1

Parallel Large-Scale Feature Selection

Jeremy Kubica[a], Sameer Singh[b], and Daria Sorokina[c]

1.1 Introduction

The set of features used by a learning algorithm can have a dramatic impact on the performance of that algorithm. Including extraneous features can make the learning problem harder by adding useless, noisy dimensions that lead to *over-fitting* and increased computational complexity. Conversely, leaving out useful features can deprive the model of important signals. The problem of feature selection is to find a subset of features that allows the learning algorithm to learn the "best" model in terms of measures such as accuracy or model simplicity.

The problem of feature selection continues to grow in both importance and difficulty as extremely high-dimensional data sets become the standard in real-world machine learning tasks. Scalability can become a problem for even simple approaches. For example, common feature selection approaches that evaluate each new feature by training a new model containing that feature require a learning a linear number of models each time they add a new feature. This computational cost can add up quickly when we are iteratively adding many new features. Even techniques that use relatively computationally inexpensive tests of a feature's value, such as mutual information, require at least linear time in the number of features being evaluated.

As a simple illustrative example consider the task of classifying websites. In this case the data set could easily contain many millions of examples. Just including very basic features such as text unigrams on the page or HTML tags could easily provide many thousands of potential features for the model. Considering more complex attributes such as bigrams of words

[a] Google Inc.
[b] University of Massachusetts
[c] Carnegie Mellon University

or co-occurrences of particular HTML tags can dramatically drive up the complexity of the problem.

Similar large scale, high dimensional problems are now common in other applications such as internet algorithms, computational biology or social link analysis. Thus as we consider feature selection on modern data sets, traditional single machine algorithms may no longer be feasible to produce models in reasonable time.

In this chapter we examine parallelizing feature selection algorithms for logistic regression using the map-reduce framework. In particular, we examine the setting of *forward* feature selection where on every step new features are added to an existing model. We describe and compare three different techniques for evaluating new features: Full forward feature selection (Whitney, 1971), grafting (Perkins et al., 2003), and single feature optimization (Singh et al., 2009). While each of these techniques provide fast greedy approaches to the full feature selection problem, they still scale poorly with the number of features. We show how each of these techniques can naturally be parallelized to gracefully scale to much larger feature sets.

Our discussion focuses on the logistic regression learning algorithm. Recent comparison studies of machine learning algorithms in high-dimensional data have shown that logistic regression, along with Random Forests and SVMs, is a top performing algorithm for high dimensional data (Caruana et al., 2008). Given the fact that logistic regression is often faster to train than more complex models like Random Forests and SVMs (Komarek and Moore, 2005), in many situations it can be a preferred method for dealing with large-scale high dimensional data sets.

1.2 Logistic Regression

Logistic regression is a simple model for predicting the probability of event and is often used for binary classification. Assume that we have a data set containing N data points $\{(\vec{x}_i, y_i)\}, 1 \leq i \leq N$, where $\vec{x}_i$ are the vectors of input feature values and $y_i \in \{0, 1\}$ are binary response values. Logistic regression represents log odds of the event as a linear model:

$$log\left(\frac{p}{1-p}\right) = \vec{\beta} \cdot \vec{x} \tag{1.1}$$

Here $p = P(y = 1)$ is the predicted probability of a positive outcome and $\vec{\beta}$ is the vector of model parameters. Equation (1.1) is equivalent to the

following representation of p:

$$p = f_{\vec{\beta}}(\vec{x}) = \frac{e^{\vec{\beta}\cdot\vec{x}}}{1 + e^{\vec{\beta}\cdot\vec{x}}} \tag{1.2}$$

Therefore the logistic regression model is completely defined by the vector of coefficients $\vec{\beta}$. The *link function* $f_{\vec{\beta}}(\vec{x})$ defines a sigmoid that translates the linear function of $\vec{\beta}\cdot\vec{x}$ onto the range $[0, 1]$. It is useful to note that this model can be extended to categorical prediction using the multinomial logit and that many of the techniques described below can similarly be adapted.

Logistic regression models are most often learned by *maximum likelihood estimation*, which finds the $\vec{\beta}$ that maximizes the probability of the data given the model:

$$\vec{\beta}_{learned} = \underset{\vec{\beta}}{\operatorname{argmax}}\, P(\mathbf{Y}|\vec{\beta}, \mathbf{X}) = \underset{\vec{\beta}}{\operatorname{argmax}} \left(\prod_{i=1}^{N} f_{\vec{\beta}}(\vec{x}_i)^{y_i} (1 - f_{\vec{\beta}}(\vec{x}_i))^{(1-y_i)} \right) \tag{1.3}$$

where $\mathbf{X} = \{\vec{x}_1, \vec{x}_2, \cdots, \vec{x}_N\}$ is the full $N \times D$ data set. Equivalently we can maximize the model's log-likelihood:

$$\vec{\beta}_{learned} = \underset{\vec{\beta}}{\operatorname{argmax}} \sum_{i=1}^{N} \left(y_i \ln f_{\vec{\beta}}(\vec{x}_i) + (1 - y_i) \ln(1 - f_{\vec{\beta}}(\vec{x}_i)) \right) \tag{1.4}$$

which simplifies the mathematics. Since there is no closed form solution to this maximization, the standard approach to solving it is to use an iterative algorithm such as Newton Raphson (Hastie et al., 2001), Fisher's scoring (Komarek and Moore, 2005) or coordinate descent (Friedman et al., 2008). These approaches can still be computationally intensive, especially for high dimensional problems.

For the rest of the discussion we assume that the values of $\vec{x}$ are binary or continuous. For general categorical attributes, we use the standard technique of unrolling categorical attributes into disjoint binary attributes. For example, we can represent a single categorical attribute `COLOR` $\in$ { `RED`, `BLUE`, `GREEN` } as three Boolean attributes: `COLOR_IS_RED`, `COLOR_IS_BLUE` and `COLOR_IS_GREEN`. Thus a k-valued feature becomes k disjoint binary features that form a logical grouping, called a *feature class*. The features in a feature class are by definition constrained to be completely disjoint.

1.3 Feature Selection

The goal of feature selection is to find a subset of features that produces the "best" model $f_{\vec{\beta}}(\vec{x})$ for the data set $\{(\vec{x}, y)\}$. Often this means finding a small model that performs well on a given loss metric. In this chapter we primarily focus on cases where "best" is defined as the model's (unpenalized) likelihood on the training or test set. However, the techniques described can easily be adapted to other scoring measures.

An exhaustive approach for feature selection on a set of D possible features would simply learn models for all 2^D possible combinations of features and directly evaluate their performance. However, the cost of this approach grows exponentially with the number of features and this method becomes completely unfeasible for even small feature sets.

A range of feature selection techniques have been developed to avoid this combinatorial explosion while still accurately finding good sets of features. These techniques vary widely in how they measure quality of the features and their knowledge of the underlying algorithm. One common split is between *wrapper* algorithms, techniques that utilize knowledge of the underlying model, and *filter* algorithms that are independent of the underlying learning algorithm.

In this chapter we focus on the *forward selection* wrapper framework introduced by Whitney (1971). Forward selection algorithms incrementally build feature sets by adding new features to the current model. Such methods are expected to perform well, but often at the cost of high computational complexity (John et al., 1994). Below we describe three different techniques for evaluating new features in this framework: Full forward feature selection (Whitney, 1971), grafting (Perkins et al., 2003), and single feature optimization (Singh et al., 2009). For a good discussion of other feature selection techniques also see (Guyon and Elisseeff, 2003).

1.3.1 Forward Feature Selection

Forward feature selection is a heuristic that significantly reduces the number of feature sets that are evaluated (Whitney, 1971). We begin with an empty model. Then on each iteration of the algorithm we choose a feature that gives the best performance when added to the current set of features. Each feature is evaluated by adding it to the current set of features, relearning the whole model, and evaluating the model under the desired score metric. This means that on d-th iteration we build $D - (d - 1)$ models, where D is the original number of candidate features. Thus the overall number of models to

build and evaluate becomes quadratic in the number of features. It is better than exponential, but the complexity is still very high when all $\vec{\beta}$ in every model are learned by a complex iterative method.

Throughout the rest of the chapter we use the notation x_{id} and β_d to denote features and coefficients in the current model. And we use the notation x'_{id} and β'_d to denote features that are being evaluated for addition to the model. In this notation forward feature selection starts with an empty set of features, $\vec{x} = \{\}$, and repeatedly adds the best feature from $\vec{x}'$ to $\vec{x}$.

1.3.2 Single Feature Optimization (SFO)

Ideally we would like to evaluate each new feature in the context of a fully learned model containing both that feature and the current set of features. However, as described above, this presents a computational challenge. For even moderate data set sizes and numbers of features, this may not be feasible.

The **S**ingle **F**eature **O**ptimization (SFO) heuristic speeds up this evaluation by retaining coefficients from the current best model and optimizing only the coefficient β'_d of the new feature (Singh et al., 2009). The result is an approximate model that can be evaluated. This way we create $D-(d-1)$ *approximate* models on each iteration of forward selection. After evaluating them and choosing the best feature, we rerun the full logistic regression to produce a fully learned model that includes the newly selected feature(s). We begin with this model for the next iteration of features selection, so the approximation errors do not compound. As we rerun the full logistic regression solver only once on each iteration, we need to learn D models — linear in the number of features as opposed to quadratic.

We can quickly learn an approximate model by limiting the optimization to only the coefficient of the new feature. We hold the previous model parameters constant and perform one dimensional optimization over the new coefficient. For each new feature x'_d, we compute an estimated coefficient β'_d by maximizing the log-likelihood L with the new feature:

$$\underset{\beta'_d}{\operatorname{argmax}} \sum_{i=1}^{N} \left(y_i \ln f_{\vec{\beta}^{(d)}}(\vec{x}_i^{(d)}) + (1 - y_i) \ln(1 - f_{\vec{\beta}^{(d)}}(\vec{x}_i^{(d)})) \right) \tag{1.5}$$

where $\vec{x}_i^{(d)} = \vec{x}_i \cup \{x'_{id}\}$ is the set of features in the model and the candidate feature. Similarly $\vec{\beta}^{(d)}$ includes the candidate coefficient β'_d. Thus

$$f_{\vec{\beta}^{(d)}}(\vec{x}_i^{(d)}) = \frac{e^{\vec{\beta} \cdot \vec{x}_i + x'_{id}\beta'_d}}{1 + e^{\vec{\beta} \cdot \vec{x}_i + x'_{id}\beta'_d}} \tag{1.6}$$

There are a variety of optimization approaches that we could use to solve Equation 1.5. We use Newton's method to the maximize the log-likelihood L:

$$\frac{\partial L}{\partial \beta'_d} = 0 \tag{1.7}$$

We start at $\beta'_d = 0$ and iteratively update β'_d using the standard update:

$$\beta'_d = \beta'_d - \frac{\frac{\partial L}{\partial \beta'_d}}{\frac{\partial^2 L}{\partial \beta'^2_d}} \tag{1.8}$$

until convergence. In the case where L is the log-likelihood in (1.5), the derivatives simplify to:

$$\frac{\partial L}{\partial \beta'_d} = \sum_{i=1}^{N} x'_{id}(y_i - f_{\vec{\beta}^{(d)}}(\vec{x}_i^{(d)})) \tag{1.9}$$

$$\frac{\partial^2 L}{\partial \beta'^2_d} = -\sum_{i=1}^{N} x'^2_{id} f_{\vec{\beta}^{(d)}}(\vec{x}_i^{(d)})(1 - f_{\vec{\beta}^{(d)}}(\vec{x}_i^{(d)})) \tag{1.10}$$

This optimization only needs to iterate over those records that contain the new feature, $x'_{id} = 1$, and thus can be very efficient for sparse data.

Once we have the approximate model containing the new feature(s), it is trivial to use it to compute standard performance metrics such as log-likelihood, AUC, or prediction error. Thus we can score the new feature class by the directly evaluating the approximate model.

An obvious drawback of such single feature optimization is that we are not relearning the remaining coefficients. Therefore we can only get an approximate estimate of the effect of adding the new feature. For example, we will *underestimate* the performance of the new model on training set metrics. Despite this potential drawback, this limited optimization still can provide a strong signal.

Feature Class Optimization

Many real-world problems contain categorical attributes that can be exploded into a series of binary features. As noted above, the resulting features from a single feature class are by definition disjoint. Since we are holding all of the other coefficients fixed, we can optimize each feature *independently* of others and later combine the resulting coefficients to form a complete model. Further, each of these optimizations only needs to run over those records containing the relevant feature. For a k-valued categorical attribute

that has been unrolled into $\{x'_1, \ldots, x'_k\}$ we estimate $\{\beta'_1, \ldots, \beta'_k\}$ by solving Equation 1.5 independently for each of the resulting binary features. Thus we can trivially break the problem of evaluating categorical attributes into a series of smaller independent optimizations. Such unrolling is particularly well suited for the single feature optimization described above.

1.3.3 Grafting

Perkins et al. (2003) propose using the loss function's gradient with respect to the new feature to decide whether to add this feature. The gradient of the loss function with respect to the new feature is used as an indication of how much that new feature would help the model. As with SFO above this gradient is computed independently for each feature using the current coefficients in the model.

More formally grafting uses the magnitude of gradient with respect to that feature's coefficient:

$$\left|\frac{\partial L}{\partial \beta'_d}\right| = \left|\sum_i x'_{id}(y_i - p_i)\right| \tag{1.11}$$

where p_i is the model's prediction for the feature vector $\vec{x}_i$.

At each step the feature with the largest magnitude of gradient is added to the model:

$$\underset{\beta'_d}{\operatorname{argmax}} \left|\frac{\partial L}{\partial \beta'_d}\right| \tag{1.12}$$

In a gradient descent algorithm this is equivalent to initially fixing coefficients at zero and at each step allowing parameter with the greatest effect on the objective function to move away from zero. As with full forward feature selection and SFO, the non-zero coefficients are added one at a time and the model is fully relearned after each feature is added.

1.3.4 Multi-Class Prediction Problems

The above forward feature selection techniques can also be easily applied to multiple class prediction problems. One standard approach is to learn a separate model for each class, by treating the problem as a series of binary classification problems. Each model is queried at prediction time and the class label with the highest predicted value is chosen. Thus if we have C different class labels, we need to train and query C models.

In the case of feature evaluation we want to find the best feature to add

for each of these models. We can do this in several different ways. The easiest approach is to create a separate training data set for each classification label and run feature evaluation for each of these data sets. This approach is also equivalent to creating a single data set that creates C modified copies of each record by appending the corresponding class label to each feature in that record, including the bias key. The result is a data set with $N \cdot C$ records that effectively represents C different data sets.

For concreteness consider a classification problem with three labels a, b, and c. In this case a single input record with features and label a, $\{1, 2, 3\} = a$, would be transformed into 3 records: $\{a : 1, a : 2, a : 3\} = 1$, $\{b : 1, b : 2, b : 3\} = 0$ and $\{c : 1, c : 2, c : 3\} = 0$. Since the modified copies of the features are completely disjoint for each model, the overall model acts as C independent models.

1.4 Parallelizing Feature Selection Algorithms

All three of the forward feature evaluation methods described above can become computationally expensive as the size of the data grows. Even the SFO and gradient evaluations require at least a single pass through the data set for each iteration of the feature evaluation algorithm. One approach for scaling up these algorithms to even larger data sets is to parallelize the evaluation and distribute the cost over many machines.

1.4.1 Parallel Full Forward Feature Selection

The basic forward feature selection approach is trivial to parallelize by just partitioning evaluation of features. At each iteration of forward feature selection we want to determine the "best" feature to add. We can do this in parallel by partitioning the D features to be evaluated into K subsets and evaluate each subset of features independently on K machines. Each machine then only needs to learn and evaluate $\frac{D}{K}$ full models per iteration. This can lead to significant savings over a single machine implementation for large D.

The disadvantage of this approach is that, although the work is distributed over K machines, we still have to relearn a full model for each feature - which can be costly. Thus this approach does not help us scale to large data sets in terms of the number of records.

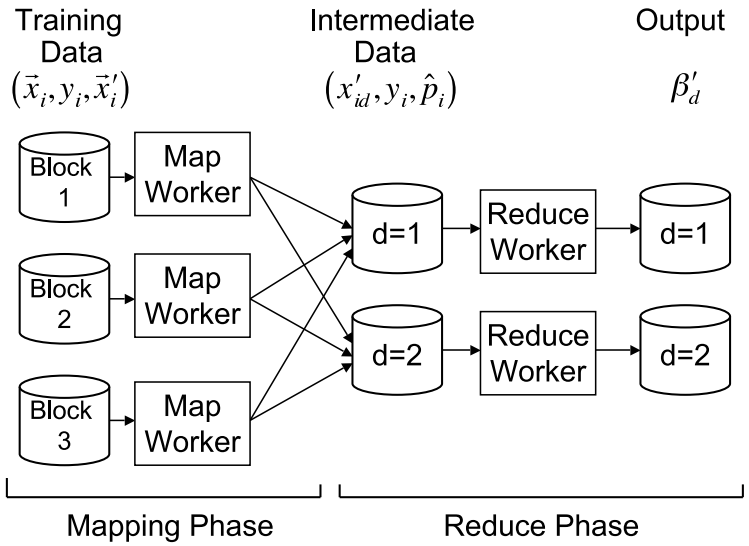


Figure 1.1 Conceptual data flow of the feature evaluation map-reduce with 3 input data blocks and 2 features. In the mapping stage separate processors operate on blocks of training data $(\vec{x}_i, y_i, \vec{x}'_i)$ to produce data sets of statistics $(\vec{x}'_i, y_i, p_i)$ for each new feature in the record $\vec{x}'_i$. In the reduce phase separate processors operate on each of these data sets, computing estimated coefficients for the new features β'_d.

1.4.2 Parallel SFO

The Single Feature Optimization heuristic is especially well suited for parallelization. In particular we can effectively partition the computation first over the records and then over the candidate features. This allows SFO to scale to both high dimensionality and a large number of records. We developed the SFO evaluation algorithm in the context of the map-reduce framework (Dean and Ghemawat, 2004).

At a high level the map-reduce framework consists of two distinct phases, mapping and reducing, which parallelize the computation over the training records and potential features respectively. During the mapping phase the algorithm iterates over the input records and collects the statistics needed to estimate the candidate coefficients for each feature. During the reduce phase the algorithm operates on these per-feature sets of data to compute the estimated coefficients. (See Figure 1.1 for the illustration of the process)

More formally the SFO map-reduce algorithm consists of three steps:

Algorithm 1 The SFO map function. Takes a data block $\{\mathbf{X}, \vec{y}\}$ and a vector of coefficients $\vec{\beta}$. Produces a data set $T_d = \bigcup_{\{i:x'_{id}=1\}}(\vec{x}'_i, y_i, p_i)$ of statistics for each candidate feature.

```
for all records {x_i, y_i, x'_i} in the data {X, y} do
  Compute the predicted probability: p_i = f_β(x_i)
  for all active candidate features x'_id ∈ x'_i do
    Output (x'_i, y_i, p_i)
  end for
end for
```

1. **Mapping Phase** (*parallel over records, Algorithm 1*):

   Iterate over the training records, computing the raw information needed to later estimate the coefficients. For each training record, $(\vec{x}_i, y_i, \vec{x}'_i)$, compute the predicted probability of the current model $p_i = f_{\vec{\beta}}(\vec{x}_i)$. Then for each of the candidate features that is active in this record output the necessary statistics, $(\vec{x}'_i, y_i, p_i)$, to the reduce phase. We can collect the lists of predicted probabilities and true outcomes in a separate, intermediate data set for each of the candidate features: $T_d = \bigcup_{\{i:x'_{id}=1\}}(\vec{x}'_i, y_i, p_i)$. The reduce phase will then be able to use each of these lists to independently compute the coefficients. Note that since the input features are binary we only need to collect statistics for a candidate feature in those records where that feature is active.
2. **Reduce Phase** (*parallel over features, Algorthim 2*):

   For each feature being evaluated use the corresponding outputs of the mapping phase, $T_d = \bigcup_{\{i:x'_{id}=1\}}(\vec{x}'_i, y_i, p_i)$, to compute an estimated coefficient β'_d as in § 1.3.2. Here we are computing an approximate coefficient by learning a correction to the predicted probabilities for the records in which this feature occurs. We can also aggregate estimated changes in training set log-likelihood by using the estimated coefficients.
3. **Post-processing**: Aggregate the coefficients for all features in the same feature class.

Since the features are always treated independently up to the post-processing phase, we can also use this algorithm to evaluate different non-disjoint feature classes in a single run. This allows us to trivially explore *all* potential feature classes in parallel.

In the ideal case parallelizing this computation over K machine reduces the runtime by a factor of K. We can see this by looking at the running time of the component phases. The running time of the mapping phase is

Algorithm 2 The SFO reduce function. Takes an data set $T_d = \bigcup_{\{i:x'_{id}=1\}}(\vec{x}'_i, y_i, p_i)$ for a candidate feature. Produces an estimated coefficient β'_d for that feature.

$\beta'_d = 0$
while β'_d has not converged **do**
  Initialize the first and second derivatives: $\frac{\partial L}{\partial \beta'_d} = \frac{\partial^2 L}{\partial \beta'^2_d} = 0$
  for all records $(\vec{x}_i, y_i, p_i) \in T_d$ **do**
    $a_i = \log\left(\frac{p_i}{1-p_i}\right)$
    Compute the new prediction under β'_d: $p'_i = \frac{e^{a_i+\beta'_d}}{1+e^{a_i+\beta'_d}}$
    Update the first derivative sum: $\frac{\partial L}{\partial \beta'_d} = \frac{\partial L}{\partial \beta'_d} + (y_i - p'_i)x'_{id}$
    Update the second derivative sum: $\frac{\partial^2 L}{\partial \beta'^2_d} = \frac{\partial^2 L}{\partial \beta'^2_d} - p'_i(1-p'_i)x'^2_{id}$
  end for
  Update the estimate of β'_d using a Newton's step: $\beta'_d = \beta'_d - \frac{\partial L}{\partial \beta'_d} / \frac{\partial^2 L}{\partial \beta'^2_d}$
end while

approximately $O(\frac{N \cdot D_{\max}}{K})$ where N is the number of records and $D_{\max}$ is the maximum number of features active in any record. This running time follows directly from the $O(D_{\max})$ cost of computing a predicted probability using the current feature set and storing it for each active candidate feature that is required for each of the N records. Similarly, the running time of the reduce phase is $O(\frac{N_{\max} \cdot D}{K})$ where $N_{\max}$ is the maximum number of records containing a single new feature. This follows from the cost of computing the estimated coefficient for each candidate feature. Note that the $N_{\max}$ cost can be significant for common features, such as the bias term. Singh et al. (2009) describe a method for further reducing this running time by using histograms.

However, it is important to note that we do not expect to see this ideal speed-up often in practice. In real world systems there is also a (non-trivial) per-machine start up cost that limits the benefit of adding more machines.

We can also use the same framework to compute test set metrics with a second map-reduce-based algorithm. We can then use these test set metrics to rank the candidate features and select the best one. In this case the algorithm knows the estimated coefficients $\vec{\beta}'$ of all the candidate features and the phases become:

1. **Mapping Phase** (*parallel over records*): Iterate over the test records

$(\vec{x}_i, y_i, \vec{x}'_i)$ and for each new feature $\vec{x}'_{id}$: compute the predicted probabilities under the old model $p_i = f_{\vec{\beta}}(\vec{x}_i)$ and the new model $p'_{id} = f_{\vec{\beta}^{(d)}}(\vec{x}_i)$.
2. **Reduce Phase** (*parallel over features*): For each evaluated feature β'_d use the predictions from the old model p_i and new model p'_{id} to compute the model scores and difference in scores.
3. **Post-processing**: Aggregate the score changes for all features in the same feature class.

1.4.3 Parallel Grafting

We can also apply a similar map-reduce framework to other feature evaluation methodologies, such as grafting (Perkins et al., 2003). The grafting approach to feature evaluation chooses the feature that has the largest magnitude gradient. Formally we choose the next feature using:

$$\operatorname*{argmax}_{\beta'_d} \left| \frac{\partial L}{\partial \beta'_d} \right| = \operatorname*{argmax}_{\beta'_d} \left| \sum_i x'_{id}(y_i - p_i) \right| \tag{1.13}$$

Here we note that we can compute the gradient independently for each record and aggregate the sum of the gradients independently for each feature.

The grafting map-reduce algorithm, given in Algorithms 3 and 4, consists of three steps:

1. **Mapping Phase** (*parallel over records*): Iterate over the training records $(\vec{x}_i, y_i, \vec{x}'_i)$, computing which new features are active in $\vec{x}'_i$, the predicted probability of the current model $p_i = f_{\vec{\beta}}(\vec{x}_i)$, and the gradient:

$$\frac{\partial L_i}{\partial \beta'_d} = y_i - p_i \quad \forall x'_{id} \in \vec{x}'_i \tag{1.14}$$

2. **Reduce Phase** (*parallel over features*): For each evaluated feature β'_d compute the absolute value of the sum of the per-record gradients.
3. **Post-processing**: Aggregate the gradients for all features in the same feature class.

Since the features are always treated independently up to the post-processing phase, we can trivially explore all potential feature classes in parallel.

1.4.4 Other Related Algorithms

There exists a wide range of other feature selection algorithms and methodologies that can also be parallelized to improve scalability. While a full

Algorithm 3 The grafting map function. Takes a data block $\{\mathbf{X}, \vec{y}\}$ and a vector of coefficients $\vec{\beta}$. Produces an data set T_d for each candidate feature.

for all records $\{\vec{x}_i, y_i, \vec{x}'_i\}$ in the data $\{\mathbf{X}, \vec{y}\}$ **do**
  Compute the gradient for that record: $g_i = y_i - f_{\vec{\beta}}(\vec{x}_i)$
  for all active candidate features $x'_{id} \in \vec{x}'_i$ **do**
    Output g_i: $T_d = T_d \cup \{g_i\}$
  end for
end for

Algorithm 4 The grafting reduce() function. Takes a data set T_d for a candidate feature. Produces an absolute sum of gradient G_d.

$G_d = 0$
for all records $g_i \in T_d$ **do**
  $G_d = G_d + g_i$
end for

discussion of approaches beyond the scope of this chapter, below we provide a brief high level discussion of some other types of feature selection methodologies. Many of these techniques can easily be adapted to use the parallelization techniques described in this chapter.

In this chapter we focus on the category of wrapper methods. One easy approach to scaling techniques of this type are to parallelize the evaluation of features as described in §1.4.1. For example Garcia et al. (2006) describes an approach that evaluates many random subsets of features in parallel and then merges the best performing subsets. López et al. (2006) describes an evolutionary algorithm that parallelizes the evaluation, combination, and improvement of the solution population by distributing the work for different subsets of the population. This general approach could also be applied to many other recent forward-wrapper techniques, such as the algorithms of Della Pietra et al. (1997) and McCallum (2003), by evaluating new features in parallel. Della Pietra et al. (1997) describe a feature selection method for random fields that holds the features in the current model fixed and selects the new feature by minimizing the KL-divergence between the model and the empirical distribution. McCallum (2003) introduces a similar method for conditional random fields, but like SFO his algorithm chooses the feature that maximizes the new model's log-likelihood. Similarly the same approach could apply to *backward elimination* techniques, which start with a full feature set and iteratively remove the least helpful feature. As noted by Abe

(2005) backward feature selection also suffers from potentially high computational costs when the models are fully relearned.

In contrast to wrapper methods, filter methods attempt to select features using signals independent of the underlying model. Again some of these techniques could naturally be parallelized by distributing the computation by record or feature. Some techniques such as mutual information feature selection, which uses the mutual information between a feature and the class label (Lewis, 1992; Battiti, 1994), could even be implemented within the map-reduce framework. In this case the map phase would consist of counting feature occurrences and the reduction phase would consist of summing these counts and computing the mutual information. Similarly, Fleuret (2004) proposes a filter method based on conditional mutual information that chooses the feature that maximizes the minimum mutual information with the response variable conditioned on each of the features already in the model. This technique could also be implemented similar to the SFO algorithm with the current feature set used during mapping to compute counts of joint occurrences between the response variable, candidate features, and the current features.

Another general approach to feature selection is to automatically select features during the model training process. For example LASSO (Tibshirani, 1996) is a well known algorithm of this type. It uses regularization with an L_1 penalty to force the coefficients of non-useful attributes toward zero - encouraging sparsity in the model. Genkin et al. (2005) provide a good discussion of using regularization for sparsity in high dimensional problems. More recently Krishnapuram et al. (2004) proposed Bayesian methods for joint learning and feature selection. Since these algorithms rely on the learning algorithm itself for feature selection, both the parallelization and scalability are determined by the algorithms.

1.5 Experimental Results

We conduct a series of experiments to test the algorithms' effectiveness in determining the "next" feature to add to a model. Since our intention is to provide an accurate approximation of a full forward selection step, we empirically evaluate the performance of the approximate algorithms by using full forward feature selection as the ground truth. Note that it is not our intention to compare greedy forward selection against the full space of feature selection approaches.

In the experiments below we consider feature selection starting with dif-

ferent sized base models. We randomly choose a set F_B of *base* features from a pool of frequently occurring features and train a full model. By varying the size of the base feature set, we can examine the algorithms' performance as the incremental benefit of adding features decreases. We also choose a set F_E of candidate *evaluation* features from the remaining features to be evaluated for inclusion in the model. We then evaluate each of the candidate features with each algorithm and compare the results to full forward feature selection.

We also ran the same experimental set up to examine the accuracy of the resulting learning models. In these runs we held out a 10% test set and computed the AUC and log-likelihood on this data. Each evaluation model was trained on all of the training data using the full set of base features and the feature selected by the feature selection algorithm. Results are reported in average improvement in AUC and average percent improvement in log-likelihood. This tests gives an indication of how much the selected feature helps on the actual prediction task.

In addition to SFO and grafting we also consider two baseline feature evaluation approaches: random selection (rand) and mutual information (M. I.). Random selection provides a baseline by choosing the best feature to add at random without using the current model or data. Mutual information, a common filter algorithm, provides a more principled baseline by choosing the feature with the highest mutual information between that feature and the label:

$$MI_d = \sum_{y \in \{0,1\}} \sum_{x'_d \in \{0,1\}} P(x'_d, y) log \left(\frac{P(x'_d, y)}{P(x'_d)P(y)} \right) \tag{1.15}$$

where $P(x'_d, y)$, $P(x'_d)$ and $P(y)$ are the probabilities empirically computed from the data. It is important to note that when we are evaluating the performance of the algorithms in choosing the same features as forward feature selection we are *not* trying to compare the overall accuracy of mutual information versus the other techniques. Instead we are using mutual information as a baseline to help compare how well the approximate forward feature selection algorithms perform relative to each other.

We use a publicly available IRLS logistic regression package[1] to learn the logistic regression models (Komarek and Moore, 2005). In particular, this solver is used to learn the base models, fully retrained models and full-forward feature selection models.

[1] Available at http://www.autonlab.org/

Table 1.1 *Empirical results on the UCI Internet Ads data. The fraction of runs where SFO, grafting, mutual information and random selection selected the same feature as full forward feature selection.*

$\|F_B\|$	SFO	Grafting	M. I.	Rand.
0	0.96	0.96	0.84	0.00
50	1.00	0.88	0.68	0.00
100	0.84	0.84	0.64	0.00
150	0.96	0.64	0.52	0.00
200	0.88	0.84	0.72	0.04
250	0.88	0.80	0.72	0.00

1.5.1 UCI Internet Ads Data Set

First we examine the performance of the algorithms on a small-scale simple data set from the UCI repository (Asuncion and Newman, 2007), the internet ads data set. The internet ads data set includes 3279 image instances that are labeled as "Ad" or "Non-Ad" and contains 1558 features. We use the 1555 binary features for our candidate feature set. As described above, we examine the performance of the algorithms as the size of the base features is increased. Experiments are run 25 times in each setting with randomly chosen base features and 100 candidate features.

The results of the empirical evaluation, shown in Table 1.1, show a strong performance for the parallel algorithms in approximating full forward feature evaluation. Both SFO and grafting perform well above random selection ($\sim 1\%$ accuracy). As expected the results also show the features selected by mutual information often differ from those of forward feature selection.

Table 1.2 shows the average improvement in accuracy as a result of adding the feature selected by each method. Here we see strong and often comparable performance on all four of the feature selection approaches. All four approaches perform better than random selection.

1.5.2 RCV1-v2 Data Set

To demonstrate a more realistic data size for the distributed algorithms, we apply the algorithms to the RCV1-v2 data (Lewis et al., 2004) . This data consists of stemmed tokens from text articles on a range of topics. We examine accuracy for the binary prediction problem for each of the 5 largest

Table 1.2 *Empirical accuracy results on the UCI Internet Ads data using a 10% holdout test set. Average improvement of AUC (top) and percent improvement on log-likeihood (bottom) for a full trained model containing the feature selected by each technique.*

$\|F_B\|$	Full	SFO	Grafting	M. I.	Rand.
0	0.064	0.065	0.068	0.065	0.005
50	0.058	0.057	0.055	0.056	0.004
100	0.012	0.014	0.007	0.011	0.002
200	0.010	0.009	0.008	0.008	0.001
0	3.28	3.33	3.36	3.27	0.04
50	4.21	4.09	4.12	4.10	0.10
100	2.29	2.33	2.15	2.26	0.05
200	2.06	1.91	1.80	1.60	0.15

subcategories (`C15`, `C18`, `GPOL`, `M13`, `M14`) as the size of the base model is increased. For our purposes, we combine original training set and four test data sets into a single data set with $N = 804{,}679$ records. Furthermore, in order to bias the set of evaluation features to potentially beneficial features, we filter the feature set to include only the features that appear at least 5000 times in the data. This results in $D = 1{,}992$ possible features. Experiments were run 25 times in each setting with randomly chosen base features and 25 candidate features.

On the task of selecting the same features as full forward feature selection, Table 1.3 shows the parallel algorithms performing well. Both SFO and grafting perform well above random selection ($\sim 4\%$ accuracy), but below the performance of the full forward evaluation. Of the two approximate parallel algorithms, SFO often outperforms grafting.

It is also interesting to note the mutual information selects many of the same features as full forward feature selection when the number of base features is small. In fact, for empty models, $|F_B| = 0$, mutual information is often one of the top performing algorithms on this metric. However, as the size of the base model increases, the selections of mutual information begins to differ. This is expected, because, unlike both SFO and grafting, mutual information does not use information about the features currently in the model. Thus it can select features which look good on their own, but are correlated with features already in the model.

Table 1.3 *Empirical results on the RCV1-v2 data. The fraction of runs where SFO, grafting, mutual information and random selection selected the same feature as full forward feature selection.*

Label	$\|F_B\|$	SFO	Grafting	M. I.	Rand.
C15	0	0.96	0.68	0.96	0.08
C15	100	0.76	0.72	0.60	0.04
C15	500	0.60	0.44	0.36	0.00
C18	0	0.88	0.88	0.96	0.00
C18	100	0.72	0.72	0.68	0.04
C18	500	0.68	0.72	0.52	0.08
GPOL	0	1.00	0.80	1.00	0.00
GPOL	100	0.80	0.44	0.52	0.04
GPOL	500	0.76	0.80	0.60	0.00
M13	0	0.96	0.72	0.96	0.00
M13	100	0.76	0.56	0.56	0.04
M13	500	0.84	0.72	0.64	0.00
M14	0	0.88	0.64	0.96	0.00
M14	100	0.96	0.72	0.80	0.04
M14	500	0.88	0.64	0.52	0.00

Table 1.4 shows the average percent improvement in test set log-likelihood as a result of adding the feature selected by each method. Both full forward feature selection and SFO perform well, often selecting the features that provide good improvement in test set log-likelihoods. Mutual information is often also a strong contender, especially for smaller base models. As in the internet ads experiments, the test set AUC results, which are not shown here to save space, show little difference among the feature selection approaches.

1.5.3 Timing Results

To examine the effectiveness of using the map-reduce framework, we can examine the wall clock running time of the algorithm as we vary the number of machines K. Below we look at the *speed-up* over the single machine:

$$\text{SpeedUp}(K) = \frac{\text{Running time with 1 machine}}{\text{Running time with } K \text{ machines}} \tag{1.16}$$

An ideal speed-up is linear in K indicating perfect parallelization.

Table 1.4 *Empirical accuracy results on the RCV data using a 10% holdout test set. Average percent improvement of log-likelihood on a full trained model containing the feature selected by each technique.*

Label	$\|F_B\|$	Full	SFO	Grafting	M. I.	Rand.
C15	0	530.8	530.8	500.2	530.8	85.0
C15	100	318.9	314.6	291.4	270.0	76.0
C15	200	203.2	201.0	186.4	188.4	5.5
C18	0	86.6	86.3	80.5	86.3	8.0
C18	100	110.5	98.0	101.3	103.8	25.5
C18	200	82.9	81.9	78.4	80.7	1.5
GPOL	0	385.8	385.1	375.7	385.8	38.0
GPOL	100	123.7	123.1	118.3	117.2	21.9
GPOL	200	90.4	89.4	87.8	85.5	6.6
M13	0	165.9	165.7	154.4	166.2	12.9
M13	100	236.6	236.0	227.2	228.8	6.6
M13	200	129.8	122.9	120.1	123.6	2.8
M14	0	334.8	332.6	257.4	334.8	58.0
M14	100	183.3	196.9	167.1	190.2	27.3
M14	200	162.1	161.1	152.9	136.9	25.1

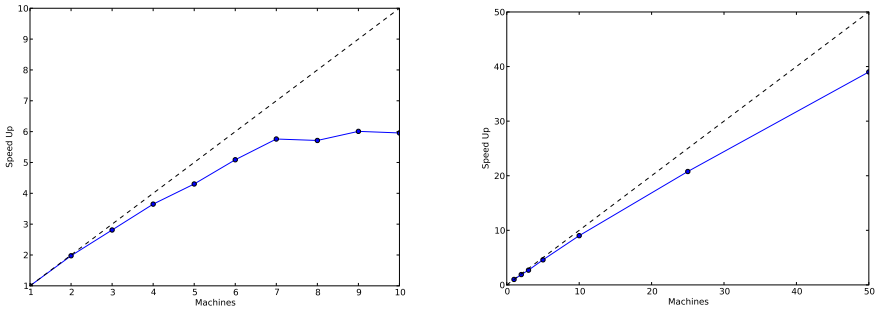
Simulated Data

We initially look at SFO's speed-up over two simulated data sets: one with 1,000,000 records and 50,000 features, and the other with 10,000,000 records and 100,000 features. The data was generated from random models with coefficients in the range $[-0.5, 0.5]$. Each record contains exactly 20 active features selected randomly without replacement. The base model has a uniform prior probability of 0.5 of generating a true label.

Figure 1.2 shows a clear benefit as we increase the number of machines. The deviation from ideal when using higher number of machines in the speed-up plots 1.2(a) and 1.2(b) occurs since the benefit of adding machines decreases as the per-machine startup costs begin to become an increasing factor. This overheads can include such factors as launching the relevant programs and copying the data to the machine. For example in Figure 1.2.a we do not see a benefit past 7 machines, because at that point we are dealing with a small problem of less than 150,000 records per machine.

Despite the decreasing returns with the number of machines, we expect this parallelization to provide significant wall clock savings for large data sets

or feature sets. This is observed by noticing that the deviation from ideal is less for the larger dataset (for the same number of machines). Further, we expect the marginal benefit of adding machines to *increase* with the computational cost of computing the features. This type of parallelization becomes more important when considering non-trivial features, such as those requiring string parsing, because we distribute the expensive computation over more machines.



(a) 1,000,000 records / 50,000 features (b) 10,000,000 records / 100,000 features

Figure 1.2 Speed-up plots of parallel SFO versus the number of machines used on simulated data, for (a) 1,000,000 records / 50,000 features, and (b) 10,000,000 records / 100,000 features. The dashed line represents ideal behavior.

RCV1-v2 Data

In looking at the RCV1-v2 data, we can further scale up the size of the problem by using the fact that the RCV1-v2 is actually a multi-class problem. We can choose C labels and learn binary models to predict the outcome for each label. As discussed in §1.3.4, we can learn independent models for each class label by effectively replicating the data set C times. For each class label `L` we create a new copy of the data set such that:

1. A record is positive if and only if its class label is `L` in the original data.
2. We create new copies of each attribute that include the label.

For example in the RCV1-v2 data we could modify the record { `world`, `qualify`, $\cdots$ } to be { `C15:world`, `C15:qualify`, $\cdots$ } in the data set for the `C15` label. Once we have the C data sets we can merge them into a single large data set and train a single model over them. In doing so we effectively learn a separate binary model for each label. Further we can use a single iteration of the feature selection algorithms to choose the best feature for *each* model by selecting the best feature for each label.

We used the above multi-class approach to test the scalability of the SFO algorithm. In our tests we looked at feature evaluation for the $C = 9$ subcategories that have $\geq 40,000$ positive records: C15, C17, C18, C31, E21, GPOL, M11, M13, M14. This provides a total data size of approximately 7.2 million records and 2.6 million features.

Figure 1.3 shows a clear benefit as we increase the number of machines. Although we have increased both the number of records and the dimensionality by a factor of 9, we have also increased the sparsity of data set. In particular, in the reduce phase the workers only need to process the data from the records that contain a given feature in the context of a given class label. This means that they have $C * D$ smaller chunks of work instead of D larger ones.

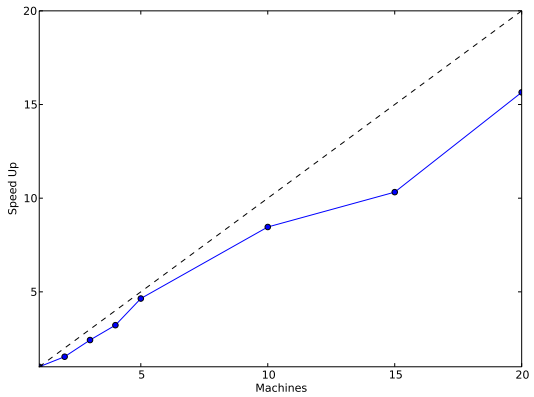


Figure 1.3 Speed-up plot of parallel SFO versus the number of machines on the RCV1-v2 data. The dashed line represents ideal behavior.

1.6 Conclusions

In this chapter we described three parallel algorithms for forward feature selection in logistic regression: full forward feature selection, grafting, and SFO. Both SFO and grafting provide computationally efficient approximations of the full evaluation of which features provide the most signal to the model. Both of these approximations are easily parallelizable within the map-reduce framework.

Empirically, we showed that the SFO heuristic results in a good performance and is comparable to techniques that relearn the whole model. In addition the approximate model can also be used to evaluate the fea-

ture's impact on a range of other metrics and on validation set performance. Further, the coefficients estimated by the SFO heuristic can provide useful starting points to relearn the model and can provide insights into the structure of the problem.

References

Abe, S. 2005. Modified backward feature selection by cross validation. In: *13th European Symposium On Artificial Neural Networks*.

Asuncion, A., and Newman, D. 2007. *UCI Machine Learning Repository*.

Battiti, R. 1994. Using mutual information for selecting features in supervised neural net learning. *IEEE Transactions on Neural Networks*, **5**, 537–550.

Caruana, R., Karampatziakis, N., and Yessenalina, A. 2008. An Empirical Evaluation of Supervised Learning in High Dimensions. In: *Proceedings of the 25th International Conference on Machine Learning, (ICML 2008)*.

Dean, J., and Ghemawat, S. 2004. MapReduce: Simplified Data Processing on Large Clusters. In: *OSDI'04: Sixth Symposium on Operating System Design and Implementation*.

Della Pietra, S., Della Pietra, V., and Lafferty, J. 1997. Inducing Features Of Random Fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **19**(4), 380–393.

Fleuret, F. 2004. Fast Binary Feature Selection with Conditional Mutual Information. *Journal of Machine Learning Research*, **5**, 1531–1555.

Friedman, J., Hastie, T., and Tibshirani, R. 2008. *Regularized Paths for Generalized Linear Models via Coordinate Descent*.

Garcia, D., Hall, L., Goldgof, D., and Kramer, K. 2006. A Parallel Feature Selection Algorithm from Random Subsets. In: *Proceedings of the 17th European Conference on Machine Learning and the 10th European Conference on Principles and Practice of Knowledge Discovery in Databases*.

Genkin, A., Lewis, D., and Madigan, D. 2005. *Sparse Logistic Regression for Text Categorization*.

Guyon, I., and Elisseeff, A. 2003. An introduction to variable and feature selection. *Journal of Machine Learning Research*, **3**(March), 1157–1182.

Hastie, T., Tibshirani, R., and Friedman, J. 2001. *The Elements of Statistical Learning*. Springer.

John, G., Kohavi, R., and Pfleger, K. 1994. Irrelevant Features and the Subset Selection Problem. Pages 121–129 of: *Proceedings of the Eleventh International Conference on Machine Learning, (ICML 1994)*. Morgan Kauffmann.

Komarek, P., and Moore, A. 2005. Making Logistic Regression A Core Data Mining Tool With TR-IRLS. In: *Proceedings of the 5th International Conference on Data Mining Machine Learning*.

Krishnapuram, B., Carin, L., and Hartemink, A. 2004. Joint Classifier and Feature Optimization for Comprehensive Cancer Diagnosis Using Gene Expression Data. *Journal of Computational Biology*, **11**(2-3), 227–242.

Lewis, D. 1992. Feature selection and feature extraction for text categorization. Pages 212–217 of: *Proceedings of the workshop on Speech and Natural Language*.

Lewis, D., Yang, Y., Rose, T., and Li, F. 2004. RCV1: A New Benchmark Collection for Text Categorization Research. *Journal of Machine Learning Research*, **5**, 361–397.

López, F., Torres, M., Batista, B., Pérez, J., and Moreno-Vega, M. 2006. Solving feature subset selection problem by a Parallel Scatter Search. *European Journal of Operational Research*, **169**(2), 477–489.

McCallum, A. 2003. Efficiently Inducing Features of Conditional Random Fields. In: *Conference on Uncertainty in Artificial Intelligence (UAI)*.

Perkins, S., Lacker, K., and Theiler, J. 2003. Grafting: Fast, Incremental Feature Selection by Gradient Descent in Function Space. *Journal of Machine Learning Research*, **3**, 1333–1356.

Singh, S., Kubica, J., Larsen, S., and Sorokina, D. 2009. Parallel Large Scale Feature Selection for Logistic Regression. In: *SIAM International Conference on Data Mining (SDM)*.

Tibshirani, R. 1996. Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society*, **58**(1), 267–288.

Whitney, A. 1971. A Direct Method of Nonparametric Measurement Selection. *IEEE Transactions on Computers*, **20**(9), 1100–1103.