Bounded Coordinate-Descent for Biological Sequence Classification in High Dimensional Predictor Space

Georgiana Ifrim
Cork Constraint Computation Centre (4C)
University College Cork
Cork, Ireland
g.ifrim@4c.ucc.ie

Carsten Wiuf
Bioinformatics Research Centre (BiRC)
C.F. Møllers Allé 8
DK-8000 Aarhus C, Denmark
wiuf@cs.au.dk

ABSTRACT

We present a framework for discriminative sequence classification where linear classifiers work directly in the explicit high-dimensional predictor space of all subsequences in the training set (as opposed to kernel-induced spaces). This is made feasible by employing a gradient-bounded coordinate-descent algorithm for efficiently selecting discriminative subsequences without having to expand the whole space. Our framework can be applied to a wide range of loss functions, including binomial log-likelihood loss of logistic regression and squared hinge loss of support vector machines. When applied to protein remote homology detection and remote fold recognition, our framework achieves comparable performance to the state-of-the-art (e.g., kernel support vector machines). In contrast to state-of-the-art sequence classifiers, our models are simply lists of weighted discriminative subsequences and can thus be interpreted and related to the biological problem – a crucial requirement for the bioinformatics and medical communities.

Categories and Subject Descriptors

H.3 [**Information Storage and Retrieval**]: Content Analysis and Indexing; I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Algorithms, Performance, Experimentation

Keywords

Greedy Coordinate-Descent, Sequence Classification, String Classification, Logistic Regression, Support Vector Machines

1. INTRODUCTION

Many problems in biology today require accurate computational prediction of properties. For example, the primary DNA sequence is a main determinant of functional and structural protein properties, yet little is known about this relationship and we must therefore turn to computational prediction for advancing our understanding. Likewise, accurate gene and motif prediction is of crucial importance for the annotation of recently sequenced genomes. To achieve this goal, we need machine learning techniques that are fast, highly scalable and preferably treat feature selection as an integral part of the learning algorithm. The latter requirement means that neither time nor expertise is invested in pre-processing the original data (e.g., for defining features) and no potentially hard to validate assumptions are made about data distribution. Recent advances in developing efficient machine learning tools such as fast logistic regression [7, 27, 28] and support vector machines [16, 9, 36, 35] enable learning of classifiers in very large predictor spaces, thus reducing the need for pre-processing the data. Nevertheless, most of these techniques are typically designed to exploit the sparsity of the training set (i.e., many features occur sparsely in the training data) which holds for many applications, such as text categorization [27, 9]. This assumption unfortunately does not hold for biological sequences, where many subsequences often occur very frequently.

We present an efficient coordinate-descent algorithm for optimizing regularized fitting of classification loss in high dimensional predictor space. Our approach does not rely on feature sparsity assumptions. In our framework, the feature space is spanned by all subsequences present in the training set. Furthermore, the features can have a flexible number of wildcard matches, which allows us to model complex biological processes such as substitutions, insertions and deletions. The optimization proceeds coordinate-wise by iteratively selecting the feature with maximum (absolute) gradient value, following the Gauss-Southwell rule [30, 37]. In order to select the best features quickly, we provide bounds on the gradient value of any subsequence based on its prefix. This drastically reduces the search space. We discuss the tightness of the proposed bounds as well as the convergence of the algorithm.

Our learning technique is applicable to both unregularized and regularized loss functions. In this paper we show bounds for the more complex case of *elastic-net* regularized loss [10]. By adding an explicit elastic-net penalty (a convex combination of $l1$ and $l2$ regularizers) to the loss function, we allow the user to directly trade-off $l1$-regularization (encouraging model sparsity) for $l2$-regularization (correcting for correlations) [8, 10, 6]. As can be observed in our experiments this positively affects the prediction quality. We present applications of our learning algorithm to protein remote homology detection and fold recognition. In order to compare to previously published results, we work with standard protein benchmarks. For homology detection we use SCOP1.59 [15, 22, 24, 19], and for fold recognition we use the challenging dataset of [3]. The classification problems associated with these datasets are hard and the training data is rather small scale (2,800 and 300 sequences respectively). In order to further analyse the scalability of our technique, we present a large scale experiment on the latest version of the Silva-LSUPARC102 database (150,000 unique sequences). We compare our algorithm to state-of-the-art sequence classifiers, support vector machines with spectrum kernel [24], mismatch kernel [22, 21], and the recent sparse spatial sample kernel [19, 20]. Besides being fast and highly accurate, our classification models are easily interpretable, an important advantage for the bioinformatics and medical communities.

The remainder of this paper proceeds as follows. Section 2 addresses related work. Section 3 presents our learning framework. Section 4 and 5 analyse the experimental setup and results. We conclude with pointers to future work in Section 6.

2. RELATED WORK

Working in high dimensional predictor space and regularizing is statistically preferable to a two-step procedure of first reducing the dimension, then fitting a model in the reduced space [32]. Recently, efficient regularized learning algorithms for logistic regression and support vector machines (SVM) were proposed for fitting classifiers in high dimensional predictor space [27, 9]. In order to scale, most of these techniques restrict the allowed features (i.e., use k-mers, for fixed and reasonably small k) and exploit the feature space sparsity. For some applications, such as text categorization, where the features occur sparsely in the training instances, existing techniques perform very well [12]. However, this is not the case for applications involving biological sequence classification where the feature space is dense (i.e., many subsequences occur in many of the training sequences), thus challenging the running time and memory requirements of existing methods.

Recent work on efficiently computing string kernels for SVM [22, 24, 19, 33, 36] addressed some of the computational challenges associated with this type of techniques. Nevertheless, the kernels proposed still restrict the set of features to subsequences of certain length (e.g, $k = 10$) or format (e.g., allowing some mismatches) in order to scale. In this paper we work with the space of all (unrestricted-length) subsequences in the training set and allow wildcard matches.

A popular class of models used for sequence classification is the generative classifier family, such as profile Hidden Markov Models [4]. Generative models only learn from positive training examples, while discriminative techniques (such as SVM and those proposed in this paper) directly focus on separating the positive and negative training examples. Previous work has shown that discriminative approaches outperform generative approaches for sequence classification [2, 22, 17].

In this paper we propose learning linear classifiers directly in the explicit high dimensional feature space rather than using an implicit mapping via a string kernel as in kernel-SVM. The computational trick in our algorithm is to use coordinate descent coupled with bounding the search for the best coordinate. The idea of using branch-and-bound to efficiently search the feature space has been proposed for graph boosting [18]. Boosting approximately minimizes its l1-regularized loss criterion [32]. Our generic algorithm can be used for *exact* regularized fitting of a range of classification loss functions, such as exponential loss, binomial loglikelihood loss of logistic regression, gaussian loss and the squared hinge-loss of SVM [8]. This work is an extension of [14] which focused on (unregularized) logistic regression applied to text categorization. Here, we extend the bounds of [14] to a class of regularized loss functions, discuss the tightness of the proposed bounds and focus on the aspect of dense feature spaces characteristic of biological applications.

3. METHOD PROPOSED

In this section we present our generic learning algorithm and its concrete implementation for logistic regression and support vector machines.

3.1 Preliminaries and Basic Notation

We first introduce the theoretical framework and some basic notation. Assume we have a training set of instance-label pairs $\{x_i, y_i\}_{i=1}^N$, with $y_i \in \{-1, +1\}$. The training instances x_i are sequences, e.g., biological sequences $x_i = AGTCAACTGGAA....$, text sequences $x_i = ABCD...$, sequences of rankings $x_i = A < B < C < D < \ldots$, sequences of tasks $x_i = ABCD\ldots$. Let d be the number of distinct subsequences in the feature space. We formally represent the training sequences as binary vectors in the space of all subsequences of the training set: $x_i = (x_{i1}, \ldots x_{ij}, \ldots x_{id})^T$, $x_{ij} \in \{0, 1\}$, $i = \overline{1, N}$. Let $\beta = (\beta_1, \ldots, \beta_j, \ldots, \beta_d)$ be a parameter vector (defining a linear classifier).

The goal is to learn a mapping, also called classification model, $f : X \to \{-1, +1\}$ from the given training set such that given a new sample $x \in X$, we can predict a label $y \in \{-1, +1\}$.

Let

$$L(\beta) = \sum_{i=1}^{N} \xi(y_i, x_i, \beta) + CR_\alpha(\beta) \tag{1}$$

be the regularized classification loss criterion. Here, $\xi(y_i, x_i, \beta)$ is a classification loss function, C is a constant and $R_\alpha(\beta)$ is a regularizer. Learning a classification model is achieved by finding the parameter vector β that minimizes the regularized classification loss on the training set

$$\hat{\beta} = \text{argmin}_\beta L(\beta).$$

In Equations (2)–(5) we list a few examples of commonly used classification loss functions.

$$Exponential\ loss : \xi(y_i, x_i, \beta) = e^{-y_i \beta^t x_i} \tag{2}$$

$$Binomial\ loglikelihood\ loss : \xi(y_i, x_i, \beta) = \log(1 + e^{-y_i \beta^t x_i}) \tag{3}$$

$$Squared\ hinge\ loss : \xi(y_i, x_i, \beta) = \max(1 - y_i \beta^t x_i, 0)^2 \tag{4}$$

$$Gaussian\ loss^1 : \xi(y_i, x_i, \beta) = \log \frac{1}{\Phi(y_i \beta^t x_i)} \tag{5}$$

The penalty weight $C \geq 0$ controls the amount of regularization of the parameter vector β. A large C means more

[1] Φ is the cdf of the standard normal distribution.

penalty on the β parameters. The regularization term can take various forms, the most common being the $l1$ and the $l2$ (Equation (6)). In our work $R_\alpha(\beta)$ is the elastic-net regularizer [10, 6] (defined in Equation (7)), which is a compromise between the $l2$ ($\alpha = 0$) and the $l1$ ($\alpha = 1$).

$$l1 = \sum_{j=1}^{d} |\beta_j|, \qquad l2 = \frac{1}{2}\sum_{j=1}^{d} \beta_j^2 \tag{6}$$

$$R_\alpha(\beta) = \alpha \sum_{j=1}^{d} |\beta_j| + (1-\alpha)\frac{1}{2}\sum_{j=1}^{d} \beta_j^2 \tag{7}$$

Depending on the type of regularization, increasing the weight C of the penalty term results in many zero β_j (for $l1$-regularization) or shrinking the coefficients β_j of correlated predictors (for $l2$-regularization). The elastic-net regularizer allows balancing the two effects.

3.2 Gradient-Bounded Coordinate-Descent Algorithm

In this section we describe our learning algorithm and characterize the properties of loss functions to which it can be applied. Assume the following properties for the loss function:

1. ξ depends on y_i, x_i and β only through the classification margin $m_i = y_i\beta^t x_i$. Note that x_i is correctly classified if the margin m_i is positive and the higher the margin the higher the classification confidence. We write $\xi(y, x, \beta) = \xi(m)$.

2. ξ is a monotone decreasing function of the margin: $\xi'(m) \leq 0$.

3. ξ is convex and continuously differentiable.

Properties 1-2 are required for the bounding strategy (Theorem 1), while property 3 is required for the convergence result (Theorem 2). The gradient of $L(\beta)$ (defined in Equation (1)) with respect to a coordinate β_j is

$$\frac{\partial L}{\partial \beta_j}(\beta) = \sum_{i=1}^{N} y_i x_{ij}\xi'(y_i\beta^t x_i) + C[\alpha \text{ sign}(\beta_j) + (1-\alpha)\beta_j]$$

or using the margin notation

$$\frac{\partial L}{\partial \beta_j}(\beta) = \sum_{i=1}^{N} y_i x_{ij}\xi'(m_i) + CR'_\alpha(\beta_j).$$

If $R_\alpha(\beta)$ is not differentiable in $\beta = 0$ we use the left-right derivatives. We proceed minimizing $L(\beta)$ by coordinate-wise gradient descent in the feature space of all subsequences. In each iteration we update only the coordinate corresponding to the subsequence with the largest gradient magnitude, following the Gauss-Southwell rule [30]:

$$j = \text{argmax}_l \left| \frac{\partial L}{\partial \beta_l}(\beta) \right|$$

This results in a greedy advance towards the optimum of the objective function, without having to explicitly generate the full space of subsequences. All we need is an efficient way to find the best coordinate (i.e., feature) in each iteration. We summarize the steps of our generic coordinate-descent algorithm in Algorithm 1. The core part of the algorithm

Algorithm 1 Generic coordinate-descent algorithm

```
1. Set $\beta^{(0)} = 0$
2. For t=1:T
    (a) Find $j_t = \text{argmax}_j \left|\frac{\partial L}{\partial \beta_{j_t}}(\beta^{(t-1)})\right|$ (using Theorem 1)
    (b) Use line search to set the step length $\epsilon_t$ for coordinate $j_t$
    (c) Set $\beta_{j_t}^{(t)} = \beta_{j_t}^{(t-1)} - \epsilon_t \frac{\partial L}{\partial \beta_{j_t}}(\beta^{(t-1)})$ and $\beta_k^{(t)} = \beta_k^{(t-1)}, k \neq j_t$
```

is a search routine (line 2(a) in Algorithm 1) that quickly returns the best feature in each iteration. The search procedure relies on bounding the gradient value of any subsequence early on, by only looking at its prefix. This looks difficult at first glance since the prefix may be a poor feature, while its extension can be highly discriminative. The intuition behind our bound is based on two observations: the gradient is determined by class-wise feature frequency and the subsequence space has structure (that we can exploit to bound the per-class frequency). More concretely, a feature has high gradient value if its frequent in one class and infrequent in the other. Per class, the frequency of a given feature is bounded by the frequency of its prefix. Thus, class-wise, we can bound the gradient of a given subsequence based on its prefix, and then build a global bound from the class-wise bounds. Using this bound we can discard large parts of the search space during the search for the best subsequence, making the whole search process efficient, both time-wise (fast running time) and space-wise (low memory). Once we found the best feature, we do an adapted Armijo rule line search [37, 12] to compute the feature weight and guarantee convergence.

Let j be a coordinate corresponding to a given subsequence s_j, and l be a coordinate corresponding to a super sequence of s_j, s_l, i.e. s_j is a prefix of s_l. We write $s_j \in x_i$ to denote $x_{ij} \neq 0$.

The following theorem gives a *tight* upper bound on the gradient value of any subsequence.

THEOREM 1. *For any loss function ξ satisfying properties 1-2 and any subsequence $s_l \supseteq s_j$, $j = 1, \ldots, d$,*

$$\left|\frac{\partial L}{\partial \beta_l}(\beta)\right| \leq \max \left\{ \left| \sum_{\{i|\mathbf{s_j}\in\mathbf{x_i}, y_i=+1\}} \xi'(m_i) + CR'_\alpha(\beta_l) \right|, \left| \sum_{\{i|\mathbf{s_j}\in\mathbf{x_i}, y_i=-1\}} -\xi'(m_i) + CR'_\alpha(\beta_l) \right| \right\}. \tag{8}$$

PROOF. *We split the analysis in two parts, the first part focuses on deriving a bound within the negative class, and the second part focuses on the positive class. Recall that*

$\xi'(m) \leq 0$. *Then,*

$$\begin{aligned} \frac{\partial L}{\partial \beta_l}(\beta) &= \sum_{\{i|s_l \in x_i\}} y_i x_{il} \xi'(m_i) + CR'_\alpha(\beta_l) \quad (9) \\ &\leq \sum_{\{i|s_l \in x_i, y_i=-1\}} y_i x_{il} \xi'(m_i) + CR'_\alpha(\beta_l) \\ &\leq \sum_{\{i|s_j \in x_i, y_i=-1\}} y_i x_{ij} \xi'(m_i) + CR'_\alpha(\beta_l) \\ &= \sum_{\{i|s_j \in x_i, y_i=-1\}} -\xi'(m_i) + CR'_\alpha(\beta_l) \end{aligned}$$

The last inequality in Equation (9) holds because of the anti-monotonicity property $\{i|s_l \in x_i, y_i = -1\} \subseteq \{i|s_j \in x_i, y_i = -1\}$. *Similarly, we can show for the positive class that*

$$\frac{\partial L}{\partial \beta_l}(\beta) \geq \sum_{\{i|s_j \in x_i, y_i=+1\}} \xi'(m_i) + CR'_\alpha(\beta_l)$$

Thus we obtain the lower and upper bounds

$$\begin{aligned} \sum_{\{i|s_j \in x_i, y_i=+1\}} \xi'(m_i) + CR'_\alpha(\beta_l) \leq \frac{\partial L}{\partial \beta_l}(\beta) \quad (10) \\ \leq \sum_{\{i|s_j \in x_i, y_i=-1\}} -\xi'(m_i) + CR'_\alpha(\beta_l). \end{aligned}$$

We are interested in an upper bound on the absolute magnitude of the gradient at a given coordinate, thus we write more conveniently

$$\left|\frac{\partial L}{\partial \beta_l}(\beta)\right| \leq \max \left\{ \left| \sum_{\{i|s_j \in x_i, y_i=+1\}} \xi'(m_i) + CR'_\alpha(\beta_l) \right|, \left| \sum_{\{i|s_j \in x_i, y_i=-1\}} -\xi'(m_i) + CR'_\alpha(\beta_l) \right| \right\}.$$

□

The main property on which Theorem 1 relies on (last inequality in Equation (9)) is *anti-monotonicity*: the prefix frequency is higher than that of its extension subsequence. This means that the same bound holds for *any* weight x_j as long as $x_j \geq x_l$. This observation is useful for example if we want to integrate prior biological knowledge about the target problem, such as the fact that some amino acids in protein sequences are more useful than others. We could encode this information by giving weights $w_a \in (0, 1)$ to individual amino acids and still have the property that $x_j = \Pi_a w_a \geq x_l$.

As we can observe from Equation (8), without explicit regularization of the loss function, the upper bound on the gradient of a subsequence s_l solely depends on the frequency of its prefix s_j. This means that we can decide not to expand the s_j prefix further, without inspecting the longer feature s_l. If we use a regularizer, we have the term β_l in the bound which depends on the longer subsequence, rather than the prefix alone. Thus, we have to take care to compute the correct bound whenever $\beta_l \neq 0$. Since at start all β_j are zero, during the iterations we only need to check those features that have non-zero β_l because they were selected in the model in previous iterations. In practice, as supported by our experiments, this part is fairly fast due to the sparsity of the final model learned by this algorithm.

The bound presented in Theorem 1 is tight since we can construct examples for which the inequality is an equality. Consider the simple case of a training set with positive examples of the type *AAAAAAA*, and negative examples of the type *BBBBBB*. Whenever the subsequence set of occurrences is the same as that of its prefix, the inequality becomes equality.

The iterates generated by Algorithm 1 converge to the optimal solution of the objective function given in Equation (1). Based on results from [37] characterizing the convergence of iterates generated by coordinate-descent using the Gauss-Southwell rule for l1-regularized convex minimization we have:

Theorem 2. *Let* $\beta^{(t)}$ *be a sequence of iterates generated by Algorithm 1. Then* $\beta^{(t)}$ *converges to the optimal solution of (1).*

In the next subsections we describe the specific bounds used to design learning algorithms in the style of Algorithm 1 for logistic regression and support vector machines. We choose these popular classifiers for concrete implementations because their loss functions satisfy the properties 1-3 and they are margin maximizing [32]. The latter property typically translates into good generalization ability in practice.

3.3 Logistic Regression

Let

$$L(\beta) = \sum_{i=1}^{N} \log(1 + e^{-y_i \beta^t x_i}) + CR_\alpha(\beta)$$

be the elastic-net-regularized binomial log-likelihood loss [8]. The gradient of $L(\beta)$ with respect to a coordinate j at a given parameter vector β is:

$$\frac{\partial L}{\partial \beta_j}(\beta) = \sum_{i=1}^{N} y_i x_{ij} \underbrace{\left(\frac{-1}{1 + e^{y_i \beta^t x_i}}\right)}_{\xi'(m_i)} + CR'_\alpha(\beta_l)$$

Corollary 1. *For binomial log-likelihood loss and any subsequence* $s_l \supseteq s_j$, $j = 1, \ldots, d$,

$$\left|\frac{\partial L}{\partial \beta_l}(\beta)\right| \leq \max \left\{ \left| \sum_{\{i|s_j \in x_i, y_i=+1\}} \frac{-1}{1 + e^{\beta^t x_i}} + CR'_\alpha(\beta_l) \right|, \left| \sum_{\{i|s_j \in x_i, y_i=-1\}} \frac{1}{1 + e^{-\beta^t x_i}} + CR'_\alpha(\beta_l) \right| \right\}.$$

3.4 Support Vector Machines

Let

$$L(\beta) = \sum_{i=1}^{N} \max(1 - y_i \beta^t x_i, 0)^2 + CR_\alpha(\beta)$$

be the elastic-net-regularized squared hinge loss [1]. We can rewrite $L(\beta)$ in the equivalent form

$$L(\beta) = \sum_{\{i|1 - y_i \beta^t x_i > 0\}} (1 - y_i \beta^t x_i)^2 + CR_\alpha(\beta)$$

The gradient of $L(\beta)$ with respect to a coordinate j at a given parameter vector β is:

$$\frac{\partial L}{\partial \beta_j}(\beta) = \sum_{\{i|1 - y_i \beta^t x_i > 0\}} y_i x_{ij} \underbrace{2(y_i \beta^t x_i - 1)}_{\xi'(m_i)} + CR'_\alpha(\beta_l)$$

Corollary 2. *For squared hinge loss and any subsequence $s_l \supseteq s_j$, $j = 1, \ldots, d$,*

$$\left|\frac{\partial L}{\partial \beta_l}(\beta)\right| \leq \max\left\{\left|\sum_{\{i|s_j \in x_i, 1-\beta^t x_i > 0, y_i = +1\}} 2(\beta^t x_i - 1) + CR'_\alpha(\beta_l)\right|,\right.$$
$$\left.\left|\sum_{\{i|s_j \in x_i, 1+\beta^t x_i > 0, y_i = -1\}} 2(1 + \beta^t x_i) + CR'_\alpha(\beta_l)\right|\right\}.$$

3.5 Algorithmic Details

In this section we give some details on using the bound of Theorem 1 to prune the search space.

Before starting the optimization iterations we build an inverted index on all the distinct unigrams in the training set, i.e., a list of document ids and positions of occurrence for each unigram. In each iteration we start the process of searching for the best feature from the level of unigrams.

We implement two strategies of prefix-expansion. The breadth-first-search (BFS) expands all unigrams to bi-grams, then all bi-grams to tri-grams, etc. For each unigram, we compute the gradient value, we keep track of the best gradient/feature seen so far and we compare the bound to the current best gradient. If the bound is lower than the current best gradient value, no subsequence starting with this unigram can improve the current optimum and we prune this part of the search space (i.e., we discard this prefix). Then we move on to the next level of expansion, and repeat this process. The depth-first-search (DFS) strategy expands the unigram to its longest subsequence until the pruning condition is met, and it then backtracks to the longest valid prefix and re-attempts to expand. Depending on the sequence tokenization used, word-level or character-level, the two strategies have different benefits. For word-level tokenization, such as in text categorization, short subsequences are more likely to be useful than long ones (e.g., phrases of 2-3 words are typically good discriminators), and therefore BFS may be a better expansion strategy. For character-level tokenization, such as for biological sequences, longer subsequences tend to be more useful than shorter ones (e.g., subsequences of length 3 or less may occur in all the sequences), so DFS is a better expansion choice. For the experiments in this paper we have used DFS. In order to keep track of the occurrences of all active prefixes (e.g., subsequences that could not be pruned using Theorem 1), we expand the inverted index on-demand. The inverted index does not grow excessively due to the effectiveness of the bound. After selecting the feature with the best gradient in a given iteration, we do a line search to compute the final feature weight [12]. We stop the optimization iterations based on a threshold on the change of predicted scores [12].

The worst-case complexity of each iteration is $O(dN)$, where d is the number of features and N is the number of training samples. In practice, our algorithm drastically prunes the search space and is thus very fast. For example, for the classification task *Scorpion-toxin like* (presented in detail in Section 5) we have observed the following behaviour. There are 23 distinct unigrams in the training set (22 amino acids and we add the wildcard as an additional unigram). The longest sequence in the training set has length 892, and the average sequence lenght is ~ 50. Even if we were to restrict the maximum k-mer size to $k = 50$, the number of features d is $O(23^{50})$ (all the subsequences up to size $k = 50$ created using the 23 unigrams; in practice, without wildcards, there are 6.5 million features in the training set). For finding the best feature in the first iteration using SVM loss and unrestricted k, our algorithm checks the pruning bound 1,113 times, and prunes the space 1,104 times (i.e., discards 1,104 prefixes out of the total 1,113 analysed). The algorithm stops in 3 seconds, after 77 iterations. In the last iteration, it checks the bound 14,077 times and prunes the space 13,952 times. Thus, the bound in Theorem 1 efficiently prunes the huge search space. For future work we will focus on quantitative measures of index growth as a function of the alphabet size and number of training examples.

For simplifying the implementation we currently prune the space using a prefix-expansion strategy. Theorem 1 however can be used for pruning all features that contain a given subsequence, not necessarily in the start position as a prefix. This observation could be used to further speed up the training process.

4. EXPERIMENTS

We implement the generic coordinate-descent algorithm in our machine learning tool SEQL (**SEQ**uence **L**earner), available from `http://www.daimi.au.dk/~ifrim/seql` (v2.0). For now, SEQL implements the two elastic-net-regularized classifiers presented in the previous section: logistic regression and support vector machines. In this section we describe the datasets, the techniques compared and the methodology for designing experiments.

4.1 Datasets

In order to compare our results to the state-of-the-art we compare directly to published results and perform experiments on the same benchmarks.

For **protein remote homology detection** we use the benchmark SCOP1.59 [22, 24, 38, 19]. This is an expert-curated database of protein domains organized heirarchically into folds, superfamilies and families. Protein sequences belonging to different families, but the same superfamily, are considered to be remote homologs in SCOP. This dataset contains 2,862 labeled sequences organized into 54 binary classification problems simulating homology detection by predicting the super-family using a hold-one-family-out strategy. No pair of sequences shares more than 95% identity. The positive training sets are quite small and the learning task is challenging. For more details on this benchmark see [15, 24, 38].

For **protein remote fold recognition** we use the benchmark published by [3]. This dataset consists of sequences from 27 folds divided into two independent sets such that the training and test sequences share less than 35% sequence identities and within the training set, no sequences share more than 40% sequence identities. There are 311 training sequences and 383 test sequences.

In order to analyze the **scalability** of our method in a large scale experiment, we have downloaded the latest **ribosomal RNA** (rRNA) database Silva-LSUParc[2] [31], version 102 (released in February 2010.) The LSUParc102 database contains 180,344 rRNA sequences. After removing duplicate sequences we obtain a dataset of 150,780 unique sequences organized in 3 (one-vs-all) binary classification tasks

[2]Silva-LSUParc database: http://www.arb-silva.de/documentation/background/release-102/.

according to the Bacteria, Archaea and Eukarya domains. The distribution by domain is dominated by Eukarya with 141,601 sequences, followed by Bacteria with 8,967 and Archaea with 212 sequences. All datasets are available from `http://www.daimi.au.dk/~ifrim/seql/data`.

4.2 Techniques Compared

Previous studies have shown that discriminative approaches for sequence classification (such as kernel-SVM) outperform generative approaches (such as profile HMM) by a large margin [2, 17, 22, 26].

We compare our algorithms to the latest sequence kernels for SVM: the spectrum kernel, the mismatch kernel [22, 24, 29], and the recent sparse spatial sample kernel (SSSK) [20, 19]. All these methods aim at computing similarity for all pairs of sequences (the kernel matrix) in a particular feature space. Due to computational challenges, these techniques typically restrict the length and expressive power of the subsequences used as features. For example, the spectrum-k kernel [23] implicitly compares sequences in the space of all k-mers, where the length k of subsequence-features is a parameter of the model. The mismatch kernel [22] generalizes the spectrum-k kernel by allowing up to m mismatches or substitutions to accomodate mutations. The sparse spatial sample kernel (SSSK) further generalizes the mismatch kernel by sampling the sequences at different resolutions and comparing the resulting spectra [19]. SSSK has 3 parameters, (k, t, d), where k is the probe size, t is the number of probes and d is the number of *maximum* allowed positions between the probes. Our learning algorithm SEQL uses all (unrestricted-length) subsequences in the training set as features. Furthermore, we also allow mismatches or so called wildcard matches, by a parameter that controls the *maximum* number of *consecutive* wildcards allowed. This allows us to model complex biological processes such as substitutions, insertions and deletions. Figure 1 gives examples of the types of features implicitly used by the above described kernels, as compared to the features used by our technique.

One advantage of kernel techniques is their ability to integrate unlabeled data during kernel computation to relax labeled data requirements [19]. However, computing all-pair similarities for large datasets (resulting from the addition of unlabeled sequences) remains computationally very challenging for kernel methods both time and memory-wise. For example, we couldn't apply any of the above mentioned kernel methods on our large dataset (150,000 sequences) since this would require more than 90GByte memory. In contrast, by exploiting the feature space structure, SEQL can use more expresive features and still be able to scale.

For completion we also report the results of a technique published by [26] which applies SVM using an empirical kernel map based on pairwise Smith-Waterman sequence alignment scores (SVM-pairwise in Table1).

We use SEQL-LR and SEQL-SVM to denote the logistic regression and support vector machines implementations of our generic learning algorithm.

4.3 Methodology

Previous studies on biological sequence classification have extensively used the AUC (area under the ROC curve) and AUC50 for evaluation [5, 24, 34]. AUC describes the ranking of prediction scores rather then being dependent on a fixed classification threshold. The ROC curve is obtained by plotting the fraction of true positives versus the fraction of false positives for a binary classifier as its discrimination threshold is varied [5]. A common aggregate measure is to report the area under the ROC curve (AUC) where an area of 1 represents a perfect ranking of all positives above all negatives and an area close to 0.5 represents a random classifier. The AUC50 focuses on top ranked examples and is defined as the normalized area under the ROC curve computed for up to 50 true negatives [34]. It is typically used to evaluate classifiers on datasets where the number of positives is much lower than the number of negatives. While for a complete test set the AUC is betwen 0.5 and 1, the AUC values for truncated top lists are between 0 and 1. The shorter the top list is, the smaller the AUC values [34]. In some studies instead of AUC or AUC50 only the balanced-error-rate (BER) is reported. To compare our results to the state-of-the-art, we also show the balanced-error (Equation (11)) which measures the average of the errors on each class for a fixed classification threshold. It can equivalently be interpreted in terms of Specificity and Sensitivity [25].

$$BER = \frac{1}{2}\left(\underbrace{\frac{FN}{TP+FN}}_{1-\text{Sensitivity}} + \underbrace{\frac{FP}{FP+TN}}_{1-\text{Specificity}}\right) \qquad (11)$$

The benchmarks for remote homology detection and fold recognition have pre-defined training and test splits. We use the same data for experiments for all methods compared, and for the prior techniques *we report directly the results published in the original papers.* For the large scale experiment on ribosomal RNA, we show 5-fold cross-validation results on the full dataset. Since the prior techniques required more than the available memory, for the large dataset we only report results using our method.

All experiments were run on a Linux laptop with 2.5GByte memory and 2.8GHz Intel CPU.

5. RESULTS AND DISCUSSION

In this section we present the results of applying SEQL-LR and SEQL-SVM to protein remote homology detection and fold recognition. Furthermore, to evaluate the scalability of our approach, we present a large-scale experiment on the latest release of the Silva-LSUParc database [31].

5.1 Comparison to State-of-the-Art

We first compare the performance of SEQL-LR and SEQL-SVM to that of kernel-SVM techniques on two protein classification tasks.

5.1.1 *Protein remote homology detection*

In Table 1 we show results on the SCOP1.59 benchmark for all compared methods. The numbers in brackets next to the name of each method refer to their explicit parameters as discussed in the previous section (e.g., length k of subsequences used as features or maximum number of wildcards). The results are averaged over all 54 binary classification tasks which correspond to superfamilies.

For SEQL-LR and SEQL-SVM we do grid search to set d (the maximum allowed number of consecutive wildcards, $d \in \{0, 1, 2, .., 5\}$), and the regularization parameters C and α of the elastic-net penalty ($C \in \{0, 0.001, 0.01, .., 100\}$, $\alpha \in \{0, 0.1, 0.2, .., 1\}$). We show the most accurate results across

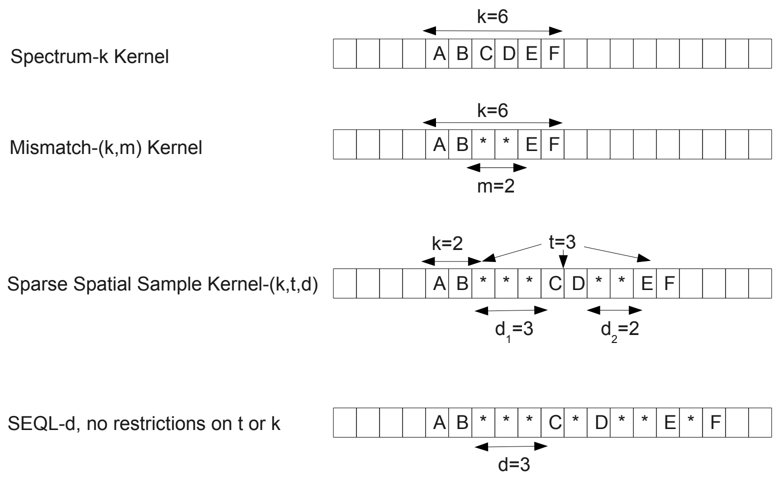

Figure 1: Features employed by state-of-the-art kernels versus our method SEQL**. Spectrum-k Kernel, k is the length of features. Mismatch-(k,m) Kernel, k is the length of features, m is the maximum number of mismatches. Sparse Spatial Sample-(k,t,d) Kernel, k is the probe size, t is the number of probes and d is the number of maximum allowed positions between the probes.** SEQL**-d, d is the maximum number of consecutive wildcards.**

Method	AUC	AUC50	BER
SVM-SPECTRUM(2)	0.8581	0.3583	-
SVM-SPECTRUM(3)	0.8723	0.4037	-
SVM-MISMATCH(5,1)	0.8749	0.4167	-
SVM-SSSK(1,2,5)	0.8901	0.4629	-
SVM-SSSK(1,3,3)	0.9148	0.5118	-
SVM-pairwise	0.8930	0.4340	-
SEQL-LR(5)	**0.9155**	**0.5184**	**0.4477**
SEQL-SVM(5)	**0.9220**	**0.5237**	**0.4158**

Table 1: Remote homology detection on the Scop1.59 dataset. The average AUC, AUC50 and BER scores over the 54 target superfamilies. Results for kernel-SVM methods cited directly from [24, 19]. SEQL-LR **and** SEQL-SVM **are the methods proposed in this paper.**

the parameter values tried. We observe that our techniques are comparable to the state-of-the-art methods regarding classification quality. We believe it is important to allow flexible features, (e.g., allowing wildcards and not restricting the feature lenght in advance), although here the training sets are quite small and the gain is not large. We note that SEQL-SVM has slightly better AUC and AUC50 than SEQL-LR. Learning SEQL classifiers on this dataset took a few seconds per topic and about 80MByte memory. The computational requirements of SEQL are mainly influenced by the maximum allowed number of consecutive wildcards.

Figure 2 shows a graphic comparison of some of the methods (for which we had access to published per class AUC50 scores) by plotting the total number of superfamilies above an AUC50 score threshold, as a function of the threshold value. The threshold is changed so as to generate the next AUC50 value in the ranked list of AUC50 scores of the respective method. The numerical integral of this cumulative

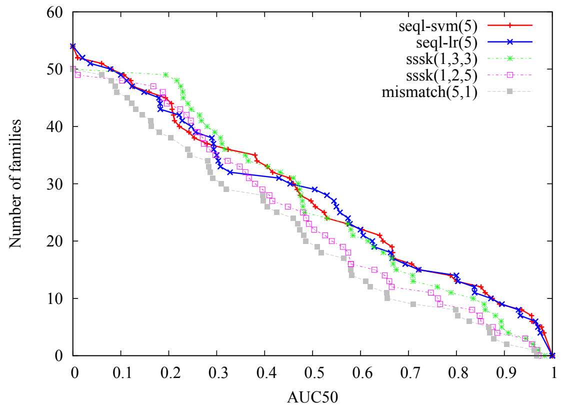

Figure 2: Comparison of mismatch-SVM, SSSK-SVM, SEQL-LR and SEQL-SVM on SCOP1.59. The total number of superfamilies for which a given method exceeds an AUC50 score threshold.

AUC50 curve is the arithmetic average of the AUC50 values shown in Table 1 [34]. SVM with sparse-sample-spatial kernel (SSSK-SVM) behaves best among the prior techniques. Even though the features employed by SSSK-SVM are quite flexible (see Figure 1), we observe we can gain some performance by imposing less restrictions on features.

[19] analyzed the biological relevance of features learned by SSSK-SVM taking the *Scorpion toxin-like* superfamily as an example. This classification task has 16 members of this superfamily as positive training examples, 1067 non-members as negative examples and the *short-chain scorpion toxin* family as a test case. SSSK-SVM(1,2,5) achieved an AUC50 of 0.7661 for this task [19]. In their work [19] present the schematic representation of the *short-chain scorpion toxin*

family obtained from PROSITE [11]. The PROSITE database consists of a large collection of manually-curated biologically meaningful signatures that are described as patterns or profiles [11]. In Figure 3 we show the PROSITE representation. The scheme shows that these type of toxins contain six

```
       +---------------------+
       |                     |
       |     xxxxxxxxxxxxxxxxxxxxxxxx
*******C*****C***C***********C****C*C***
             |   |                | |
             |   +----------------|-+
             +--------------------+

'C': conserved cysteine involved in a disulfide bond.
'x': position of the consensus pattern
```

Figure 3: Schematic representation of the *short-chain scorpion toxins* family from PROSITE.

conserved cysteines (C) residues involved in disulfide bonds. PROSITE also lists a consensus pattern present in all the members of this family shown as well in Figure 3 (marked with 'X'). We focus here on the same superfamily and look at the features learned by SEQL-SVM(5). In Table 2 we show

Positive Features	
Weight	Feature
0.0460	SG*C
0.0450	N**C***C
0.0415	C*C**L
0.0410	C***C***G
0.0340	C*C
0.0329	G**G*C
0.0326	C*****C***C
0.0308	GYC
0.0308	C***C
0.0307	G***G*C

Negative Features	
Weight	Feature
-0.0329	G
-0.0313	A
-0.0292	S
-0.0279	Q
-0.0271	E
-0.0248	A*****K
-0.0217	K
-0.0216	CP
-0.0206	C**G
-0.0205	C**C

Table 2: Top-10 positive and negative SEQL-SVM features on the SCOP.1.59 Scorpion-toxin-like superfamily.

the top-10 positive and top-10 negative features selected by SEQL-SVM on this superfamily.

SEQL-SVM(5) has an AUC50 of 0.9573 on this task. [19] analyzed the support vectors of their method and observed that the feature C***C had the highest weight. Similarly, SEQL-SVM(5) selects the feature C***C in top-10, but ranks features such as SG*C and N**C***C higher. In order to understand the effect of these features on the AUC50 score, we took a closer look at the quality of some of the selected features. We observed that although C***C occurs in all positive examples, it is also present in many negatives and is thus less useful than SG*C, which occurs in most positive examples, but in very few negative examples.

5.1.2 Protein remote fold recognition

In this section we show an application of SEQL-LR and SEQL-SVM to protein remote fold recognition on the dataset published by [3]. They proposed a technique based on building an SVM classifier using features deemed biologically relevant for this problem, such as percentage composition of the 20 amino acids, predicted secondary structure, polarity. In Table 3 we show results comparing the technique of [3], denoted by SVM(D&D), with published results of kernel-SVM methods and with our SEQL-LR and SEQL-SVM algorithms. Recall that lower BER means better classification quality.

Method	AUC	AUC50	BER
SVM (D&D)	-	-	0.5650
SVM-MISMATCH(5,1)	-	-	0.5322
SVM-SSSK(1,2,5)	-	-	0.4619
SVM-SSSK(1,3,3)	-	-	0.4499
SEQL-LR(5)	**0.7882**	**0.3551**	**0.4412**
SEQL-SVM(5)	**0.7982**	**0.3742**	**0.4222**

Table 3: Remote fold recognition on the D&D dataset. The average AUC, AUC50 and BER over the 27 target folds. Results for SVM(D&D) and other kernel-SVM methods cited from [3, 19].

Even if we don't use any domain-specific knowledge in this experiment, we observe that our techniques have better classification quality than the method of [3] which relies on deep biological insight. We believe this advantage results from having feature selection as a part of the learning algorithm and allowing for a large degree of flexibility in the features. SEQL-LR and SEQL-SVM outperform the state-of-the-art techniques in terms of balanced-error-rate on this benchmark.

5.2 Large-Scale Experiment

In this section we analyze the scalability of our method on a ribosomal RNA (rRNA) domain-prediction task.

5.2.1 Ribosomal RNA Domain-Prediction

The dataset used for this task contains 150,780 unique rRNA sequences. It is estimated that based on the new capacity for cheap and rapid sequencing there is a steady flow of about 10,000 rRNA sequences per month into the public sequence databases [31]. Furthermore, many sequences are derived from cultivation independent biodiversity surveys, which rely on rapid pattern- or clone-based approaches that often generate partial rRNA sequences [31] (problematic for sequence alignment).

We show the average results of SEQL-LR and SEQL-SVM over 5-fold cross-validation splits for fixed parameter values. We set the amount of regularization $C = 1.0$, we balance $l1$ and $l2$ regularization by setting $\alpha = 0.5$ and allow no wildcards by setting the maximum number of consecutive wildcards $d = 0$.

In Table 4 we show classification quality, training time and memory resources required by SEQL-LR and SEQL-SVM. We

Method	AUC	AUC50	BER	Time	Memory
SEQL-LR	0.9992	0.9745	0.0095	30 min	2GByte
SEQL-SVM	0.9986	0.9744	0.0118	25 min	2GByte

Table 4: Ribosomal RNA domain-prediction on Silva-LSUParc102.

note that both SEQL-based techniques have high classification quality. Additionally, they only take about half-an-hour running-time and a reasonable 2GByte memory instead of 90GByte required by a kernel matrix computation. Instead of building sequence classifiers by costly multiple sequence alignment as currently done for sequence retrieval in this database, we could directly learn domain SEQL-classifiers from the original input sequences in the respective domains.

6. CONCLUSION

In this paper we present a new learning algorithm for sequence classification in high dimensional predictor space. Our framework has at its core a gradient-bounded coordinate-descent strategy to quickly retrieve high quality features. The generic learning algorithm works with a wide range of loss functions, including that of logistic regression and support vector machines. When applied to protein remote homology detection and fold recognition, as well as large-scale domain-prediction for ribosomal RNA, our techniques are comparable to the state-of-the-art in terms of classification quality. In addition, the techniques presented are highly scalable and the resulting classification models can easily be interpreted and connected to biologically relevant facts. Furthermore, the same gradient-bounding strategy can be used for tackling sequence regression [13]. Finally, our algorithm can be applied to more complex structured data such as trees and graphs which follow the anti-monotonicity property required by our theoretical framework. We plan to investigate these research directions in our future work.

7. ACKNOWLEDGMENTS

This work was supported by the Danish Cancer Society and Science Foundation Ireland Grant 10/IN.1/I303.

8. REFERENCES

[1] K.-W. Chang, C.-J. Hsieh, and C.-J. Lin. Coordinate descent method for large-scale l2-loss linear support vector machines. *J. Mach. Learn. Res.*, 9:1369–1398, 2008.

[2] J. Cheng and P. Baldi. A machine learning information retrieval approach to protein fold recognition. *Bioinformatics*, 22(12):1456–1463, 2006.

[3] C. H. Ding and I. Dubchak. Multi-class protein fold recognition using support vector machines and neural networks . *Bioinformatics*, 17(4):349–358, 2001.

[4] S. R. Eddy. Profile hidden markov models. *Bioinformatics*, 14(9):755–763, 1998.

[5] T. Fawcett. Roc graphs: Notes and practical considerations for researchers, 2004.

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

[7] A. Genkin, D. Lewis, , and D. Madigan. Large-scale bayesian logistic regression for text categorization. *Technometrics*, 49(3):291–304, 2007.

[8] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics, 2003.

[9] C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *ICML*, 2008.

[10] Z. Hui and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Stat. Soc. Series*, 67(2):301–320, 2005.

[11] N. Hulo, A. Bairoch, V. Bulliard, L. Cerutti, E. De Castro, P. S. Langendijk-Genevaux, M. Pagni, and C. J. A. Sigrist. The prosite database. *Nucl. Acids Res.*, 34(1):227–230, 2006.

[12] G. Ifrim. *Statistical Learning Techniques for Text Categorization with Sparse Labeled Data*. PhD thesis, Universität des Saarlandes, 2009.

[13] G. Ifrim. Gradient-bounded coordinate-descent for sequence regression. www.4c.ucc.ie/ ifrim/seql/data, 2010.

[14] G. Ifrim, G. Bakir, and G. Weikum. Fast logistic regression for text categorization with variable-length n-grams. In *KDD*, pages 354–362, USA, 2008.

[15] T. Jaakkola, M. Diekhans, and D. Haussler. Using the fisher kernel method to detect remote protein homologies. In *ICISMB*, pages 149–158. AAAI Press, 1999.

[16] T. Joachims. Training linear SVMs in linear time. In *KDD*, pages 217–226, USA, 2006. ACM Press.

[17] R. Kuang, E. Ie, K. Wang, K. Wang, M. Siddiqi, Y. Freund, and C. Leslie. Profile-based string kernels for remote homology detection and motif extraction. In *CSBC*, pages 152–160, USA, 2004. IEEE Computer Society.

[18] T. Kudo, E. Maeda and Y. Matsumoto. An Application of Boosting to Graph Classification. In *NIPS*, 2004.

[19] P. Kuksa, P. H. Huang, and V. Pavlovic. A fast, semi-supervised learning method for protein sequence classification. In *International Workshop on Data Mining in Bioinformatics*, pages 29–37, 2008.

[20] P. Kuksa, P. H. Huang, and V. Pavlovic. Scalable algorithms for string kernels with inexact matching. In *NIPS*, 2008.

[21] C. Leslie, E. Eskin, A. Cohen, J. Weston, and W. S. Noble. Mismatch string kernels for discriminative protein classification. *Bioinformatics*, 20(4):467–476, 2004.

[22] C. Leslie, E. Eskin, and W. Noble. Mismatch string kernels for svm protein classification. In *NIPS*, 2002.

[23] C. Leslie, E. Eskin, and W. Noble. The spectrum kernel: a string kernel for svm protein classification. In *Pacific Biocomputing Symposium*, pages 564–575, 2002.

[24] C. Leslie and R. Kuang. Fast string kernels using inexact matching for protein sequences. *J. Mach. Learn. Res.*, 5:1435–1455, 2004.

[25] I. Levner, V. Bulitko, and G. Lin. Feature extraction for classification of proteomic mass spectra: A comparative study. *Studies in Fuzziness and Soft Computing*, 207:607–624, 2006.

[26] L. Liao and W. S. Noble. Combining pairwise sequence similarity and support vector machines for remote protein homology detection. In *ICCB*, pages 225–232, USA, 2002.

[27] C.-J. Lin, R. C. Weng, and S. S. Keerthi. Trust region Newton method for large-scale logistic regression. *J. Mach. Learn. Res.*, 9:627–650, 2008.

[28] J. Liu, J. Chen, and J. Ye. Large-scale sparse logistic regression. In *KDD*, pages 547–556, USA, 2009.

[29] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. *J. Mach. Learn. Res.*, 2:419–444, 2002.

[30] D. Luenberger. *Linear and Nonlinear Programming.* Addison-Wesley Publishing Co.: Reading Mass, 1984.

[31] E. Pruesse, C. Quast, K. Knittel, B. M. Fuchs, W. Ludwig, J. Peplies, and F. O. Glockner. SILVA: a comprehensive online resource for quality checked and aligned ribosomal RNA sequence data compatible with ARB. *Nucl. Acids Res.*, 35(21):7188–7196, 2007.

[32] S. Rosset, J. Zhu, and T. Hastie. Boosting as a regularized path to a maximum margin classifier. *J. Mach. Learn. Res.*, 5:941–973, 2004.

[33] Q. Shi, J. Petterson, G. Dror, J. Langford, A. Smola, and S. Vishwanathan. Hash kernels for structured data. *J. Mach. Learn. Res.*, 10:2615–2637, 2009.

[34] P. Sonego, A. Kocsor, and S. Pongor. ROC analysis: applications to the classification of biological sequences and 3D structures. *Brief. in Bioinf.*, 9(3):198–209, 2008.

[35] S. Sonnenburg and V. Franc. Coffin : A computational framework for linear svms. In *ICML*, 2010.

[36] S. Sonnenburg, G. Rätsch, S. Henschel, C. Widmer, J. Behr, A. Zien, F. de Bona, A. Binder, C. Gehl, and V. Franc. The SHOGUN machine learning toolbox. *J. Mach. Learn. Res.*, 11:1799–1802, 2010.

[37] P. Tseng and S. Yun. A coordinate gradient descent method for nonsmooth separable minimization. *Mathematical Programming*, 117:387–423, 2009.

[38] J. Weston, C. Leslie, E. Ie, D. Zhou, A. Elisseeff, and W. S. Noble. Semi-supervised protein classification using cluster kernels. *Bioinformatics*, 21(15):3241–3247, 2005.