Efficient Sequence Regression by Learning Linear Models in All-Subsequence Space

Severin Gsponer(✉), Barry Smyth, and Georgiana Ifrim

Insight Centre for Data Analytics, University College Dublin, Dublin, Ireland
{severin.gsponer,barry.smyth,georgiana.ifrim}@insight-centre.org

Abstract. We present a new approach for learning a sequence regression function, i.e., a mapping from sequential observations to a numeric score. Our learning algorithm employs coordinate gradient descent with Gauss-Southwell optimization in the feature space of all subsequences. We give a tight upper bound for the coordinate wise gradients of squared error loss which enables efficient Gauss-Southwell selection. The proposed bound is built by separating the positive and the negative gradients of the loss function and exploits the structure of the feature space. Extensive experiments on simulated as well as real-world sequence regression benchmarks show that the bound is effective and our proposed learning algorithm is efficient and accurate. The resulting linear regression model provides the user with a list of the most predictive features selected during the learning stage, adding to the interpretability of the method. Code and data related to this chapter are available at: https://github.com/svgsponer/SqLoss.

1 Introduction

A wide range of applications benefit from methods that can learn a mapping from sequential observations to categorical or numeric scores. For example, a mapping could be learned from a set of labeled DNA sequences, to classify each sequence into subfamilies [10], or assign to it a numeric score, such as a protein binding affinity [18]. Methods aimed at solving such problems typically employ Hidden Markov Models (HMM) [14,19], kernel Support Vector Machines (SVM) [11] or more recently, Convolutional Neural Networks (CNN) [2]. While the accuracy of such techniques is promising, their efficiency and interpretability are still critical challenges. An alternative to the above approaches is to explicitly generate all k-mers (i.e., subsequences of length k) up to a specified k, followed by learning a classification or regression model using all the generated k-mers as features. Such methods are much simpler and achieve an accuracy comparable to the more sophisticated methods above [3,18]. Nevertheless, they are limited by the huge computational burden of explicitly generating all k-mers, and therefore tend to be applied to small datasets with k fixed manually, e.g., up to 6 or 8, and need to use additional filtering to further reduce the large feature space.

In this work, we propose a regression approach that can use the entire space of k-mers, of unlimited length, by learning a linear model using an iterative

© Springer International Publishing AG 2017
M. Ceci et al. (Eds.): ECML PKDD 2017, Part II, LNAI 10535, pp. 37–52, 2017.
https://doi.org/10.1007/978-3-319-71246-8_3

branch-and-bound strategy. The main idea behind this technique is to exploit the nested structure of the feature space via greedy search, thus avoiding the need for explicitly generating all subsequences, and instead focusing on the most discriminative ones during learning. The resulting approach combines feature selection and learning into a simple algorithm and, as shown in our experiments, delivers accuracy similar to the state-of-the-art, with no pre-processing or domain knowledge required. Since during learning we only need to explore a small subset of the feature space, we can employ richer features such as gapped k-mers to allow inexact feature matches. This enables our linear models to achieve high accuracy. Our optimization algorithm relies on greedy coordinate-descent with Gauss-Southwell selection. To enable efficient coordinate selection, we give a tight upper bound for the coordinate-wise gradients of squared error loss.

We test the proposed algorithm on a simulated benchmark and on two real-world applications. First, we compare our algorithm to other regression methods on a synthetic sequence regression dataset, where we vary parameters such as the true motif length and the alphabet size, to study potential gains from using rich features such as all k-mers. Next, we apply our model to a sequence regression problem where the goal is to score the protein binding affinity of DNA sequences. We work with a publicly available dataset of 40,000 DNA sequences prepared by [18] for a popular data challenge[1]. Finally, we study our algorithm on a large sequence classification dataset to compare the effectiveness of our approach to existing methods designed for linear sequence classification [7,8]. In this application the aim is to score software files represented as hexadecimal sequences, in order to categorize malicious software into known families, also known as malware classification [1]. This dataset was released by a recent data challenge[2] organized by Microsoft.

Contribution. We propose a new method for efficient sequence regression by learning linear models with rich subsequence features, e.g., unrestricted-length, contiguous and gapped k-mers. Our algorithm uses an optimization strategy based on coordinate-descent coupled with an upper bound on the coordinate-wise gradients of squared error loss, to enable efficient Gauss-Southwell selection. We evaluate our learning algorithm on simulated data and on two real-world applications and show that our simple linear models are as accurate as more complex state-of-the-art sequence regression methods, while requiring no feature engineering or heavy parameter tuning. We release all our code for producing synthetic sequence regression data, as well as for our learning algorithm[3].

2 Related Work

We discuss a range of approaches for sequence regression and classification, with a focus on the two application domains studied in this paper.

[1] https://www.synapse.org/#!Synapse:syn2887863/wiki/72186.
[2] https://www.kaggle.com/c/malware-classification.
[3] Code of our algorithms: https://github.com/svgsponer/SqLoss.

Sequence Regression for DNA. The sequence regression benchmark provided by the DREAM5 challenge aims to advance the state-of-the-art in recognizing DNA-binding proteins. It consists of 40,000 DNA training sequences, each with a numeric score describing the binding specificity of a particular protein, from a class of proteins named Transcription Factors (TF). The task is to learn a regression function for a given TF, then predict the TF binding affinity for unseen sequences. There are 66 TF defining 66 different sequence regression tasks. The interpretability of the learned model is also important as new knowledge can be extracted from knowing the individual k-mer binding specificity. The work by [18] presents a systematic comparison of 26 methods evaluated on this benchmark. The winning method was a linear regression model with squared error loss and optimization by gradient descent on sequences represented in the feature space of all 4 to 8-mers (Team_D) [3]. Among the top-5 methods were a Markov model (Team_F) [9], an HMM trained by Expectation-Maximization combined with a linear model (Team_E) [16], and a linear regression model using contiguous and gapped 6-mers (Team_G)[4].

A second round of the challenge has added new competing techniques, the most notable of which are a new HMM model for regression (RegHMM) [19] and a deep learning method based on CNN (named DeepBind) [2]. The linear regression method of Team_D came second. RegHMM which restricts the k-mers to $k = 6$, obtained results similar to Team_D. DeepBind slightly outperformed Team_D regarding prediction accuracy, but used a CNN architecture designed for this specific task and required a custom implementation for GPUs with extensive parameter calibration over 30 sets of parameters. We test our approach on the same TF-DNA benchmark and show that we can achieve similar accuracy to prior techniques, with a much simpler and more efficient approach.

Sequence Classification for Malware. Microsoft released a malware classification benchmark on Kaggle in 2015, containing about 20,000 files amounting to 500GB of data, to train and evaluate classifiers of malware files. Besides the large number of samples, individual files are also quite large (up to 50 Mb per file). Each file can be interpreted as a discrete sequence of bytes. Several competing methods have exploited the sequence structure of the files. The winning method [17] has explicitly generated all k-mers with $k \in \{2, 3, 4\}$ using a machine with 104 Gb memory, 16 CPUs, and extra 200 Gb disk space for the generated data. They have used these features with a boosting method implemented in the Xgboost library. The k-mer features have proven to be very useful for achieving high classification accuracy, but generating them explicitly requires extensive storage and computational resources. Recent work [1] has studied new feature engineering approaches on the same benchmark. They have decided not to use k-mer features due to the excessive computational requirements, but this has lead to lower accuracy than the challenge winner. In [5] the authors use explicitly

[4] http://www.nature.com/nbt/journal/v31/n2/extref/nbt.2486-S1.pdf.

generated 3-mers as features, but use mutual information and random projections to select a feature subset that is manageable for learning a neural network. Malware coders routinely use masking and other code obfuscation techniques. To detect such manipulations, gapped k-mers that allow flexible, rather than exact matching, could improve the classifier. To compare our regression algorithm to prior work, we treat sequence classification as a regression with binary scores. We learn and evaluate our linear regression function using *unrestricted-length k-mer features*.

Linear Sequence Classification. Some of our key intuitions come from the work of [7,8] which proposed the SEQL framework implementing greedy approaches for linear sequence classification. In [8] it was shown that a branch-and-bound approach can be used for a variety of classification loss functions. We build on previous research [8] and propose an algorithm for efficient sequence regression, by exploiting the structure of the feature space and separately bounding the positive and negative coordinate-wise gradients for squared error loss. This bound allows us to guarantee that we iteratively find the best (with respect to a given loss function) k-mer feature from a very large feature space. We compare our methods to the linear sequence classifiers from [7,8] on the malware classification benchmark.

3 Method Proposed

3.1 Basic Notation

Let $D = \{(s_1, y_1), (s_2, y_2), \ldots, (s_N, y_N)\}$ be a training set of instance-label pairs, where $s_i = c_1 c_2 \ldots c_{m_i}$ is a sequence of variable length m_i, with each $c_i \in \Sigma$ a symbol from the alphabet of possible symbols denoted by Σ. For example, in the case of DNA sequences $\Sigma = \{A, C, G, T\}$. Each sequence s_i has an associated score $y_i \in \mathbb{R}$. We define a subsequence as a contiguous part of a sequence, e.g., $s_j = c_2 c_3 c_4$ and write $s_j \subseteq s_i$ if s_j is a subsequence of s_i. Given this definition we can represent a sequence s_i as a binary vector in the space of all subsequences in the training data: $x_i = (x_{i1}, \ldots, x_{ij}, \ldots, x_{in})^T, x_{ij} \in \{0, 1\}, i = 1, \ldots, N$, where $x_{ij} = 1$ means that subsequence s_j occurs in sequence s_i. We denote by n the number of distinct subsequences in the feature space, i.e., the coordinates of the vectors space in which we learn. Although this space is huge and in practice infeasible to generate explicitly, we show how to work with this representation by exploiting its nested structure to develop a lazy search procedure.

The goal is to learn a mapping from sequences to scores, $f : S \rightarrow \mathbb{R}$, from the given training set D, so that we can predict a score $y \in \mathbb{R}$ for a new sample $s \in S$. In our framework we want to learn a linear model, i.e., a parameter vector β that allows us to estimate the real score y by setting $\hat{y} = \beta^T x_i$. Although linear models are not powerful enough to capture non-linear relationships, by working in a very complex feature space (e.g., all k-mers) we can learn a powerful model, similar to the kernel trick applied by kernel Support Vector Machines. We compute $\beta = (\beta_1, \ldots, \beta_j, \ldots, \beta_n)$ by minimizing a loss function over the training set:

$$\beta^* = \operatorname*{argmin}_{\beta \in \mathbb{R}^n} L(\beta). \tag{1}$$

In our work $L(\beta)$ is the regularized squared loss:

$$L(\beta) = \sum_{i=1}^{N} (y_i - \beta^T \cdot x_i)^2 + C R_\alpha(\beta). \tag{2}$$

$C \in \mathbb{R}_0^+$ is the weight for the regularizer $R_\alpha(\beta)$. We use the elastic-net regularizer $R_\alpha(\beta) = \alpha \sum_{j=1}^{n} |\beta_j| + (1-\alpha)\frac{1}{2}\sum_{j=1}^{n} \beta_j^2$ defined in [6] which allows trading-off $l1$ and $l2$ penalties.

3.2 Learning via Coordinate-Descent with Gauss-Southwell Selection

Recent work [12] has shown that for a class of loss functions, which includes the squared loss, learning via coordinate descent is faster than random coordinate descent optimization. Furthermore, for squared loss in particular, coordinate descent via the Gauss-Southwell rule was proven to converge much faster than other coordinate descent methods. In our setting, the feature space of all subsequences is potentially exponential, thus it is not even possible to explicitly compute the full gradient. We first give the generic learning algorithm and then provide an upper bound that makes Gauss-Southwell selection feasible for this complex feature space.

We are interested in solving the convex optimization problem in (1). The coordinate descent method is based on the iteration step:

$$\beta^{(t)} = \beta^{(t-1)} - \eta_{j_t} \frac{\partial L}{\partial \beta_{j_t}} (\beta^{(t-1)}) e_{j_t} \tag{3}$$

To determine the descent direction we use the Gauss-Southwell rule [12]:

$$j_t = \arg\max_j \left| \frac{\partial L}{\partial \beta_j} (\beta^{(t-1)}) \right| \tag{4}$$

This formulation transforms the learning problem into a search problem since in each iteration we have to find the best coordinate j_t, i.e., the subsequence with the largest absolute gradient value. Algorithm 1 shows the basic mechanics of our method. The crucial part of this algorithm is the search for the best coordinate (line 5), for which we present an efficient algorithm in the next section.

Algorithm 1. Greedy Coordinate Descent with Gauss Southwell Selection

```
1: Set $\beta^{(0)} = 0$
2: while not termination condition do
3:     Adjust intercept
4:     Calculate objective function $L(\beta^{(t)})$
5:     Find coordinate $j_t$ with maximum gradient value
6:     Find optimal step size $\eta_{j_t}$ by line search or exact optimization
7:     Update $\beta^{(t)} = \beta^{(t-1)} - \eta_{j_t} \frac{\partial L}{\partial \beta_{j_t}}(\beta^{(t-1)}) e_{j_t}$
8:     Add corresponding feature to feature set
9: end while
```

Step Size. The parameter η_{j_t} is called step size and acts as a scaling factor on the gradient value, to enforce convergence. The work in [12] analyzed a variety of options to set this parameter, from constant step size to exact optimization. As exact optimization was shown to produce much faster convergence, and is feasible to compute for squared loss, we also optimize η_{j_t} exactly.

3.3 Upper Bound for Fast Gauss-Southwell Selection

Formulating a learning algorithm via coordinate descent with Gauss-Southwell rule does not provide a solution for the problem of finding the best subsequence in a huge feature space. Here we give an upper bound on the coordinate-wise gradient value for the squared loss function, that enables us to efficiently search for the best coordinate in each iteration. The theory relies on the following intuitions. First, the subsequence space has a structure that we can exploit to focus the search on parts of the feature space. Namely, we can bound the frequency of a sequence, based on the frequency of any of its subsequences, using an argument similar to that of the Apriori market basket analysis algorithm. Second, we can separate the positive and negative terms of the gradient, to obtain an upper bound on the gradient for squared loss. This allows us to incrementally generate feature candidates[5] and to quickly rule out parts of the feature space, while guaranteeing to find a coordinate with maximum gradient magnitude, as required by the Gauss-Southwell optimization strategy. In the following, $s_j \in x_i$ means that the corresponding vector entry $x_{ij} = 1$. Furthermore, we denote $s_p \subseteq s_j$ if s_p is a subsequence of s_j. Theorem 1 gives an upper bound on the gradient value of any subsequence s_j, using only information about its prefix s_p.

Theorem 1 (Bounding the search for the best coordinate). *Let $L(\beta)$ be the squared loss function and $y_i \in \mathbb{R}$. For any subsequence $s_p \subseteq s_j$, it holds that*

$$\left| \frac{\partial L}{\partial \beta_j}(\beta) \right| \leq \mu(s_p), \text{ where}$$

[5] To generate candidate features we start from 1-mers and use breadth-first-expansion to generate k-mers with $k \geq 1$.

$$\mu(s_p) = \max \left\{ \left| \sum_{\{i|x_{ip}=1, y_i-\beta^T x_i \geq 0\}} -2x_{ip}(y_i - \beta^T x_i) + C(\alpha \text{sign}(\beta_j) + (1-\alpha)\beta_j) \right|, \right.$$
$$\left. \left| \sum_{\{i|x_{ip}=1, y_i-\beta^T x_i \leq 0\}} -2x_{ip}(y_i - \beta^T x_i) + C(\alpha \text{sign}(\beta_j) + (1-\alpha)\beta_j) \right| \right\}$$

Proof. We first focus on bounding the positive terms of the coordinate-wise gradients:

$$\begin{aligned}
\frac{\partial L}{\partial \beta_j}(\beta) &= \sum_{i=1}^{N} -2x_{ij}(y_i - \beta^T x_i) + CR'_\alpha(\beta_j) && (5)\\
&= \sum_{\{i|x_{ij}=1\}} -2x_{ij}(y_i - \beta^T x_i) + CR'_\alpha(\beta_j) && (6)\\
&\leq \sum_{\substack{\{i|x_{ij}=1,\\ y_i-\beta^T x_i \leq 0\}}} -2x_{ij}(y_i - \beta^T x_i) + CR'_\alpha(\beta_j) && (7)\\
&\leq \sum_{\substack{\{i|x_{ip}=1,\\ y_i-\beta^T x_i \leq 0\}}} -2x_{ip}(y_i - \beta^T x_i) + CR'_\alpha(\beta_j) && (8)\\
&\leq \sum_{\substack{\{i|x_{ip}=1,\\ y_i-\beta^T x_i \leq 0\}}} -2x_{ip}(y_i - \beta^T x_i) + C(\alpha \text{sign}(\beta_j) + (1-\alpha)\beta_j) && (9)
\end{aligned}$$

The step from (7) to (8) moves from coordinate j to coordinate p. The inequality holds since $\{i|x_{ij} = 1, y_i - \beta^T x_i \leq 0\} \subseteq \{i|x_{ip} = 1, y_i - \beta^T x_i \leq 0\}$ as every sequence which contains s_j also contains its subsequence s_p. Similarly, by separating the negative terms, we get a second bound:

$$\begin{aligned}
\frac{\partial L}{\partial \beta_j}(\beta) &= \sum_{i=1}^{N} -2x_{ij}(y_i - \beta^T x_i) + CR'_\alpha(\beta_j) && (10)\\
&\geq \sum_{\substack{\{i|x_{ip}=1,\\ y_i-\beta^T x_i \geq 0\}}} -2x_{ip}(y_i - \beta^T x_i) + C(\alpha \text{sign}(\beta_j) + (1-\alpha)\beta_j) && (11)
\end{aligned}$$

The two bounds provide an upper bound on the absolute value of the gradient at coordinate j, using only information about coordinate p:

$$\begin{aligned}
\sum_{\{i|s_p \in x_i, y_i-\beta^T x_i \geq 0\}} -2x_{ip}(y_i - \beta^T x_i) + CR'_\alpha(\beta_j) &\leq \frac{\partial L}{\partial \beta_j}(\beta) && (12)\\
\leq \sum_{\{i|s_p \in x_i, y_i-\beta^T x_i \leq 0\}} -2x_{ip}(y_i - \beta^T x_i) + CR'_\alpha(\beta_j)
\end{aligned}$$

With the regularization term $CR'_\alpha(\beta_j)$ included, the bound depends on the prefix s_p as well as on the weight β_j of the subsequence s_j. Since in the beginning all β_j are set to zero this does not represent a problem, as the regularizer is zero in this case. The bounds of features that were already selected in a previous iteration are the only ones that have to be adjusted by adding this term.

The bound allows us to efficiently search for the coordinate with the largest gradient. Algorithm 2 shows the search procedure. We start the search by expanding from 1-mers. Throughout the search we keep track of the current best feature (*best_feature*) and in τ we save its absolute gradient value. Before the expansion of any subsequence, we check if its upper bound μ is smaller than τ. If this is the case, we can prune the subtree starting at this node, as no further expansion can improve the current gradient.

Algorithm 2. Fast Gauss-Southwell Coordinate Selection

```
 1: $\tau \leftarrow 0$
 2: $best\_feature \leftarrow NIL$
 3: for all $s' \in \bigcup_{i=1}^{N} \{s | s \in s_i, |s| = 1\}$ do          ▷ For each 1-mer
 4:     GROW_SEQUENCE($s'$)
 5: end for
 6: return $best\_feature$
 7:
 1: function GROW_SEQUENCE($s$)
 2:     if $\mu(s) \le \tau$ then return                    ▷ $\mu(s)$ like in Theorem 1
 3:     else if $abs(gradient(s)) > \tau$ then
 4:         $best\_feature = s$                           ▷ Suboptimal solution
 5:         $\tau = abs(gradient(s))$
 6:     end if
 7:     for all $s'' \in \{s' | s' \supseteq s, s' \in \bigcup_{i=1}^{N} s_i, |s'| = |s| + 1\}$ do
 8:         GROW_SEQUENCE($s''$)
 9:     end for
10: end function
```

Proposition 1 (Tightness of upper bound). *The upper bound given in Theorem 1 is tight.*

Proof. It suffices to show one example in which the upper bound (12) is reached. The inequality becomes an equality when, e.g., $y_i - \beta^T x_i = 0, \forall i = 1, \ldots, N$ or whenever all $y_i - \beta^T x_i \le 0, \forall i = 1, \ldots, N$ and the set of occurrences of a subsequence s_j is the same as that of its subsequences $s_p \subseteq s_j$, i.e., $\{i | x_{ij} = 1\} = \{i | x_{ip} = 1\}$.

Proposition 2 (Convergence rate). *The proposed learning algorithm for sequence regression by optimizing the squared loss, converges to the global optimum of the objective function with a convergence rate of*

$$L(\beta^{(t)}) - L(\beta^*) \le \left[\prod_{r=1}^{t} \left(1 - \frac{\mu}{l_{j_r}}\right)\right] [L(\beta^{(0)}) - L(\beta^*)].$$

Proof. We use recent convergence results for coordinate-descent optimization of functions that are μ-strongly convex, with coordinate-wise l_{j_r}-Lipschitz continuous gradient (e.g., squared loss). In particular, we use coordinate descent with Gauss-Southwell selection for squared loss and exact step size optimization. For a detailed proof see [12].

Algorithm Complexity. The time complexity of the proposed algorithm is $O(fN)$ per iteration, where f is the number of features that need to be investigated for Gauss-Southwell selection and N is the number of training examples. *Implementation.* In practice we use data structures such as inverted indexes and tries to fully take advantage of the sparsity and the nested structure of the feature space. We also investigate empirically the quality of the upper bound by computing the number of distinct features investigated per iteration, and measuring the running time per iteration, and for fully learning a model.

4 Experiments

We evaluate our learning algorithm on synthetic data and two benchmarks from recent data challenges. First, we analyze and compare our method to other linear regression methods on simulated sequence regression data where we vary the data generation parameters. Next, we study a sequence regression problem, to compare our learning algorithm to state-of-the-art sequence regression approaches on real data. Finally, we study a sequence classification problem, in order to compare the squared loss bound effectiveness to related methods developed for sequence classification. We run all experiments on a PC with 132 GB RAM, single Intel Xeon 2.4 GHz CPU and 5.4 TB HDD. All our code and data is available online[6].

4.1 Synthetic Data

In this section we analyze our method (named **SqLoss**) on synthetic data. The controled generation of data allows us to compare SqLoss to the state-of-the-art (SotA) methods in a systematic way. We generate sequence regression datasets according to Algorithm 3. Before the actual sequence generation starts, n motifs have to be generated by drawing m symbols from a given alphabet Σ. For each of these motifs, the influence on the response variable (i.e., the motif weight) is set randomly. The first step of the sequence generation is to define which motifs each sequence contains. The binary indicator variables I_{ij} encode this as in (13). Each of these indicator variables is set to 0 or 1 according to a user set probability. Depending on the value of the indicator variable, an insertion position for the motif in question is determined. Next, the actual generation of the string starts. For each position in the sequence, the algorithm checks if a motif has to be inserted. If not, a random symbol from the alphabet is inserted.

[6] Code of our algorithms: https://github.com/svgsponer/SqLoss.

Otherwise the corresponding motif is placed at this position. As soon as the end of the sequence is reached, a score is assigned to the sequence, according to (13), where ϵ is Gaussian noise. This generation process can lead to the case that a motif is present in a sequence by chance. Our implementation checks all generated sequences for unintentionally inserted motifs and replaces them with a random subsequence of the same length as the motif.

$$y_i = \sum_{j=1}^{n} w_j I_{ij} + \epsilon, \text{ where } I_{ij} = \begin{cases} 1, & \text{if sequence } i \text{ contains motif } j \\ 0, & \text{otherwise} \end{cases} \tag{13}$$

Algorithm 3. Generation of Sequence Regression Dataset with $n = 2$ Motifs

Input: Number of sequences N, length of sequence L, number of motifs n, motif length m, alphabet Σ.

Output: Dataset with N (sequence, score) pairs.

```
Generate $n$ motifs by drawing $m$ symbols $\sim U(\Sigma)$
Set weights for each motif
for $i < N$ do
    pos1 $\sim U(L)$ if $I_{i1}$
    pos2 $\sim U(L)$ if $I_{i2}$
    for $l < L$ do
        if $l = pos1$ or $l = pos2$ then
            add motif to sequence
        else
            add symbol $c \sim U(\Sigma)$ to sequence
        end if
    end for
    add sequence $s_i$ to data set with $y_i = w_1 I_{i1} - w_2 I_{i2} + \epsilon$
end for
```

In the following experiment we generate 10,000 sequences of length 5,000 and insert 2 motifs. We compare SqLoss to three regression methods: ordinary least squares (ols), ridge regression (ridge) and linear support vector regression (linsvr). For all these methods we use the implementations in scikit-learn (version 0.17) [13] with default parameters and explicitly generate all k-mers up to $k = 5$. For SqLoss we do not restrict k and use default parameters (see code online). Figures 1 and 2 show the average mean squared error (MSE) and training time over 5 runs, for various alphabet sizes and 4 different motif lengths (3, 5, 7, 10). We note that the SotA methods' performance suffers if the motifs are longer than the maximal extracted k-mer. With increasing alphabet size, this effect vanishes as the density of k-mers feature space decreases. If the k-mer density is low enough, subsequences (e.g., 5-mers) of motifs can already indicate the presence of the whole motif and so be used to learn an appropriate weight. When the motif is shorter than the extracted k-mers, all methods perform similar, even though with increasing alphabet size the SotA methods achieve slightly worse results. We suspect this is caused by overfitting since the feature space becomes huge. Even though this is a simplified regression problem, it is promising to see that SqLoss achieves comparable or better results in different data generation settings, without the need to set k explicitly. This means that our method requires much less feature engineering for achieving good prediction quality.

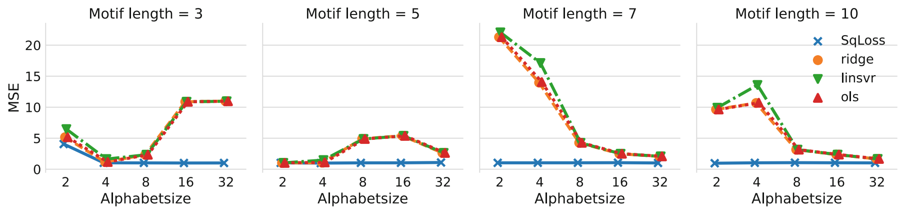

Fig. 1. Average MSE across 5 runs, comparing the impact of varying data generation parameters (alphabet size, motif length) on 4 regression methods. SqLoss has the lowest MSE across all data generation scenarios.

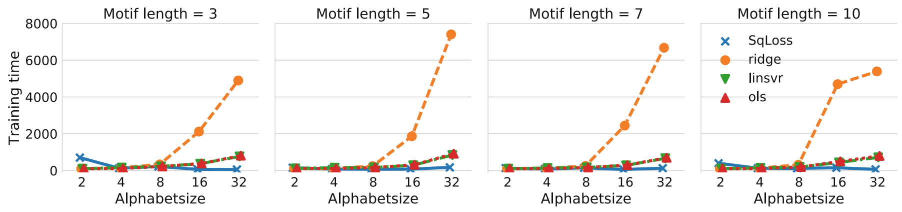

Fig. 2. Average training time (sec) across 5 runs, comparing the impact of varying data generation parameters (alphabet size, motif length) on 4 regression methods.

4.2 TF-DNA Binding Prediction Challenge

The DREAM5 challenge[7] [18] provides 40,000 DNA sequences, each with a numeric score describing the binding affinity of a particular protein called Transcription Factor (TF). The task is to learn a regression function for a given TF. There are 66 TF each defining a different sequence regression task. We use the DREAMTools [4] that allow us to compare our results to the results of the 26 challenge participants [18]. We also show results for DeepBind [2], a recent method that achieved higher scores than previous participants. Of special interest is the comparison to challenge winner (Team_D) [3], a linear regression method using k-mers as features. The authors of [3] pre-process the training data as follows: (1) log2-transform the target scores followed by subtracting the mean; (2) remove all sequences that were flagged as bad by the challenge organizers; (3) remove noisy training sequences; (4) filter low intensity probes and (5) restrict

[7] http://hugheslab.ccbr.utoronto.ca/supplementary-data/DREAM5/.

the k-mer length between 4 and 8. We only apply the first two transforms but none of the latter filters, as we want to study our algorithm's effectiveness when using minimal domain knowledge. We also study the influence of using wildcards for allowing more flexible gapped k-mer features (e.g., `A*B` where `*` is a extra symbol in Σ which stands for any symbol of the alphabet).

Table 1. TF-DNA: results of top-5 sequence regression methods for the DREAM5 data challenge.

Team	Pearson	Pearson_Log	Spearman	AUPR_8mer	AUROC_8mer
DeepBind	0.6780	0.7260	0.7060	0.6760	0.9910
`SqLossWc`	0.6483	0.6846	0.6423	0.7236	0.9967
`SqLoss`	0.6399	0.6791	0.6390	0.7049	0.9953
Team_D	0.6413	0.6742	0.6394	0.6997	0.9942
Team_E	0.6375	0.6936	0.6735	0.5223	0.9524
Team_F	0.6103	0.6732	0.6555	0.5456	0.9766

The final rank is determined by averaging the ranks of each algorithm under each evaluation metric (see [18] for details). Table 1 shows these metrics for the top-5 methods on the benchmark. Our algorithm with one wildcard allowed (SqLossWc) comes second in the overall rank, right after DeepBind. If we do not allow wildcards (SqLoss) our method comes third. We can see the benefit of using wildcards by the increase of the score across all metrics. Figure 3 shows the normalized loss functions and total number of explored nodes per iteration for both models for TF47. Additionally, it shows the Pearson correlation achieved by SqLossWc at each iteration. The flexibility of wildcards clearly increases the number of nodes in the search tree which influences the runtime. SqLoss needs on average 513 s total training time per TF, and SqLossWc takes 2,834 s total training time per TF. Our algorithm reaches similar metrics to Team_D and DeepBind, but the variation without wildcards uses a fraction of the training time of DeepBind and no specific pre-processing as Team_D. The average length of the learned motifs for SqLoss is 5.05 (std across different TFs: 0.16) with a maximum length of 10 bases. For SqLossWc the average motif length is 5.4 (std: 0.09) and the longest learned motif is 11 bases long.

An advantage of our linear model is that it can easily be interpreted, unlike complex non-linear methods such as DeepBind. Table 2 shows some high rank features for TF13 learned with k-mer features with one wildcard allowed. The weights directly reflect the importance of the learned features and can provide important knowledge to domain experts. The learned features are stored in a trie to allow efficient prediction by linearly scanning test sequences. Prediction for 40,000 sequences takes 2 s for SqLoss and 4 s for SqLossWc.

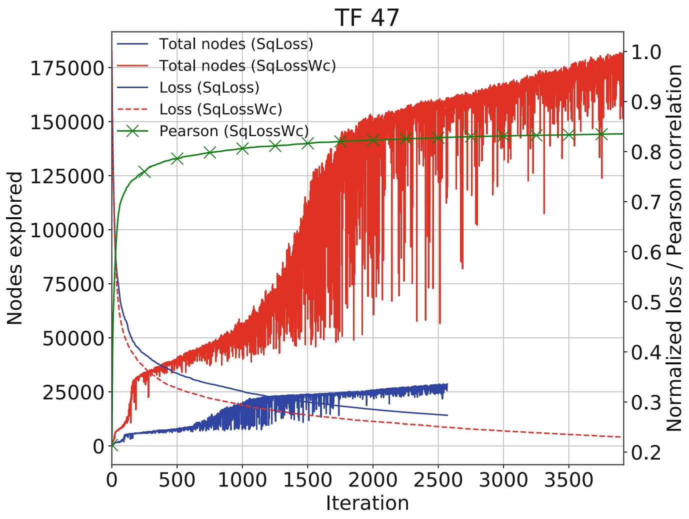

Fig. 3. TF-DNA: loss and explored nodes per iteration, w/o wildcard for protein TF47. In blue SqLoss, in red SqLossWc. Pearson correlation (green) for SqLossWc. (Color figure online)

Table 2. TF-DNA: example positive/negative features from SqLossWc model learned for TF13, where * indicates a wildcard.

Motif	Weight
`TAAT*A`	0.733985
`TAATG*G`	0.706344
`ATG*AAA`	0.674507
⋮	⋮
`GGATA`	−0.188202
`TCAAT`	−0.214858
`G*ATAG`	−0.218132

4.3 Microsoft Malware Classification Challenge

The goal of the Microsoft challenge[8] is to classify files into one of 9 malware families. The training set has 10,868 labeled samples, each with a binary file with hexadecimal representation and a file with the disassembled code. We want to find out if we can build effective classifiers using only the binary (bytes) representation, since disassembling the code requires expensive computation. Even though this dataset poses a classification, rather than a regression task, we use it to compare our method to the challenge winners and to similar approaches developed for linear classification, as implemented in the SEQL framework [8]. In

[8] https://www.kaggle.com/c/malware-classification.

addition, with sequences of up to 7 million symbols in length and rich alphabet ($|\Sigma| = 16$), we want to study and further calibrate our method on this large and challenging dataset.

We compare the accuracy, convergence and bound effectiveness for three learning algorithms: SEQL with classification losses (logistic loss and quadratic hinge loss) versus our SqLoss regression algorithm (for SqLoss we interpret one-vs-all binary labels as numeric scores).

To reduce the size of each bytes file we remove the offset field as well as all question marks and white spaces between hexa bytes. The original challenge uses the multiclass log-loss as main evaluation metric. We do not use this metric as it heavily depends on the calibration of the output scores of each method. Similar to published work [1,17], we report the accuracy results on 4-fold stratified cross-validation (Table 3). The same input data is used for the SEQL methods and SqLoss.

Table 3. Malware: accuracy and training time 4-fold CV.

Method	Accuracy	Training time (mins)	
Wang et al. (challenge winner)	0.9983	(multicore, preprocessing only)	2,880
Ahmedi et al.	0.9976	(multicore, preprocessing only)	2,780
SEQL logistic regression loss	0.9958	(singlecore, full training)	603
SEQL quadratic hinge loss	0.9949	(singlecore, full training)	410
SqLoss (our method)	0.9916	(singlecore, full training)	3,183

We note that SqLoss has lower accuracy than the challenge winner [17] and the recent solution of [1]. Nevertheless, both those methods use a variety of hand picked features extracted from both the binary and the disassembled files. Disassembled code is expensive to extract and is inexact, i.e., it is possible for a single program to have two or more disassemblies. To extract features [1,17] have to do costly data preprocessing and heavy feature engineering. In particular [17] have to limit k-mers to a maximum $k = 4$ and need explicit generation of k-mers (for both training and test data) which requires 100 GB memory, 16 CPUs, and 48 h extraction time on the training data alone.

To better analyze our method we compare it to SEQL, a linear classification method that uses branch-and-bound for selecting subsequence features. As we can see in Table 3, the accuracy of the SEQL losses is comparable to that of SqLoss, while training time is better for the classification losses. Figure 4 (left column) shows the value of the three loss functions (normalized by the start loss) per iteration, for classes 1, 2 and 7. As in [15], we also find that the squared error loss decreases slower than the classification loss functions. This is expected given that SqLoss is a regression algorithm used for a classification task. We currently implement the same stopping criterion for all three losses and we believe this may be ill suited for SqLoss, as in practice we could stop the iterations earlier without compromising accuracy. Figure 4 (right column) shows the total number of nodes explored per iteration during the search for the best feature, for each

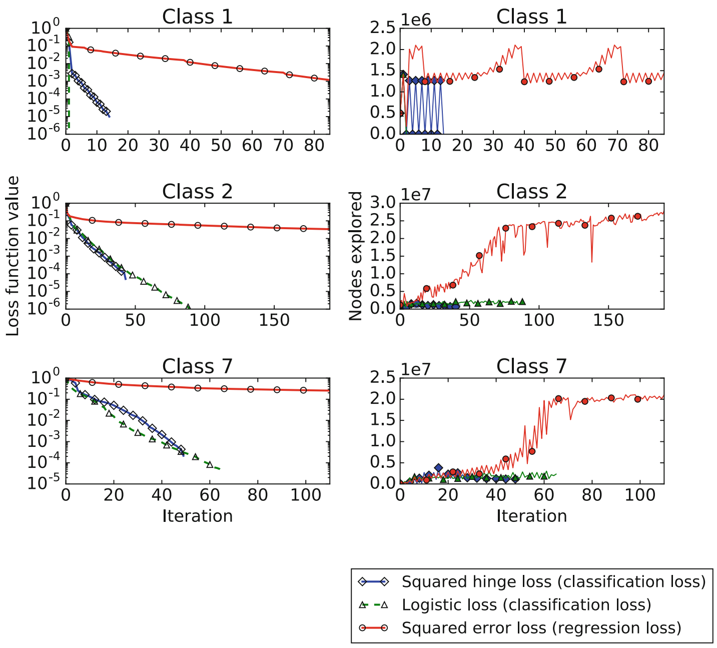

Fig. 4. Malware: comparison of normalized loss function value (left, log scale) and number of nodes explored (right) per iteration for classes 1, 2 and 7. Blue for squared hinge loss, green for logistic regression and red for SqLoss. (Color figure online)

of the three methods. The pruning for SqLoss seems to be less efficient than the one for logistic regression or the hinge loss, on this classification task.

5 Conclusion and Future Work

We present a new method for efficient linear sequence regression in the feature space of all subsequences. The proposed method uses coordinate gradient descent with Gauss-Southwell rule to optimize squared error loss. We propose a branch-and-bound algorithm for efficient Gauss-Southwell selection. Our empirical study shows that we can achieve results comparable to the state-of-the-art, with a simple linear regression model, while employing little to no domain knowledge or pre-processing. In particular, our models can use unrestricted-length, flexible k-mer features (with wildcards), without compromising training and testing efficiency. In the future we want to explore other flexible operators on feature representations of sequences. Further, we want to improve the scalability of our method as well as extend it to make use of multiple cores.

Acknowledgment. This work was funded by Science Foundation Ireland (SFI) under grant number 12/RC/2289.

References

1. Ahmadi, M., Ulyanov, D., Semenov, S., Trofimov, M., Giacinto, G.: Novel feature extraction, selection and fusion for effective malware family classification. In: CODASPY (2016)
2. Alipanahi, B., Delong, A., Weirauch, M.T., Frey, B.J.: Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning. Nat. Biotechnol. **33**(8), 831–838 (2015)
3. Annala, M., Laurila, K., Lähdesmäki, H., Nykter, M.: A linear model for transcription factor binding affinity prediction in protein binding microarrays. PLoS ONE **6**(5), e20059 (2011)
4. Cokelaer, T., Bansal, M., Bare, C., et al.: DREAMTools: a Python package for scoring collaborative challenges. F1000Research (2016)
5. Dahl, G.E., Stokes, J.W., Deng, L., Yu, D.: Large-scale malware classification using random projections and neural networks. In: ICASSP (2013)
6. Hui, Z., Hastie, T.: Regularization and variable selection via the elastic net. J. Roy. Stat. Soc. B **67**(2), 301–320 (2005)
7. Ifrim, G., Bakir, G., Weikum, G.: Fast logistic regression for text categorization with variable-length n-grams. In: KDD (2008)
8. Ifrim, G., Wiuf, C.: Bounded coordinate-descent for biological sequence classification in high dimensional predictor space. In: KDD (2011)
9. Keilwagen, J., Grau, J., Paponov, I.A., Posch, S., Strickert, M., Grosse, I.: De-novo discovery of differentially abundant transcription factor binding sites including their positional preference. PLoS Comput. Biol. **7**(2), e1001070 (2011)
10. Leslie, C., Eskin, E., Noble, W.S.: The spectrum kernel: a string kernel for SVM protein classification. In: PSB (2002)
11. Leslie, C., Kuang, R.: Fast string kernels using inexact matching for protein sequences. JMLR **5**(Nov), 1435–1455 (2004)
12. Nutini, J., Schmidt, M., Laradji, I.H., Friedlander, M., Koepke, H.: Coordinate descent converges faster with the gauss-southwell rule than random selection. In: ICML (2015)
13. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: machine learning in Python. JMLR **12**(Oct), 2825–2830 (2011)
14. Punta, M., Coggill, P.C., Eberhardt, R.Y., Mistry, J., Tate, J., Boursnell, C., Pang, N., Forslund, K., Ceric, G., Clements, J., Heger, A., Holm, L., Sonnhammer, E.L.L., Eddy, S.R., Bateman, A., Finn, R.D.: The Pfam protein families database. Nucleic Acids Res. **40**(Database issue), D290–D301 (2012)
15. Rosasco, L., De Vito, E., Caponnetto, A., Piana, M., Verri, A.: Are loss functions all the same? Neural Comput. **16**(5), 1063–1076 (2004)
16. Schütz, F., Delorenzi, M.: MAMOT: hidden Markov modeling tool. Bioinformatics **24**(11), 1399–1400 (2008)
17. Wang, X., Liu, J., Chen, X.: Microsoft malware classification challenge (BIG 2015) first place team: say no to overfitting. In: BIG (2015)
18. Weirauch, M.T., Cote, A., Norel, R., Annala, M.: Evaluation of methods for modeling transcription factor sequence specificity. Nat. Biotech. **31**(2), 126–134 (2013)
19. Zhang, Y., Henao, R., Carin, L., Zhong, J., Hartemink, A.: Learning a hybrid architecture for sequence regression and annotation. In: AAAI (2016)