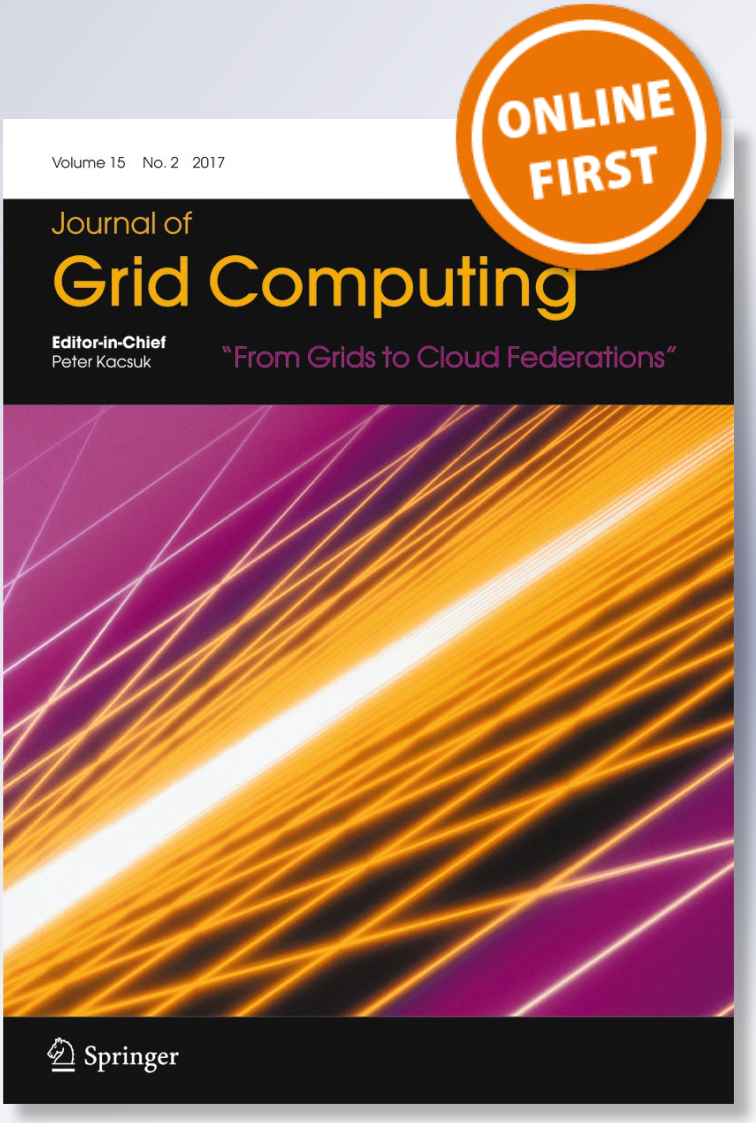
A Parallel Conditional Random Fields Model Based on Spark Computing Environment

Zhuo Tang, Zhongming Fu, Zherong Gong, Kenli Li & Keqin Li

Journal of Grid Computing
From Grids to Cloud Federations

ISSN 1570-7873

J Grid Computing
DOI 10.1007/s10723-017-9404-4



Springer

J Grid Computing
DOI 10.1007/s10723-017-9404-4

A Parallel Conditional Random Fields Model Based on Spark Computing Environment

Zhuo Tang · Zhongming Fu · Zherong Gong · Kenli Li · Keqin Li

Received: 6 April 2017 / Accepted: 14 June 2017

Abstract As one of the famous probabilistic graph models in machine learning, the conditional random fields (CRFs) can merge different types of features, and encode known relationships between observations and construct consistent interpretations, which have been widely applied in many areas of the Natural Language Processing (NLP). With the high-speed development of the internet and information systems, some performance issues are certain to arise when the traditional CRFs deals with such massive data. This paper proposes SCRFs, which is a parallel optimization of CRFs based on the Resilient Distributed Datasets (RDD) in the Spark computing framework. SCRFs optimizes the traditional CRFs from these stages: First, with all features are generated in parallel, the intermediate data which will be used frequently are all cached into the memory to speed up the iteration efficiency. By removing the low-frequency features of the model, SCRFs can also prevent the overfitting of the model to improve the prediction effect. Second, some specific features are dynamically added in parallel to correct the model in the training process. And for implementing the efficient prediction, a max-sum algorithm is proposed to infer the most likely state sequence by extending the belief propagation algorithm. Finally, we implement SCRFs base on the version of Spark 1.6.0, and evaluate its performance using two widely used benchmarks: *Named Entity Recognition* and *Chinese Word Segmentation*. Compared with the traditional CRFs models running on the Hadoop and Spark platforms respectively, the experimental results illustrate that SCRFs has obvious advantages in terms of the model accuracy and the iteration performance.

Keywords Conditional random fields · Feature selection · Machine learning · Parallel computing · Spark

1 Introduction

With the rapid development of information society, the data which generated from the internet is rapidly growing with the index grade, which usually have the following characteristics: large amount, high dimension, complex structure and containing much noises, but also have a widespread application prospect [1]. The traditional sequential data processing algorithms are not good enough to analyze this large volume of data especially for the machine learning model with high iterative calculation [2].

Z. Tang (✉) · Z. Fu · Z. Gong · K. Li
College of Information Science and Engineering, Hunan University, Changsha 410082, China
e-mail: ztang@hnu.edu.cn

K. Li
Department of Computer Science, State University of New York, New Paltz, New York 12561, USA
e-mail: lik@newpaltz.edu

Published online: 04 July 2017

Conditional random fields (CRFs) is precisely this kind of model: a type of conditional probability model with large amount of calculation for the parameters evaluation [3]. It has been widely applied in many applications, such as classifying regions of an image [5, 6], biomedical named entity [7, 8], and text annotation [9, 10] with satisfactory results. The advantage of the CRFs model is the ability to express long-distance-dependent and overlapping features, which is a significant superiority compared with the generative statistical model. As another probabilistic graph model, Hidden Markov Models (HMMs) [11, 12] is unable to use the complex features for its strict assumption of independence. Furthermore, CRFs can overcome the label bias question which exists in other discriminate model, such as maximum entropy Markov model (MEMMs) [13], etc.

However, for the model which complicated with many more parameters, the training time of CRFs is usually longer than other models. When facing large-scale data, the time efficiency of the CRFs model with the traditional stand-alone processing algorithm is often unsatisfied. For example, classical CRFs model needs approximately 45 hours to train only 400k training examples (3.0GHz CPU, 1.0G memory, and 400 iterations) [14]. It is caused by the problem that the model parameter estimation cycle is long, because it needs to compute the global gradient for all features. The time and space complexities of the algorithm show non-linear growth with the growth of the training data, number of kinds of labels and the number of features. The expensive training cost is one of very important issues, which make CRFs cannot be effectively applied to the applications with massive data. To overcome this bottleneck, faster processing and optimization algorithms in parallel computing platforms have been becoming a very active research area.

For most machine learning algorithms, the high iterations needs frequent I/O operations for intermediate data which stored in disk. In current popular data processing frameworks, compared with Hadoop [15], Spark platform [16] supports a Resilient Distributed Datasets (RDD) model which is built on a memory computing framework. It allows users to store data cache in memory, and to do computation and iteration for the same data directly from the memory. Based on the computing in memory mode, Spark platform can save huge amounts of disk I/O operations time. Therefore, it is more suitable for machine learning algorithms with iterative computation compared with the traditional computation methods.

Traditional CRFs needs to consider three key steps, i.e., feature selection, parameter estimation, and model inference. For the features selection is the key problem that exert great effects on the training results, it can determine the performance of the model largely. In theory, CRFs model has a great flexibility to use a wide range of features, and the more features used in the model, the more accurate the results can be obtained in implementations [17]. However, for not all features are decisive to the results of mode inference, as the number of complicated features increases, the difficulty and importance of constructing the useful features grows.

Moreover, the parameter estimation becomes the most important reason that degrades the performance of the CRFs model, especially when the training datasets is large. In actual implementation, Limited Memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) is a popular method that has been used to do parameter estimation of CRFs [10, 18]. However, for the execution of likelihood function is a batch process, the model parameters are usually not revised until the overall training set has been scanned. Gradient descent is the main step in L-BFGS, as a specific improved method, Mini-batch gradient descent (Minibatch-GD) [19–21] can updates the parameters only after scanning part of the training data. However, for improving the performance of this algorithm to handle large-scale data efficiently, it is still a challenge to parallelize such a dependent iterative process.

In this paper, we solve such an inter-dependent problem with an efficient strategy based on the Spark platform. Most of the ways to improve the time efficiency of the CRFs model often focus on how to reduce the model parameter estimation time. However, the complexity of the model inference step increases quickly with the growth of constraint length of training datasets as well, and the model inference can be formerly performed using a modified specific belief propagation algorithm [22, 23], which is improved within the Spark platform to parallelize the model inference with a simple strategy in our works.

This paper proposes an improved conditional random fields algorithm based on the parallel computing model based on the Spark platform (SCRFs), which focuses on both the accuracy and performance for massive unstructured data. By reducing the low-

-frequency features of the model, SCRFs can avoid the overfitting of the model, which can improve the F1 value of the model prediction [24]. Moreover, SCRFs can caches the intermediate data which are used frequently into the memory to speed up the model iteration. All of these are designed to improve the effect and performance of SCRFs for the large and complicated input data. From the experimental results in the Spark environment, it has been proved that with F1 value increasing on the test data, the algorithm achieves significant speed-up ratio compared with the original CRFs algorithm implemented on Hadoop and Spark platforms. The major contributions of this paper are summarized as follows:

- We propose an efficient and fast method to generate features on the Spark platform. The features can be chosen in parallel and adjusted dynamically through excluding the unused-features to correct the model in the training procedure.
- A Parallel training and prediction mechanism of SCRFs model is proposed based on the distributed memory management architecture, which can cache the generated intermediate data into memory timely. For the iteration in training procedure is executed in memory, this way can speed up the traditional solution procedure of CRFs significantly.
- We implement SCRFs based on Spark 1.6.0 platform and evaluate its performance using some of the most common benchmarks. Experimental results verify the effectiveness and correctness of the proposed algorithms.

The rest of the paper is organized as follows: Section 2 reviews the background of traditional conditional random fields. Section 3 proposes the parallel implementation of SCRFs based on Spark RDD. Experimental results and evaluations are showed in Section 4 with respect to the accuracy and performance. Section 5 surveys related works on accelerating CRFs. Finally, Section 6 concludes the paper.

2 Background

2.1 Conditional Random Fields

Conditional Random Fields (CRFs) was first introduced by Lafferty et al. [3] in 2001 as a sequence data labeling recognition model based on statistical approaches, and it is regarded as an undirected graph model or Markov Random Field as Fig. 1.

A linear-chain CRFs defines the conditional probability of the state sequence Y for a given input sequence X, which can be formalized as (1):

$$P(y|x) = \frac{1}{Z(x)} \exp\left(\sum_{t=1}^{T}\sum_{k=1}^{K}\lambda_k f_k(y_{t-1}, y_t, x_t)\right) \quad (1)$$

where $Z(x)$ is a normalization factor, which is used to ensure that $p(y|x)$ meets the classical probability distribution $\sum_y p(y|x) = 1$:

$$Z(x) = \sum_{y} \exp\left(\sum_{t=1}^{T}\sum_{k=1}^{K}\lambda_k f_k(y_{t-1}, y_t, x_t)\right) \quad (2)$$

where $f_k(y_{t-1}, y_t, x_t)$ is a feature function which can only have a value of 1 or 0 in general cases. Many linear-chain CRFs use richer features of the input, such as the previous and the next one of current word, the identities of surrounding words, and so on. In Fig. 1, $X = \{x_1, x_2, \cdots, x_T\}$ is treated as a single large observed node on which all of the factors depend, rather than each of the $x_1, x_2, \cdots, x_T$ is treated as an individual node. In the linear-chain CRFs, each feature function can rely on the observations from any state and any point in time, and the observation argument to f_k is shown as a vector x_t, which should be understood as containing all the components of the global observations x that are needed for computing the features at time t. For example, if CRFs uses the next word x_{t+1} as a feature, then the feature vector x_t is assumed to include the identity of word x_{t+1}.

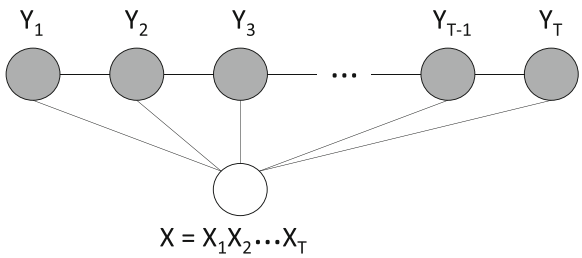


Fig. 1 Linear-chain CRFs model

The corresponding weights of feature functions $\lambda = \{\lambda_1, \lambda_2, \cdots, \lambda_T\}$ are obtained in the training procedure of the model, which is the key problem of CRFs. A higher λ_k weight means the corresponding feature appears more frequently, which is more likely to occur in the model.

$D = \{(x^i, y^i)\}_{i=1}^N$ are the examples of the training data, where each $x^i = \{x_1^i, x_2^i, \cdots, x_T^i\}$ denotes the input sequence, and $y^i = \{y_1^i, y_2^i, \cdots, y_T^i\}$ is the known output sequence. In the CRFs, parameter estimation is the procedure to find a set of appropriate parameters $\{\lambda_k\}$ to make the conditional probability $P(y|x, \lambda)$ which obtained from the model fit the training data as much as possible. For example, given an arbitrary training instance x^i, the conditional distribution $P(y|x^i, \lambda)$ of model should be as close as possible to the true output y^i from the training data.

In linear-chain CRFs, for each feature $f_k(y_{t-1}, y_t, x_t)$, the total values of f_k that occurs in the training should be as close as possible to the total values of f_k from the conditional distribution $P(y|x, \lambda)$:

$$\sum_{i=1}^{N}\sum_{t=1}^{T} f_k(y_{t-1}^i, y_t^i, x_t^i) = \sum_{i=1}^{N}\sum_{t=1}^{T}\sum_{y'y} f_k(y', y, x_t^i)\, p(y', y|x_t^i) \tag{3}$$

And the log-likelihood function of the training data D is shown below:

$$L(\lambda) = \sum_{i=1}^{N} \log P(y^i|x^i, \lambda) \tag{4}$$

From (1) and (4), the log-likelihood function can be formalized as (5):

$$\begin{aligned} L(\lambda) &= \sum_{i=1}^{N} \log\left(\frac{1}{Z(x^i)} \exp\left(\sum_{t=1}^{T}\sum_{k=1}^{K} \lambda_k f_k(y_{t-1}^i, y_t^i, x_t^i)\right)\right) \\ &= \sum_{i=1}^{N}\sum_{t=1}^{T}\sum_{k=1}^{K} \lambda_k f_k(y_{t-1}^i, y_t^i, x_t^i) - \sum_{i=1}^{N} \log Z(x^i) \end{aligned} \tag{5}$$

To find the appropriate value for each item in the parameter λ to minimize the convex function $L(\lambda)$, the partial derivative of the parameter λ_k can be calculated as (6):

$$\begin{aligned} \frac{\partial L(\lambda)}{\partial \lambda_k} &= \sum_{i=1}^{N}\sum_{t=1}^{T} f_k(y_{t-1}^i, y_t^i, x_t^i) - \frac{1}{Z(x)}\frac{\partial Z(x)}{\partial \lambda_k} \\ &= \sum_{i=1}^{N}\sum_{t=1}^{T} f_k(y_{t-1}^i, y_t^i, x_t^i) \\ &\quad - \sum_{i=1}^{N}\sum_{t=1}^{T}\sum_{y'y} f_k(y', y, x_t^i)\, p(y', y|x^i) \end{aligned} \tag{6}$$

From (3), the gradient of the log-likelihood function just equals to these constraint equations. Unfortunately, there is no closed-form solution for searching the target parameter λ by making the above equation to be zero. As a result, the parameter selection procedure needs to use some numerical iteration techniques, such as GIS (Generalized Iterative Scaling Algorithm), IIS (Improved Iterative Scaling Algorithm), L-BFGS (Limited-memory Broyden-Fletcher-Goldfarb-Shanno Algorithm) [25], and so on.

Parameter estimation requires computing the conditional distribution $P(y|x, \lambda)$. This is the task of probabilistic inference, which is a very challenging problem and is NP hard [3, 4]. In arbitrarily-structured CRFs, exactly calculating the problem is intractable, thus we would use some approximation methods such as loopy Belief Propagation (BP) [26] and Gibbs sampling [27]. There are two specific inference problems:

(1) Computing marginal distributions $p(y_s|x, \lambda)$ over a subset Y_s. The set of Y_s of the marginal distributions usually contains either of adjacent variables or a single variable. The BP algorithm can be used to solve the problem.

(2) Computing the $y^* = argmax_y p(y|x, \lambda)$. Based on this, when given a input sequence x, we can find out the most likely output sequence y.

2.2 The Model Inference in CRFs

Efficient inference is critical for CRFs, which is the key procedure during data training and label predicting on new inputs. The two inference problems are intractable for general graphs where we could only

use approximation algorithms. In the case of linear-chain CRFs, both of the above inference tasks can be accomplished efficiently and exactly by extending BP algorithm.

The inference algorithm is usually called repeatedly in overall training procedure of parameters. For its expensive computation with massive data, how to improve the efficiency is increasingly urgent. Although the marginal can just be obtained by the brute force inference, it is unable to handle anything larger than a tiny graph. The time required for force inference is proportional to the length of sequence. Let the length of sequence y be D, and y has N different labels, thus the time complexity of an brute force inference is $O(N^D)$, which is intolerable.

The structure of cliques in the linear-chain CRFs is a line, in which each of clique contains only pairs of adjacent states: (y_{t-1}, y_t). Hence, we can establish clique tree for each instance in the training data. Let $\Psi_t(C_t)$ be the t^{th} clique in the line, and C_t represents the contained variables, which contains the pairs of adjacent variables: (y_{t-1}, y_t). The value of $\Psi_t(C_t)$ is determined by an exponentiated weighted sum over the features of the clique as (7):

$$\Psi_t(y_{t-1}, y_t, x_t) = exp(\sum_{k=1}^{K} \lambda_k f_k(y_{t-1}, y_t, x_t)) \quad (7)$$

Let $\delta_{i\to j}(s_{i,j})$ be the message passed from Ψ_i to Ψ_j, and $s_{i,j}$ is the shared variables both Ψ_i and Ψ_j all contain. The value of $\delta_{i\to j}(s_{i,j})$ can be calculated by (8):

$$\delta_{i\to j}(s_{i,j}) = \sum_{C_i - S_{ij}} \Psi_i \times \prod_{k\in(N_i - j)} \delta_{k\to i} \quad (8)$$

where N_i denotes all its neighbors. After all messages have been passed, the marginal distributions of C_i can be calculated by (9):

$$p(C_t|x) = \frac{1}{Z(x)} \times \Psi_t \times \prod_{k\in(N_t)} \delta_{k\to t} \quad (9)$$

For the message passing in the clique tree is a sum-product, the pseudo code to compute the marginal distributions for sum-product belief propagation is as described in Algorithm 1.

Algorithm 1 Sum-product belief propagation

Input:

An instance of training data, parameters vector λ, model feature function $F(x, y)$.

Output:

The marginals of $\{P(y_{t-1}, y_t|x), P(y_t|x)\}_{t=1}^{T}$.

1: **for** $t \leftarrow 1$ *to* T **do**

2: $\Psi_t(y_{t-1}, y_t, x_t) \leftarrow exp(\sum_{k=1}^{K} \lambda_k f_k(y_{t-1}, y_t, x_t))$;

//Add each features f_k to its corresponding cliques and construct the initial potentials.

3: **end for**

4: $\delta_{1\to 2}(y_1) \leftarrow \sum_{y_0} \Psi_1$;

5: **for** $t \leftarrow 2$ *to* $T-1$ **do**

6: $\delta_{t\to t+1}(y_t) \leftarrow \sum_{y_{t-1}} \Psi_t \times \delta_{t-1\to t}$;

//Forward calculate the massage passed from the first clique to the last clique.

7: **end for**

8: $\delta_{T\to T-1}(y_{T-1}) \leftarrow \sum_{y_T} \Psi_T$;

9: **for** $t \leftarrow T-1$ *to* 2 **do**

10: $\delta_{t\to t-1}(y_t) \leftarrow \sum_{y_t} \Psi_t \times \delta_{t+1\to t}$;

//Backward calculate the massage passed from the last clique to the first clique.

11: **end for**

12: **for** $t \leftarrow 1$ *to* T **do**

13: $P(y_{t-1}, y_t|x) \leftarrow \frac{1}{Z(x)} \times \Psi_t \times \delta_{t-1\to t} \times \delta_{t+1\to t}$;

14: $P(y_t|x) \leftarrow \frac{1}{Z(x)} \times \delta_{t\to t+1} \times \delta_{t+1\to t}$;

//Calculate the marginal distributions.

15: **end for**

16: **return** The marginals of $\{P(y_{t-1}, y_t|x), P(y_t|x)\}_{t=1}^{T}$.

Algorithm 1 consists of four steps:

Step 1. Initialization (lines 1-3): Add each related features f_k to its corresponding cliques, and construct the initial potentials according to the (7).

Step 2. Repeatedly calculate forward propagation messages (lines 4-7): Forward calculate the massage passed from the first clique to the last clique according to (8).

Step 3. Repeatedly calculate backward propagation messages (lines 8-11): Backward calculate the massage passed from the last clique to the first clique according to (8).

Step 4. Calculate the marginal distributions according to (9) (lines 12-15).

In this model, we need to output the most likely label sequence Y corresponding to a set of input X, instead of calculating marginal probabilities over individual characters. This can be attained by using max-sum message passing [26] to calculate $y^* = argmax_y p(y|x, \lambda)$. Let $\gamma_{i\to j}(s_{i,j})$ be the max-sum message, and its value can be calculated as (10):

$$\gamma_{i\to j}(s_{i,j}) = \max_{C_i - S_{i,j}} \left(\theta_i + \sum_{k\in(N_i - j)} \gamma_{k\to i} \right) \tag{10}$$

2.3 Batch-GD Algorithm in CRFs

CRFs tends to have more parameters and more complex structure than a simple classifier, e.g., there maybe several hundred thousand parameters. It is necessary to use some regularization penalty on the parameter values to avoid the overfitting. A common choice of penalty is the L2-regularization based on a regularization parameter $1/2\sigma^2$ that determines the strength of the penalty. The regularization can also be viewed as performing maximum a posteriori (MAP) estimation of λ, and the best value of parameter σ^2 depends on specific training data. The regularized log likelihood function we seek to minimize is as (11):

$$L(\lambda) = \sum_{i=1}^{N}\sum_{t=1}^{T}\sum_{k=1}^{K} \lambda_k f_k(y_{t-1}^i, y_t^i, x_t^i) - \sum_{i=1}^{N} \log Z(x^i) - \sum_{k=1}^{K} \frac{\lambda_k^2}{2\sigma^2} \tag{11}$$

Parameters estimation in CRFs needs an iterative process with high time complexity, but some quasi-Newton methods, such as L-BFGS, are still showed significantly more efficient [3]. This method requires computing the average gradient over the entire datasets before making a single parameter update. If the training set has a very large size and contains a large number of independent and identical distributed (i. i. d) samples, then this may seem wasteful. Thus it should be more efficient to update the parameters after computing only a few examples, rather than sweeping through all of them.

Batch Gradient descent (Batch-GD) is such a simple optimization method that conforms to this insight. Although directions of the individual steps may be much better in L-BFGS, Batch-GD directions can be worked out much faster which can get reasonable results after only a few passes through the overall datasets.

With the size of mini-batch is set as m, Batch-GD algorithm consists of three steps in each iteration and the process in each iteration is as follows.

Step 1. Select m samples from the training data.

Step 2. Calculate the gradient vector ∇L as (12):

$$\nabla L = \left(\frac{\partial L(\lambda)}{\partial \lambda_1}, \frac{\partial L(\lambda)}{\partial \lambda_2}, \cdots, \frac{\partial L(\lambda)}{\partial \lambda_K} \right) \tag{12}$$

Obviously, ∇L can be obtained by calculating the partial derivative of each λ_k:

$$\frac{\partial L(\lambda)}{\partial \lambda_k} = \sum_{i=1}^{m}\sum_{t=1}^{T} f_k(y_{t-1}^i, y_t^i, x_t^i) - \sum_{i=1}^{m}\sum_{t=1}^{T}\sum_{y'y} f_k(y', y, x_t^i) p(y', y|x^i) - \frac{\lambda_k}{\sigma^2} \tag{13}$$

Step 3. Calculate the step size a_t in the t^{th} iteration as (14):

$$a_t = \frac{a}{\sqrt{t}} \tag{14}$$

where the input parameter a is the initial step-size. Note that selecting the best step-size for Batch-GD methods can often be delicate in practice, which is needed to update the estimated parameters of the λ in each iterative step.

3 The Parallel CRFs Model Based on Spark

Apache Spark has an advanced DAG execution engine that supports cyclic data flow and in-memory computing, which can run programs up to 100× faster than Hadoop MapReduce in memory, or 10× on disk [16]. Based on the parallel computing and distributed memory management mechanism in Spark platform, the performance of the SCRFs algorithm can be improved through three optimized parallel processes: features generation, parameters training, and the prediction process.

3.1 Parallel Feature Generation

Traditional procedure of features generation in CRFs model is usually in a serial way. Because the features based on each instance are constructed in order,

it will cause a considerable waste of computation for some useless features. In order to improve efficiency, SCRFs proposed a new way to generate all features which can be selected in parallel based on the Spark platform.

For CRFs model is log-linear, one way to improve the precision and complexity is to introduce more features. Based on the identity of words in natural language texts, there can easily be thousands of simple features, and ten or more complicated features would result in several million features. Large amounts of features consume large resources of memory and computation. However, most of these features are not be useful, and they may occur only once in the training data, or even never occur. For example, the feature "word x_t is from and label y_t is person" should never occur in the training data, and it is impossible to get a value of 1. These features are usually classified into unused-features. But when they are assigned a negative value, the possibility of incorrect label can be decreased. The more negative the assigned value is, the less likely the label is. (The training procedure of the parameters does in fact assign negative values to these features). In general, unused-features can slightly improve the accuracy of the model at the expense of greatly increasing the complexity.

The model still can make correct decisions without these unused-features. For example, the feature "word x_t is from and label y_t is person" is impossible to be useful, while "*from*" is a common word and will be assigned high probability to other (not a person) label. To reduce the complexity of the model and ensure its accuracy, only these unused-features which will likely correct the model can be included into the model. In many tasks, even in the early training procedure, many of properties are correctly labeled. For example, for the word "*from*" occurs with high frequency, it will not be labeled as person. The model can learn this so quick that there is no need to include this feature "word x_t is from and label y_t is person".

Only these unused-features which contribute to the model will be included. In the phase of feature generation, all features will be generated but only these ones appears in will be added to the model at first. But as the model entering into the training procedure, if some used-features will do good to the model, these features will also be checked and included.

First of all, the collected training data in a particular format should be loaded into the Spark platform. The big data should be stored as a type of RDD object in the Spark platform.

$$RDD(TrainData) = SparkContext.textFile(TrainData);$$

Each instance of $RDD_{original}$ contains the input sequence $X = \{x_1, x_2, \cdots, x_i, \cdots, x_n\}$ and corresponding output sequence $Y = \{y_1, y_2, \cdots, y_j, \cdots, y_m\}$. Because the feature template depends on x_t, in order to generate all features, we need to scan all of the training data. Without a parallel processing framework based on computing cluster, this procedure will be time-consuming in serial way. The algorithm of the parallel generating features procedure of SCRFs Model is shown in Algorithm 2.

Algorithm 2 Parallel feature generation of SCRFs

Input:

The RDDs of train data, the RDDs of feature template.

Output:

Parameters vector λ, unused-features unf, the RDDs of features.

1: $RDD(feature) \leftarrow RDD(TrainData).flatMap(featureTemp)$;
//Transform the training data into the features according to the feature template.
2: $RDD(feature, totalcount) \leftarrow RDD(feature).reduceByKey$;
//Combine the same features with the same key and accumulate their values.
3: $unf \leftarrow RDD(feature, totalcount)$;
//Obtain the unused-features.
4: $\lambda \leftarrow RDD(feature)$;
//Update the parameters vector λ.
5: $RDD(feature).saveAsTextFile$;
//Save the RDDs of features.
6: **return** parameters vector λ, unused-features unf, the RDDs of features.

In the transformation stage, the x_t^i in each training instance is converted to some key-value pairs. The keys contains all features generated by x_t^i according to the feature template, and corresponding value in the Map can only get a value of 1 or 0 representing the presence or absence of this feature.

In the Reduce stage, the same features will be combined and their value will be accumulated. The result will be used to initialize parameters vector, which is a sparse vector to reduce computational cost. The features with total counts equal to zero will be set to the unused-features while the others will be set to the used-features. The unused-features will not be added to the model naturally, but their parameters may be updated during the training procedure.

3.2 The Transformation & Caching for the Train Data

For the inference algorithm are called frequently in the training procedure, the training data need to be transformed into corresponding features many times. This transforming procedure can also be implemented in parallel on the Spark platform to improve the performance. Based on the distributed memory management mechanism of the Spark platform, we can cache the intermediate results into the memory on the Spark platform, which supports frequently reuse in the next procedure.

Specifically, the x_t^i in each training instance will be converted to a vector which contains the indexes of all features generated by x_t^i in the $RDD(feature)$. In (15), we assume $RDD(feature)$ has K features, and $FI(x_t^i)$ represents the index feature vector of x_t^i:

$$FI(x_t^i) = \{k \mid f_k(y', y, x_t^i) = 1\} \tag{15}$$

In SCRFs, $(y_{t-1}^i, y_t^i, x_t^i)$ in each training instance is converted to a scalar, which means the index of feature $f(y_{t-1}^i, y_t^i, x_t^i)$ in the $RDD(feature)$. And the scalar can be formalized as $EI(y_{t-1}^i, y_t^i, x_t^i)$ as (16):

$$EI(y_{t-1}^i, y_t^i, x_t^i) = \{k \mid f_k(y_{t-1}^i, y_t^i, x_t^i) = 1\} \tag{16}$$

In the implementation of SCRFs, the intermediate results will be cached into the memory on the Spark framework instead of being saved on the HDFS, which will be directly reused in the next iteration of the training procedure. The specified steps of the parallel transforming for training data are shown in Algorithm 3. In this way, each training instance is transformed into the feature vector and the empirical feature factor, and then these intermediate results will be cached into memory to improve the efficiency of data I/O.

Algorithm 3 Parallel transforming of training data & data caching

Input:
The RDDs of train data, the RDDs of features.
Output:
The RDDs of middle result.

1: $RDD(FI) \leftarrow < RDD(X, Y), RDD(feature) >$;
//Transform each training instance into the feature vector.
2: $RDD(EI) \leftarrow < RDD(X, Y), RDD(feature) >$;
//Transform each training instance into the empirical feature factor.
3: $RDD(FI).cache$;
4: $RDD(EI).cache$;
//Cache the FI, EI into memory.
5: **return** The RDDs of middle result.

3.3 Parallel Training Procedure

The parameter estimation is the crucial component of model training. It is clear that there will certainly be considerable time consumption for data training and machine learning for large-scale training data. Especially with a large number of i. i. d samples, the traditional ways of batch training will be very time consuming. Equation (17) shows the calculating process for parameter λ:

$$\frac{\partial L(\lambda)}{\partial \lambda_k} = \sum_{i=1}^{m} \left(\sum_{t=1}^{T} f_k(y_{t-1}^i, y_t^i, x_t^i) - \sum_{t=1}^{T} \sum_{y'y} f_k(y', y, x_t^i) p(y', y|x^i) \right) - \frac{\lambda_k}{\sigma^2} \tag{17}$$

Note that the first part is the expectation of feature values f_k under the empirical distribution with a given instance, which can be described as (18).

$$E_{iD}[f_k] = \sum_{t=1}^{T} f_k(y_{t-1}^i, y_t^i, x_t^i) \tag{18}$$

The second part is the expectation of feature values f_k, which can be described as (19):

$$E_{i\lambda}[f_k] = \sum_{t=1}^{T} \sum_{y'y} f_k(y', y, x_t^i) p(y', y|x^i) \tag{19}$$

After simple replacements, Equation (17) can be written as:

$$\frac{\partial L(\lambda)}{\partial \lambda_k} = \sum_{i=1}^{m} (E_{iD}[f_k] - E_{i\lambda}[f_k]) - \frac{\lambda_k}{\sigma^2} \tag{20}$$

In the implementation of SCRFs based on Spark platform, $E_{iD}[f_k]$ can be directly obtained by $RDD(EI)$. However, $E_{i\lambda}[f_k]$ need inference algorithm to calculate the marginals. $RDD(FI)$ are used to establish the clique tree, and unused-features among them are filtered out and will not be added to the cliques to reduce computational cost. After same iterations while the model has not been fully trained, these unused-features which may correct the mistake of current model will be changed to used-features and will be trained in the rest training procedure.

When computing the marginal distributions, if the probability of an unused-feature is larger than a certain value which means the model may make mistakes without this feature, hence the feature would be included and be trained to modify the model. he steps of the parallel training procedure of SCRFs model is shown in Algorithm 4, we can easily work out that the time complexity of the parallel training procedure is $O(G \times m \times L \times D^2)$, where G is the iteration times, m is the number of training samples, L is the length of sentence, D is the number of labels.

Algorithm 4 Parallel training procedure

Input:

$RDD(EI, featureFactor)$, $RDD(FI)$,unused-features unf.

Output:

Parameters vector: λ.

1: $\lambda \leftarrow (0, \cdots, 0)$;
2: $t \leftarrow 0$;
3: $RDD(EI, FI) \leftarrow < RDD(EI), RDD(FI) >$;
4: **repeat**
5: $broadcast(\lambda, unf)$;
//Broadcast the λ and $unf$to all nodes after each update.
6: $RDD_{msize}(EI, FI) \leftarrow RDD(EI, FI).sample$;
//Extract m samples.
7: $RDD(E_{iD}[f_1], E_{iD}[f_2], \cdots, E_{iD}[f_K]) \leftarrow RDD_{msize}(EI, FI)$;
8: $(E_D[f_1], E_D[f_2], \cdots, E_D[f_K]) \leftarrow RDD(E_{iD}[f_1], E_{iD}[f_2], \cdots, E_{iD}[f_K]).reduce$;
//Calculate the empirical feature vector.
9: $RDD(UFI) \leftarrow RDD_{msize}(EI, FI).map.fliter$;
//Calculate the used-features.
10: $RDD(E_{i\lambda}[f_1], E_{i\lambda}[f_2], \cdots, E_{i\lambda}[f_K]) \leftarrow RDD(UFI).map$;
11: $(E_\lambda[f_1], E_\lambda[f_2], \cdots, E_\lambda[f_K]) \leftarrow RDD(E_{i\lambda}[f_1], E_{i\lambda}[f_2], \cdots, E_{i\lambda}[f_K]).reduce$;
//Calculate the expectation of feature values.
12: $\nabla L_t = (E_D[f_1], E_D[f_2], \cdots, E_D[f_K]) - (E_\lambda[f_1], E_\lambda[f_2], \cdots, E_\lambda[f_K]) - \lambda_k/\sigma^2$;
//Compute the gradient.
13: $\lambda \leftarrow \lambda + a_t \cdot \nabla L_t$;
//Update the gradient.
14: **for** $(unesdfeatures)_i$ in $RDD(FI)$ **do**
15: **if** $(P(unesdfeatures_i) > \varepsilon)$ **then**
16: $unf - unesdfeatures_i$;
17: **end if**
18: **end for**
//Detect the potential unused-features.
19: $t \leftarrow t + 1$;
20: **until** convergence
21: **return** λ.

3.4 Parallel Prediction Procedure

In most model implementations of machine learning, the obtained trained parameters of model are used to predict test data, and each test data will be inferred and predicted by serial. In order to improve the prediction speed of SCRFs with massive data, this paper proposes a parallel prediction based on the Spark framework. We adopt max-sum belief propagation algorithm to implement the prediction procedure for single instance, and the pseudo code is described as Algorithm 5.

In order to prevent underflow, Algorithm 5 performs computations in the logarithmic domain. The aim is to output the most probable instantiation of the variables in the network instead of solving the marginal distributions for each of them. Note that this algorithm is nearly identical to Algorithm 1, with the sum replaced by max and the products replaced by sum in the definitions.

Algorithm 5 Max-sum belief propagation

Input:

An observation sequence $X = \{x_1, x_2, \cdots, x_T\}$, parameters vector λ, model feature function $F(x, y)$.

Output:

The most likely label sequence $Y = \{y_1, y_2, \cdots, y_T\}$.

1: **for** $t \leftarrow 1$ *to* T **do**
2: $\theta_t(y_{t-1}, y_t, x_t) \leftarrow \sum_{k=1}^{K} \lambda_k f_k(y_{t-1}, y_t, x_t))$;
3: //Initialise the value for clique tree.
4: **end for**
5: $\gamma_{1\to 2}(y_1) \leftarrow \max_{y_0} \theta_1$;
6: **for** $t \leftarrow 2$ *to* $T-1$ **do**
7: $\gamma_{t\to t+1}(y_t) \leftarrow \max_{y_{t-1}}(\theta_t + \gamma_{t-1\to t})$;
//Calculate the value for forward propagation of messages.
8: **end for**
9: $\gamma_{T\to T-1}(y_{T-1}) \leftarrow \max_{y_T} \theta_T$;
10: **for** $t \leftarrow T-1$ *to* 2 **do**
11: $\gamma_{t\to t-1}(y_t) \leftarrow \max_{y_t}(\theta_t + \gamma_{t+1\to t})$;
//Calculate the value for backward propagation of messages.
12: **end for**
13: **for** $t \leftarrow 1$ *to* T **do**
14: $argmax\, P(y_{t-1}, y_t|x) \leftarrow \max(\theta_t + \gamma_{t-1\to t} + \gamma_{t+1\to t})$;
15: $argmax\, P(y_t|x) \leftarrow \max(\gamma_{t\to t+1} + \gamma_{t+1\to t})$;
//Calculate the marginal distributions.
16: **end forreturn** the most likely label sequence $Y = \{y_1, y_2, \cdots, y_T\}$.

Algorithm 6 Parallel prediction procedure of SCRFs

Input:

λ, test data, unused-features: unf.

Ouput

The output sequence.

1: $RDD(Testdata) \leftarrow SparkContext.textFile(Testdata)$;
//Load the test data as RDD.
2: $RDD(UFI) \leftarrow RDD(Testdata).map$;
//Transform the test data into the used-features.
3: $RDD(Clique) \leftarrow RDD(UFI).map$;
//Transform the used-feature into the clique tree.
4: $RDD(output) \leftarrow RDD(Clique).map$;
//Calculate the output by using the Max-sum Belief Propagation Algorithm.
5: $RDD(output).saveAsTextFile$;
//Save the output on HDFS.
6: **return** The output sequence.

Based on Algorithm 5, Algorithm 6 shows the implementation steps of the parallel predict algorithm. First, we need to load the test datasets into the Spark as the type of RDD objects while the global variables are needed to be broadcasted to all worker nodes. Then transform the test data into the used-features and further transform it into the clique tree. Finally, calculate the output by using the max-sum belief propagation algorithm. The number of partitions in the RDD determines the concurrent tasks number, and each map function optimizes a partition in parallel. In the case of the predict algorithm, the output of each map function is a partial state sequence for the local partition. In this way, without combined output, the reduce stage is unnecessary.

4 Evaluation

In this section, SCRFs is validated under two natural language processing tasks: *Named Entity Recognition* and *Chinese Word Segmentation*. In order to evaluate the performance in various conditions, the experimental codes are run on different numbers of computing nodes with various sizes of datasets.

4.1 Experimental Settings

SCRFs has been evaluated on a practical test cluster, which includes 6 slave nodes and 1 master node

Table 1 The software and hardware configurations in the Spark cluster

The node type	Master	Slave
Software environment	Ubuntu 12.04, JDK 1.7, Hadoop 2.6.0, Spark 1.6.0	Ubuntu 12.04, JDK 1.7, Hadoop 2.6.0, Spark 1.6.0
CPU	4 cores, 2.7 GHz	4 cores, 2.7 GHz
Memory	8G	8G
Quantity	1	6

connected by 1 Gb Ethernet switch. The experimental environment is based on Hadoop 2. 6. 0 and Spark 1. 6. 0. The hardware and software configurations are shown in Table 1. All experiments use the default configurations in Hadoop and Spark for HDFS.

4.2 The Benchmark of Named Entity Recognition

Named Entity Recognition is a critical task for automatically mining knowledge from biological literature [28], we implement this benchmark based on JNLPBA 2004 datasets [29], and the training set is GENIA corpus v.3.02 [30], which consists of 2000 abstracts that are searched out by MEDLINE database with using MeSH terms human, blood cells and transcription factors as the keywords. 404 abstracts of the test set are also from MEDLINE database. Half of them are obtained by the same way as the training set, and the other half are searched out by using MeSH terms blood cells and transcription factors as the keywords. Tables 2 and 3 shows the basic and category information about JNLPBA 2004 datasets.

In our experiments, we only use the five classes: protein, DNA, RNA, cell_line, and cell_type. In these experiments, we increase the datasets to different sizes by transforming or copying in a random sampling way to see the accuracies and performance of this algorithm.

4.2.1 Accuracy Evaluation

In this section, the indicators recall, precision, and F1 values are used to evaluate the performance and effectiveness of SCRFs, by comparing them with the following algorithms and their implementations:

CRFs: the implementation of the basic original conditional random fields algorithm in single machine;

SOCRFs: the implementation of original conditional random fields algorithm in Spark platform;

MRCRF: the basic original conditional random fields algorithm based on ordinary MapReduce implementation in Hadoop platform [31];

Recall (also known as sensitivity) is defined as (21), which is the fraction of relevant instances that are retrieved:

$$Recall = \frac{N_{correct\_c}}{N_{C\_all}} \tag{21}$$

where $N_{correct\_c}$ denotes the number of identified named entities which are correctly classified into class C, and N_{C_all} represents the number of all identified named entities in class C.

Precision is the fraction of retrieved instances that are relevant, which is defined as (22):

$$Precision = \frac{N_{correct\_c}}{N_{C\_reality}} \tag{22}$$

where $N_{C\_reality}$ denotes the number of identified named entities which are actually classified into class C.

F_1 value takes the accuracy and precision into account comprehensively, which can be calculated as (23):

$$F_1 = \frac{2 \times Recall \times Precision}{Recall + Precision} \tag{23}$$

Table 2 JNLPBA 2004 datasets basic information

Datasets	Num of abstracts	Num of sentences	Num of words
Training set	2000	18546	472006
Test set	404	3856	96780

Table 3 JNLPBA 2004 datasets category information

Datasets	Protein	DNA	RNA	Cell type	Cell line
Training set	30269	9533	951	6718	3830
Test set	5067	1056	118	1921	500

Table 4 SCRFs of Biomedical named entity recognition

Features	Class	Precision	Recall	F1 Score
charngram 1:6+ en-01-18	Protein DNA	74.43 71.34	80.64 77.47	77.41 74.28
	RNA Cell_line	62.33 59.83	83.68 71.03	71.44 64.95
	Cell_type All	73.24 69.77	78.92 82.86	75.97 75.75

Table 4 summarizes the results of SCRFs of named entity recognition:

We tried to choose more common features to constitute a combination of features in this experiments. Figure 2 illustrates the comparisons of the precision, recall, and F_1 value between our algorithm and the other algorithms on the JNLPBA 2004. The results show that MRCRF and CRFs are approximately the same due to the inherent similarity of the two implementations. As a matter of fact, these three indicators are only depend on the specific algorithm processes, and no matter whether they are running on a parallel computing environment or not. Hence, in Fig. 2, the experimental results of CRFs and SOCRFs are the same for these indicators. For the sake of simplicity, we do not show the squares of SOCRFs in Fig. 2, and the same reason can be used to explain the results in Fig. 11.

Owing to features selection, the indicators precision of SCRFs is a little bit lower than the other two algorithms, and this is because that in the training procedure of SCRFs, it requires checking and excluding the unused-features dynamically. For the above step is inexistent in the other algorithms, the precision of SCRFs would decrease slightly. And for the mechanism of used-features selection, the SCRFs algorithm can prevent over-fitting for the model, and it also can increase the recall and F1 value compared with other algorithms.

4.2.2 Running Time with Different Data Sizes

Figure 3 represents the variation of job execution time when the data size ranges among {5MB, 10MB, 20MB, 30MB}. It is obvious that the processing time increases progressively with the data size. There is no much difference between the run time of the CRFs and the MRCRF, which is mainly due to the communication procedure. Although the iterative calculation in Spark framework can be optimized based on memory, the SCRFs algorithm performs badly on the small data size. This is principally because SCRFs uses the Mini-batch GD method to train the model parameters. Mini-batch GD method is not useful in relational settings in which the training data are not independent identically distributed, or on small datasets. SCRFs is a little bit better than the SOCRFs. This is because there are not many features reduced by the SCRFs on

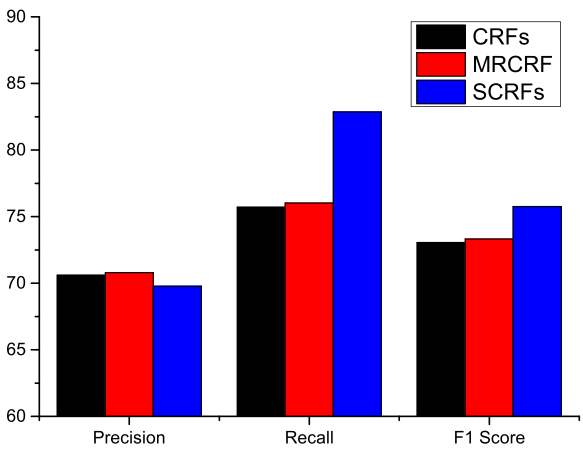


Fig. 2 The precision, recall, and F1 of different algorithm

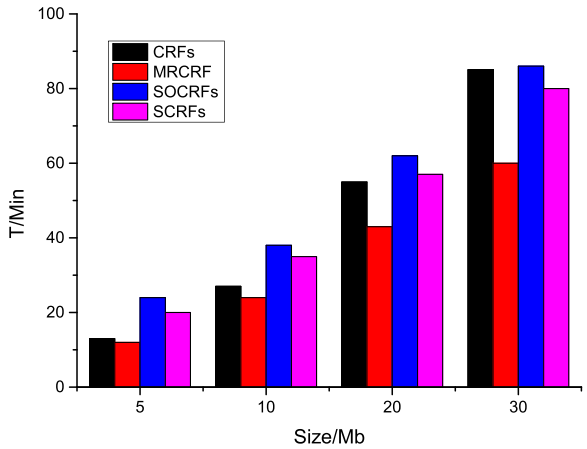


Fig. 3 Running time on small training data

the small data, and in the training procedure of SCRFs, it requires checking and excluding the unused-features dynamically.

Figure 4 reflects the algorithm performance with large training data, which ranges among {1GB, 2GB, 5GB, 10GB}. The time of the sequential method keeps growing, and the extent of growth is more and more significant. When the data size reaches 5GB, the time of sequential method is so huge that we no longer record. There is significant difference between the run times of the CRF method and the MRCRF method due to the parallel method based on the Hadoop platform. The time of the SCRFs method has modest growth and outperform the MRCRF and SOCRFs method especially for large-scale samples due to the transforming, caching the intermediate results and the used features.

4.2.3 Speedup with Different Data Sizes

In order to better demonstrate the performance benefit of the SCRFs algorithm over the other algorithms, we calculate the parallel speedup and make comparison between them. The Speedup is defined as the ratio of the sequential processing time to the parallel processing time, which is measured in the cluster environment with 7 computing nodes.

Figure 5 shows that the speedup of SCRFs on particularly small data is even lower than 1. On one hand, the reasons that Spark platform needs to start a driver program for each job, includes schedulers and some resources manager modules, etc. And the time consumptions of these processes would high-proportioned with relatively small datasets. On the other hand, for the execution time of jobs, the time of map phase decrease because of the smaller input data, but the time of reduce phase would increase because that the iterations is raised markedly even though the time for each iteration decreases. Hence, we can know that the Batch-GD algorithm in CRFs on Spark platform is not good enough to handle the small training data. From the experimental results, we can observe that the speedup effect only raise when the input data sizes exceed 30MB, which means that the traditional resolution procedure would get the better effect with the small size of datasets on the contrary.

From Fig. 6, it can be seen that when facing a large amount of data, SCRFs can bring a better speedup. The performance of SCRFs on the Spark platform is obviously better than SOCRFs and MRCRF on the Hadoop platform. On large datasets, Mini-batch GD

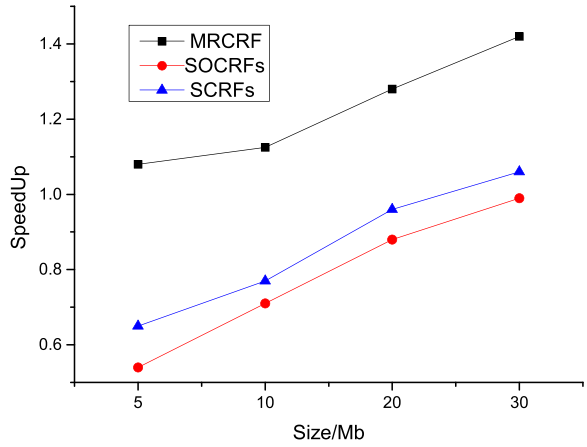


Fig. 5 Speed-up on small training data

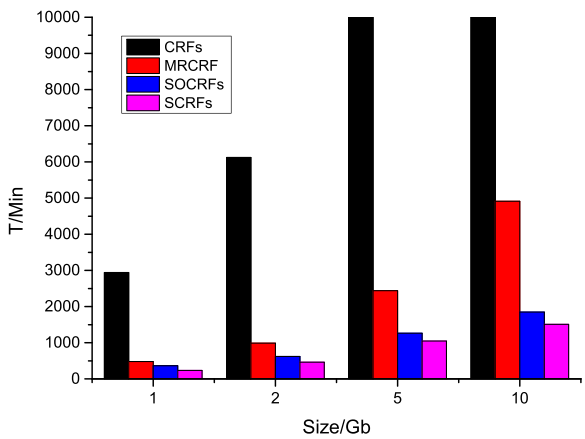


Fig. 4 Running time on large training data

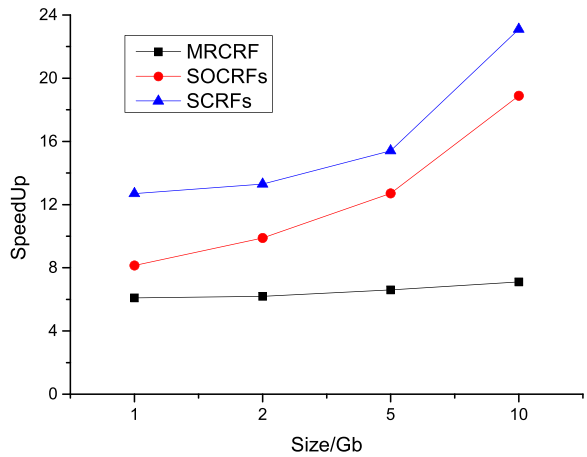


Fig. 6 Speed-up on large training data

method can offer considerable speedups and significantly reduce the training time. And in specific SCRFs implementation, the features is generated and reduced in parallel and the frequently used intermediate data is cached to memory. The overhead involved in transferring large parameter vectors across the network is decreased by using sparse vectors on the Spark platform. All these improvements make our SCRFs algorithm performs better.

4.2.4 Running Time with Different Node Scales

For the experiments in this section, the data size are set as 30M for the small training data and 10G for the large training data. Figure 7 shows the changes in the job execution time under different numbers of slave nodes with small training data. From these results, we can find that the descending trend of execution time is increasingly slow with the nodes increase. This is because the Spark computing framework is based on memory computing. In our cluster environment, each node has 8G memory. When a few computing nodes can load all datasets and finish the iterative calculation in memory, the growth of number of nodes merely increase some communication consumptions among the computing cluster.

Compared to Fig. 7, Fig. 8 illustrates the algorithm running results with large training data. It is obvious that the downward trends of execution time with the increasing nodes are faster than using small training data. This is because few data nodes cannot hold all input data, and it would be loaded into all nodes, the scale effect of the computing cluster is more obvious than the above group of experiments. Because of more features used in SOCRFs than SCRFs, the gap between the execution time of SCRFs and the SOCRFs reduce with the nodes increase.

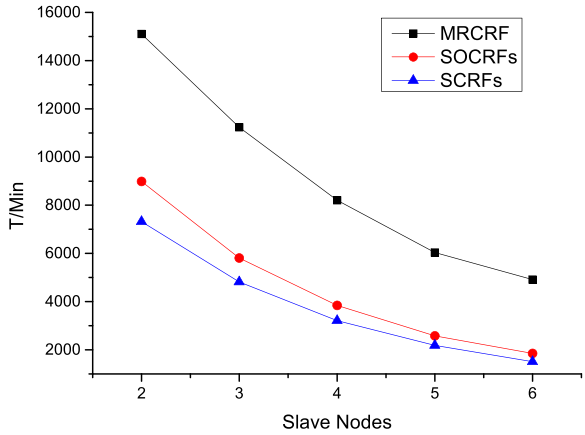


Fig. 8 Running time with different node scales on large training data

Figures 9 and 10 show the comparison of Speed-up ratio among MRCRF, SOCRFs and SCRFs respectively. Because when dealing with a small set of data, the iterative process in the Minibatch-GD would cost more resources and produce more computing time than the LBFGS algorithm, which used in MRCRF. Moreover, because the generated intermediate data are cached into memory timely, SCRFs can speed the iterative calculation effectively. As shown in Fig. 10, the performance of SCRFs is better than MRCRF

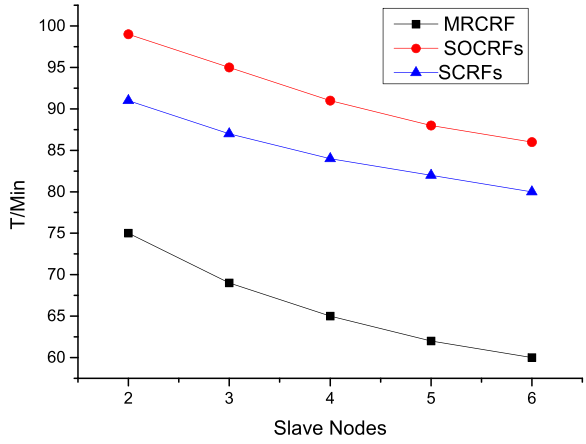


Fig. 7 Running time with different node scales on small training data

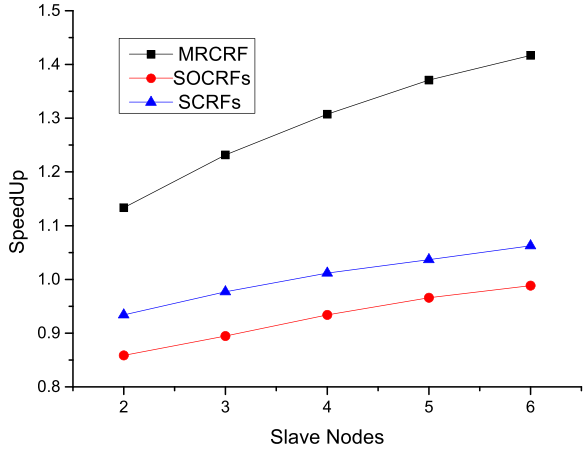


Fig. 9 Speed-up with different node scales on small training data

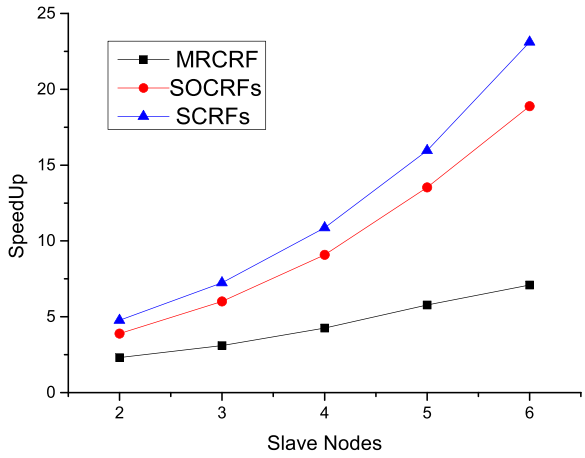


Fig. 10 Speed-up with different node scales on large training data

for Minibatch-GD algorithm which used in parameter estimation usually outperform LBFGS algorithm with large training datasets.

4.3 The Benchmark of Chinese Word Segmentation

Chinese word segmentation is a fundamental issue in the field of natural language processing [32], we experiment this benchmark based on the second International Chinese Word Segmentation Bakeoff datasets [33], which contains three corpora from different source: Microsoft Research Asia (MSR), City University of Hong Kong (CU), and Peking University (PKU). Table 5 shows the detail properties of the three datasets. For simplicity's sake, classical unigram (C_0, C_1, C_{-1}) and bigram ($C_{-1}C_0$, C_0C_1,$C_{-1}C_1$) are used as feature templates in this experiments.

4.3.1 Accuracy Evaluation

Figures 11, 12 and 13 show the comparison results about the performance of algorithms CRFs, MRCRF, and SCRFs. The results are based on convergence of batch training. In our expectation, the performance of SCRFs is consistent on the three different datasets. In Fig. 11, the accuracy effects of MRCRF and CRFs are basically in line, although the precision of SCRFs algorithm is a little bit lower than the two algorithms, a higher F_1 value is obtained. The reasons for the performance improvements are given in Subsection 4.2.1, we can see that CRFs receive relatively high precision, recall, and F1 on those three different datasets.

Table 5 Properties of SIGHAN bakeoff 2005 datasets

Data source	Word Type	Word	Char type	Char
MSR	8.8×10^4	6.9×10^4	5.5×10^4	2.4×10^6
CU	1.5×10^6	1.1×10^6	5×10^3	5×10^3
PKU	5×10^3	4.1×10^6	2.4×10^6	1.8×10^6

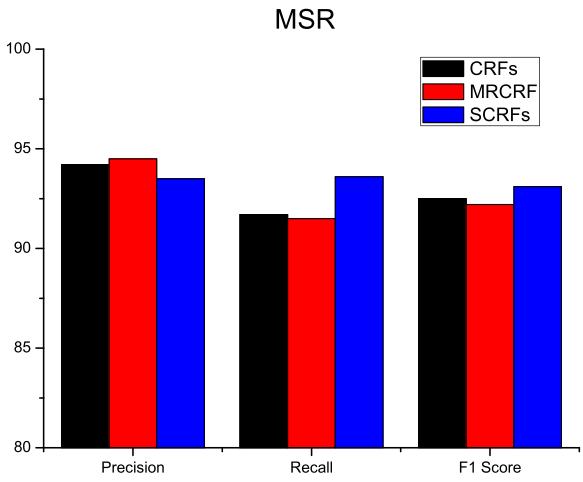


Fig. 11 The precision, recall, and F1 of different algorithms on MSR datasets

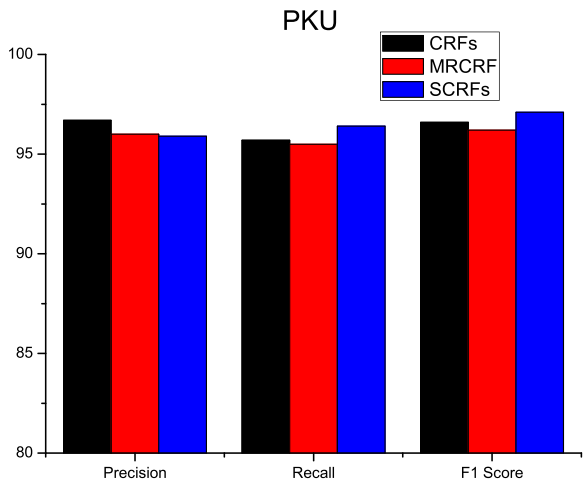


Fig. 13 The precision, recall, and F1 of different algorithms on PKU datasets

For simplicity, we don't show the squares of SOCRFs in figures because it's the same as CRFs.

4.3.2 Accelerated Training

Here, we perform the experiments on optimizing functions of different algorithms on the three different datasets. The experiments mainly compares the convergence speed of different training methods in different algorithms, such as L-BFGS, we used the Batch-GD algorithm in SCRFs.

The curves of the objective functions by varying training iterations are shown in Figs. 14, 15 and 16. As we can see that in all cases compared with the other training methods in different algorithms, the SCRFs algorithm achieved the same level of objective values on convergence. Most importantly, the SCRFs algorithm converges much faster than the all other algorithms. That is because the training procedure is accelerated by task parallelism on Spark platform and the in-memory computation for the calculation and iteration on training data. These reasons can also explain the reduction of the training time of SOCRFs. In Fig. 14, the objective functions under SCRFs and SOCRFs algorithm converge dramatically during the beginning of a short period of time, and continue converge slightly until ending. However, the convergence

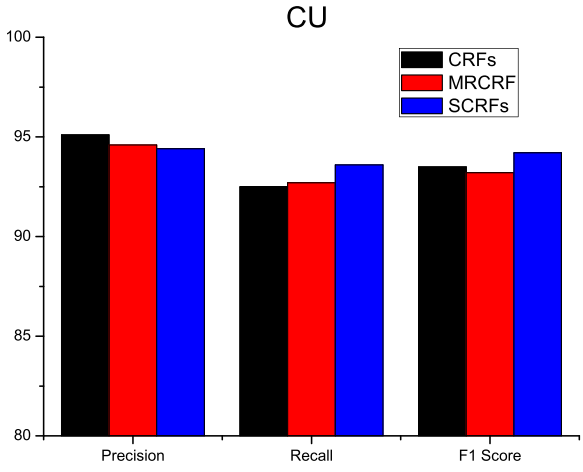


Fig. 12 The precision, recall, and F1 of different algorithms on CU datasets

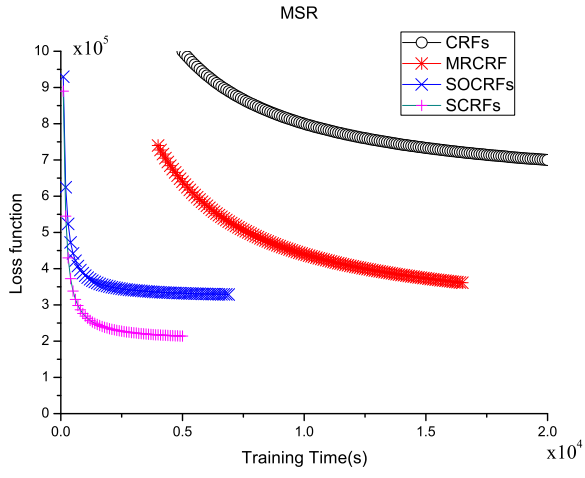


Fig. 14 Convergence speed of different training algorithms on MSR datasets

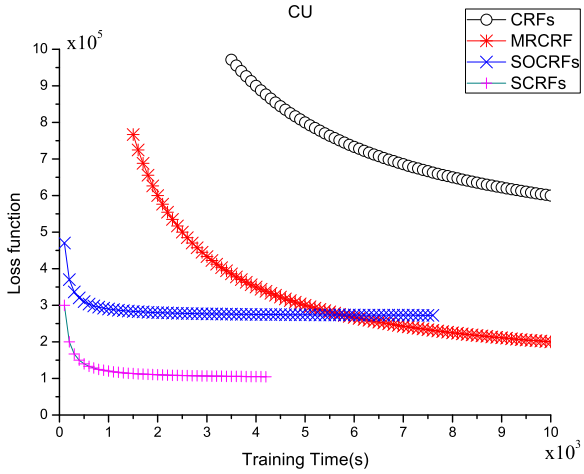


Fig. 15 Convergence speed of different training algorithms on CU datasets

process under CRFs and MRCRF algorithm are very slow and gentle, the training time of CRFs even more than 5 hours, while SCRFs and SOCRFs are less than 1.6 hours. Obviously, the training procedure of SCRFs can be accelerated significantly because of the optimization for the Batch-GD training method.

In Figs. 15, and 16, we can see that the reducing of training time of SCRFs and SOCRFs are unobvious compared to in Fig. 14, that is because the accelerated effect of SCRFs becoming more worse as the decrease of training data size, and we illustrated the reasons in detail in Subsection 4.2.3. Finally, we can take the conclusion that SCRFs algorithm can speed

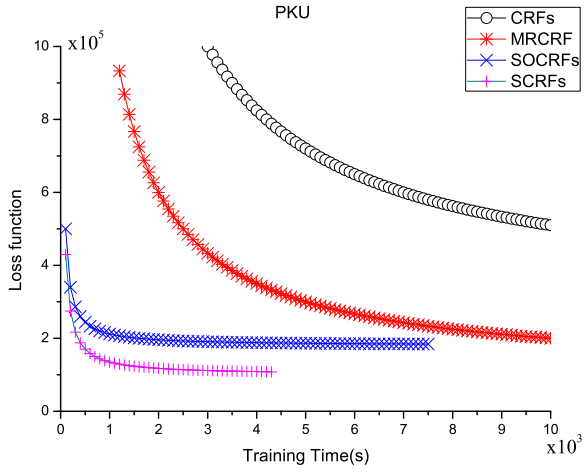


Fig. 16 Convergence speed of different training algorithms on PKU datasets

up the rate of iterations greatly, thus decrease the training time significantly, especially on large-scale data. At the same time, a high level of precision, recall and F1 score are also can obtained.

5 Related Works

The MapReduce framework has become the de facto standard for big data processing due to its attractive features and abilities [34]. As an open source implementation of MapReduce, Hadoop has been developed as a solution for performing large-scale data-parallel applications in Cloud computing, which is widely used in data mining, especially for social media, text and unstructured data (such as video and image) [35]. Apache Mahout [36] has developed many parallel algorithms in the field of machine learning based on MapReduce programming model [37]. Moreover, some machine learning algorithms based on MapReduce model are proposed [38–40], and their performance are verified on Hadoop platform. However, academia and industry have started to recognize the limitations of the Hadoop framework in several application domains and big data processing scenarios [41]. Since MapReduce is designed as a shared-nothing architecture such that each job is isolated from each other, and all jobs can only interact through the HDFS. Indeed, almost all of the machine learning algorithms need to do some iteration operations. For those machine learning algorithms based on the MapReduce, the intermediate results between each iteration operations will be stored to the HDFS, and it will cost a large amount of the disk I/O operations time.

Apache Spark [16] is another excellent cloud platform based on parallel computing and cluster computing, which is suitable for data mining and machine learning. Comprising with the MapReduce of Hadoop which based on the HDFS framework, Spark platform supports the Resilient Distributed Datasets (RDD) algorithm model based on the memory computing framework. It allows storing a data cache in memory, and doing the computation and iteration for the same data directly from memory. Spark platform saves large amounts of disk IO operations time. Therefore, Spark is more suitable for the machine learning and data mining that has more iterative computation.

There has been some excellent research works proposed for accelerating CRFs. Pal et al. proposed a

Sparse Forward Backward (SFB) algorithm, in which marginal distribution is compressed by approximating the true marginal using Kullback-Leibler (KL) divergence [42]. Cohn proposed a Tied Potential (TP) algorithm which constrains the labeling considered in each feature function, such that the functions can only detect a relatively small set of labels [43]. Both of these techniques efficiently compute the marginal with significantly reduced runtime, resulting in faster training and decoding of CRFs. However, these methods could reduce computational time significantly, which trained CRFs only on a small datasets.

In order to handle large data, Jeong et al. proposed an efficient inference algorithm of CRFs for large-scale natural language data which unified the SFB and TP approaches [44]. Lavergne et al. addressed the issue of training large CRFs, containing up to hundreds output labels and several billion features. Efficiency stems here from the sparsity induced by the use of penalty term [45]. However, none of these works described so far explore the idea of speeding up CRFs in multiple processors and thus their performance is limited by the resources of a single computing node.

Data analysis and mining based on machine learning technologies has been becoming one of hot fields. The researches in distributed and parallel data mining based on cloud computing platform have achieved a lot of outstanding achievements. There are three effective parallel implementations for CRFs currently. In [46], a novel distributed training method of CRFs based on Message Passing Interface (MPI) improved the time performance on large datasets. In [47], an efficient parallel inference on structured data with CRFs based on GPU has been proposed, and it has been testified that the approach is both practical and economical on large datasets. However, for the communication cost caused by this model is usually higher, they are not suitable for a distributed cloud environment. Li and Tang et al. [31, 48] improved the original CRFs based on the MapReduce framework and reduce the training time greatly for large-scale training corpus. Due to the bandwidth and disk I/O bottleneck, they are not suitable for an iterative machine learning algorithm. In the proposed SCRFs, we overcome these limitations by a parallel implementation of CRFs based on Spark, which is suitable for huge datasets.

6 Conclusion

This paper proposed an improved SCRFs model based on Spark RDD for massive text data. We optimize the conditional random fields algorithm by two ways: 1) Features reduction: an efficient and fast method to generate features in parallel. The features can be adjusted dynamically through excluding the unused-features to correct the model in the training procedure. And 2) Intermediate data caching: a parallel training and prediction of SCRFs model based on the distributed memory management mechanism, which can make the generated intermediate data be cached into memory timely. In this way, the iterations in the training procedure are also executed in memory. Moreover, in the implementation of SCRFs algorithm on the Spark framework, we parallelize the model in three phases: the generating features process, the training procedure and the prediction process. The experimental results indicate that the SCRFs algorithm has obvious advantages compared with the original conditional random fields algorithm and other improved conditional random fields algorithms in terms of the accuracy and performance in model training.

Acknowledgements The work is supported by the National Natural Science Foundation of China (Grant Nos. 61572176) and National High-tech R&D Program of China (2015AA015305).

References

1. Gudivada, V., Baeza-Yates, R., Raghavan, V.: Big data: Promises and problems. Computer **48**(3), 20–23 (2015)
2. Gugnani, S., Blanco, C., Kiss, T., Terstyanszky, G.: Extending science gateway frameworks to support big data applications in the cloud. Journal of Grid Computing, pp. 1–13 (2016)
3. Lafferty, J.D., Mccallum, A., Pereira, F.C.N.: Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: Proceedings of the Eighteenth International Conference on Machine Learning, ICML., pp. 282–289. ACM (2001)
4. Kim, M.: Mixtures of conditional random fields for improved structured output prediction. IEEE Trans. Neural Netw. Learn. Syst. **28**(5), 1233–1240 (2017)
5. He, X., Zemel, R.S., Carreira-Perpiñán, M.Á.: Multiscale conditional random fields for image labeling. In: 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 2, pp. II–695. IEEE (2004)
6. Li, S.Z.: Markov Random Field Modeling in Image Analysis. Springer Science & Business Media (2009)

7. Yang, L., Zhou, Y.: Exploring feature sets for two-phase biomedical named entity recognition using semi-crfs. Knowl. Inf. Syst. **40**(2), 439–453 (2013)
8. Tsai, T.-h., Chou, W.-C., Wu, S.-H., Sung, T.-Y., Hsiang, J., Hsu, W.-L.: Integrating linguistic knowledge into a conditional random fieldframework to identify biomedical named entities. Expert Syst. Appl. **30**(1), 117–128 (2006)
9. Settles, B.: Abner: an open source tool for automatically tagging genes, proteins and other entity names in text. Bioinformatics **21**(14), 3191–3192 (2005)
10. Sha, F., Pereira, F.: Shallow parsing with conditional random fields. In: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology, vol. 1, pp. 134–141. Association for Computational Linguistics (2003)
11. Eddy, S.R.: Hidden markov models. Curr. Opin. Struct. Biol. **6**(3), 361–365 (1996)
12. Rabiner, L.R., Juang, B.-H.: An introduction to hidden markov models. IEEE ASSP Mag. **3**(1), 4–16 (1986)
13. McCallum, A., Freitag, D., Pereira, F.C.: Maximum entropy markov models for information extraction and segmentation. ICML **17**, 591–598 (2000)
14. Sun, C., Guan, Y., Wang, X., Lin, L.: Rich features based conditional random fields for biological named entities recognition. Comput. Biol. Med. **37**(9), 1327–1333 (2007)
15. Apache, Hadoop, Website. http://hadoop.apache.org (2015)
16. Spark, Website. http://spark.apache.org (2015)
17. Sutton, C., McCallum, A.: An introduction to conditional random fields for relational learning. In: Introduction to Statistical Relational Learning, pp. 93–128 (2006)
18. Byrd, R.H., Lu, P., Nocedal, J., Zhu, C.: A limited memory algorithm for bound constrained optimization. SIAM J. Sci. Comput. **16**(5), 1190–1208 (1995)
19. Vishwanathan, S., Schraudolph, N.N., Schmidt, M.W., Murphy, K.P.: Accelerated training of conditional random fields with stochastic gradient methods. In: Proceedings of the 23rd International Conference on Machine Learning, pp. 969–976. ACM (2006)
20. Bottou, L.: Large-scale machine learning with stochastic gradient descent. In: Proceedings of COMPSTAT'2010, pp. 177–186. Springer (2010)
21. Zhang, T.: Solving large scale linear prediction problems using stochastic gradient descent algorithms. In: Proceedings of the Twenty-first International Conference on Machine Learning, p. 116. ACM (2004)
22. Weiss, Y., Freeman, W.T.: On the optiMality of solutions of the max-product belief-propagation algorithm in arbitrary graphs. IEEE Trans. Inf. Theory **47**(2), 736–744 (2001)
23. Yedidia, J.S., Freeman, W.T., Weiss, Y., et al.: Generalized belief propagation. NIPS **13**, 689–695 (2000)
24. David, M.W.P.: Evaluation: From precision, recall and f-measure to roc, informedness, markedness and correlation. J. Mach. Learn. Technol. **2**(1), 37–63 (2007)
25. Liu, D.C., Nocedal, J.: On the limited memory bfgs method for large scale optimization. Math. Programm. **45**(1-3), 503–528 (1989)
26. Pearl, J.: Reverend bayes on inference engines: A distributed hierarchical approach. In: Proceedings of the Second National Conference on Artificial Intelligence, pp. 133–136. AAAI-82. AAAI Press (1982)
27. Geman, S., Geman, D.: Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. IEEE Trans. Pattern Anal. Mach. Intell. **6**(6), 721–741 (1984)
28. Rahman, H., Hahn, T., Segall, R.: Advanced feature-driven disease named entity recognition using conditional random fields. In: The ACM International Conference, pp. 469–469 (2016)
29. Finkel, J., Dingare, S., Nguyen, H.: Exploiting context for biomedical entity recognition: from syntax to the web. In: International Joint Workshop on Natural Language Processing in Biomedicine and ITS Applications. Association for Computational Linguistics, pp. 397–406 (2004)
30. Kim, J.D., Ohta, T., Tateisi, Y.: Genia corpus–semantically annotated corpus for bio-textmining. Bioinformatics **19**(1), i180–2 (2003)
31. Tang, Z., Jiang, L., Yang, L., Li, K., Li, K.: Crfs based parallel biomedical named entity recognition algorithm employing mapreduce framework. Clust. Comput. **18**(2), 493–505 (2015)
32. Mai, F., Wu, S., Cui, T.: Improved Chinese Word Segmentation Disambiguation Model Based on Conditional Random Fields. Springer International Publishing (2015)
33. bakeoff2005, Website. http://sighan.cs.uchicago.edu/bakeoff2005/ (2015)
34. Wang, Y., Lu, W., Lou, R., Wei, B.: Improving mapreduce performance with partial speculative execution. J. Grid Comput. **13**(4), 587–604 (2015)
35. Rasooli, A., Down, D.G.: Guidelines for selecting hadoop schedulers based on system heterogeneity. J. Grid Comput. **12**(3), 499–519 (2014)
36. Mahout, Website. http://mahout.apache.org (2015)
37. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
38. del, S., Río, V.L., Benítez, J.M., Herrera, F.: On the use of mapreduce for imbalanced big data using random forest. Inf. Sci. **285**, 112–137 (2014)
39. Dahiphale, D., Karve, R., Vasilakos, A.V., Liu, H., Yu, Z., Chhajer, A., Wang, J., Wang, C.: An advanced mapreduce: cloud mapreduce, enhancements and applications. IEEE Trans. Netw. Serv. Manag. **11**(1), 101–115 (2014)
40. Singh, K., Guntuku, S.C., Thakur, A., Hota, C.: Big data analytics framework for peer-to-peer botnet detection using random forests. Inf. Sci. **278**, 488–497 (2014)
41. Bajaber, F., Elshawi, R., Batarfi, O., Altalhi, A., Barnawi, A., Sakr, S.: Big data 2.0 processing systems: Taxonomy and open challenges. J. Grid Comput. **14**(3), 1–27 (2016)
42. Pal, C., Sutton, C., McCallum, A.: Sparse forward-backward using minimum divergence beams for fast training of conditional random fields. In: 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on Acoustics, Speech and Signal Processing, vol. 5, pp. V–V. IEEE (2006)
43. Cohn, T.: Efficient inference in large conditional random fields. In: Machine Learning: ECML 2006, pp. 606–613. Springer (2006)
44. Jeong, M., Lin, C.-Y., Lee, G.G.: Efficient inference of crfs for large-scale natural language data. In: Proceedings of the ACL-IJCNLP 2009 Conference Short Papers, pp. 281–284. Association for Computational Linguistics (2009)

45. Lavergne, T., Cappé, O., Yvon, F.: Practical very large scale crfs. In: Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, pp. 504–513. Association for Computational Linguistics (2010)
46. Lin, X., Zhao, L., Yu, D., Wu, X.: Distributed training for conditional random fields. In: 2010 International Conference on Natural Language Processing and Knowledge Engineering (NLP-KE), pp. 1–6. IEEE (2010)
47. Piatkowski, N., Morik, K.: Parallel inference on structured data with crfs on gpus. In: International Workshop at ECML PKDD on Collective Learning and Inference on Structured Data (COLISD2011) (2011)
48. Li, K., Ai, W., Zhang, F., Jiang, L., Li, K., Hwang, K.: Hadoop recognition of biomedical named entity using conditional random fields. IEEE Trans. Parallel Distrib. Syst. **26**(11), 3040–3051 (2015)