgof, L. Hall, S. Samson, A. Remsen, and T. Hopkins. Active Learning to Recognize Multiple Types of Plankton. In *International Conference on Pattern Recognition (ICPR)*, Cambridge, UK, August 2004.

Author Index