Briefings in Bioinformatics
doi.10.1093/bioinformatics/xxxxxx
Advance Access Publication Date: XX XX XXXX
Problem solving protocol

OXFORD

Problem solving protocol

Flexible k-mers with variable-length indels for identifying binding sequences of protein dimers

Chenyang Hong [1], **Kevin Y. Yip** [1,2,3,4,*]

[1]Department of Computer Science and Engineering, [2]Hong Kong Bioinformatics Centre, [3]CUHK-BGI Innovation Institute of Trans-omics, [4]Hong Kong Institute of Diabetes and Obesity, The Chinese University of Hong Kong, Shatin, New Territories, Hong Kong

*To whom correspondence should be addressed.

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Abstract

Many DNA-binding proteins interact with partner proteins. Recently, based on the high-throughput CAP-SELEX method, many such protein pairs have been found to bind DNA with flexible spacing between their individual binding motifs. Most existing motif representations were not designed to capture such flexibly spaced regions. In order to computationally discover more co-binding events without prior knowledge about the identities of the co-binding proteins, a new representation is needed. We propose a new class of sequence patterns that flexibly model such variable regions, and corresponding algorithms that identify co-bound sequences using these patterns. Based on both simulated and CAP-SELEX data, features derived from our sequence patterns lead to better classification performance than patterns that do not explicitly model the variable regions. We also show that even for standard ChIP-seq data, this new class of sequence patterns can help discover co-bound events in a subset of sequences in an unsupervised manner. The open-source software is available at https://github.com/kevingroup/glk-SVM.

Key words: Protein dimers; Sequence motifs; CAP-SELEX; Transcription factors

1 Introduction

DNA sequence motifs are commonly represented by a position weight matrix (PWM) [1], which provides information about the nucleotide distribution at each position, usually in the form of log likelihood when the nucleotide frequencies in the motif are compared to background frequencies. A basic assumption behind the PWM is that different positions in the motif are statistically independent, which does not hold in many situations. Various alternative representations have been proposed without making this assumption, such as Markov models [2; 3], generalized PWMs considering higher-order nucleotide combination frequencies [4; 5; 6], and feature-based models.

Chenyang Hong is a PhD student in Department of Computer Science and Engineering at The Chinese University of Hong Kong. He studies sequential pattern analysis.
Kevin Y. Yip is an associate professor in Department of Computer Science and Engineering at The Chinese University of Hong Kong. He studies transcriptional regulation and human diseases.

Feature-based models derive features from the input sequences and use them to compute pairwise similarities among the sequences, which can then be summarized into a kernel matrix. One of the most fundamental kernels for biological sequences is the spectrum kernel, which uses k-mers as features [7]. As in other uses of k-mers, the choice of the word size k is critical. If k is set too small, the k-mers may not be informative enough to capture key motif features; If k is set too large, the total number of features would be huge, leading to data sparseness and potential over-fitting. To use a large word size without capturing features too specific to the training data, one way is to allow mismatches in k-mer matching [8], and another way is to include wildcards (known as the "gaps") in the k-mers [9]. There were also studies that used alignment-based methods such as the Smith-Waterman dynamic programming algorithm [10] to obtain pairwise sequence similarities [11].

These kernels were originally developed for classifying protein sequences, but they have also been adopted and further developed for classifying DNA sequences [12; 13; 14; 15], which have a smaller alphabet but possibly longer motifs. In either case, the sequences can be classified by a kernel method such as a support vector machine (SVM). Based on some testing results involving protein binding data from ChIP-seq experiments, gapped k-mers could identify transcription factor (TF) binding sequences better than standard k-mers, especially for TFs with long binding motifs [14].

Although gapped k-mers can ignore unimportant positions in a motif by setting these positions as wildcards, they cannot handle motifs with variable lengths. In particular, many TFs form physical interactions with partner TFs [16] and some TF dimers bind DNA with a variable-length spacer sequence between two relatively rigid sites [17; 18]. For example, the binding motif of TP53 contains a variable-length spacer of 0-20 base pairs between two half sites [19], while HOX4A and FOXA1 can bind DNA in the form of a dimer with their binding sites separated by up to 30bp [20]. Recently, with the high-throughput method CAP-SELEX (consecutive affinity-purification systematic evolution of ligands by exponential enrichment) [21], which determines sequences preferentially bound by specific TF pairs, variable spacing has been observed in the binding sequences of many TF pairs.

Here we show that by extending gapped k-mers to allow a variable insertion-deletion (indel) region, the resulting features can better identify binding sequences of TF pairs. The corresponding support vector machine models constructed identify sequences bound by a pair of TFs not only by looking for their individual motifs but also the distance between them learned from some training data. This is useful not only when analyzing CAP-SELEX data but ChIP-seq data of individual TFs as well, since the TFs may have unrecognized co-binding partners and our novel sequence features can discover co-binding events from a subset of sequences in an unsupervised manner.

In the literature, some previous studies have tried to predict TFs that can form dimers [22; 23], which is a problem very different from the one studied here, namely identifying DNA sequences bound by TF dimers. Some other studies have tried to detect binding sites of transcription factors that bind DNA in dimers [24; 25], but these methods either did not model the dimer sites explicitly or required some prior information such as the position specific scoring matrix (PSSM) of the potential co-binding factors. There were also some previous methods that tried to model sequence motifs with variable indel regions [26; 27], but these methods either require exact motifs at the half sites or cannot handle large indels. Motifs containing a variable indel were also studied in the context of protein sequences [28], although the method is alignment-based, which is difficult to apply to our problem since our input contains both sequences bound by the TF pair and those that are not bound.

2 Methods

2.1 The glk pattern

To allow for a variable-length indel region within each k-mer, we first define a general type of sequence patterns called glk patterns:

Definition 1. **glk pattern -** *A* **glk pattern** *for DNA is a sequence with g wildcard characters ('**N**'), l indel characters ('—') and k nucleotide characters ('**A**', '**C**', '**G**' or '**T**') in any order.*

For example, **A** — **N**— is a glk pattern with $g = 1$, $l = 2$ and $k = 1$.

A sequence is said to match a glk pattern if it can be generated from the pattern. Specifically,

Definition 2. **match -** *A sequence s* **matches** *a glk pattern p if p can be converted into s by a series of operations from the following list:*

- *Substitute a wildcard character by a nucleotide character*
- *Substitute an indel character by a nucleotide character*
- *Delete an indel character*

For example, **ACG** matches the glk pattern **A** — **N**— but **CCG** does not.

Due to the indels, only sequences of certain lengths can match a glk pattern:

Property 1. **Necessary length condition** *A sequence s can match a glk pattern only if the length of s is at least $g + k$ and at most $g + l + k$.*

These minimum and maximum lengths respectively correspond to the situations that every indel is removed and every indel is substituted by a nucleotide character.

One way to check whether a sequence s matches a glk pattern p is to perform a global alignment between them with the following scoring matrix:

		From s				
		A	**C**	**G**	**T**	**.**
	A	0	$-\infty$	$-\infty$	$-\infty$	$-\infty$
	C	$-\infty$	0	$-\infty$	$-\infty$	$-\infty$
	G	$-\infty$	$-\infty$	0	$-\infty$	$-\infty$
From p	**T**	$-\infty$	$-\infty$	$-\infty$	0	$-\infty$
	N	0	0	0	0	$-\infty$
	—	0	0	0	0	0
	.	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$,

where **.** is the gap character used in this alignment (not to be confused with the indel character — used in the glk pattern). The optimal alignment score can be computed in $O(|s||p|)$ time by dynamic programming.

Lemma 1. *s matches p if and only if the highest alignment score is 0.*

Proof. If the alignment score is 0, any alignment having this score exactly corresponds to a way that p can be converted to s by the three types of operations, and thus by definition s matches p. Conversely, if p can be converted to s by the three types of operations, the exact operations can be represented by an alignment between s and p with score 0, since all the valid operations would give a score of 0 according to the scoring matrix.

In the Supplementary Materials, we describe an algorithm for finding all glk patterns that a sequence matches (Section S1.1) . We also discuss there how the glk patterns commonly matched by two sequences can be obtained (Sections S1.2-S1.3).

2.2 The glk-kernel problem

Our ultimate goal is to construct a kernel matrix for a set of sequences based on their glk-pattern occurrence frequencies, such that sequences that contain a motif can be distinguished from those that do not using an SVM. To define this kernel matrix formally, we first introduce some additional terms:

Definition 3. **support count -** *The* **support count** *of a sequence S for a glk pattern p is the number of S's contiguous sub-sequences that match p.*

For example, if S=**ACGAT**, its support count for the glk pattern p=**AN**— is 3, since the contiguous sub-sequences of S that match p are $S[1..2]$=**AC**, $S[1..3]$=**ACG** and $S[4..5]$=**AT**.

Definition 4. **support count vector -** *The* **support count vector** *of a sequence S is the vector of its support counts for all the $\binom{g+l+k}{k}4^k\binom{g+l}{g}$ possible glk patterns, with the glk patterns ordered lexicographically.*

Definition 5. **the glk kernel matrix -** *Given a set of input sequences $S_1, S_2, ...,$ the* **glk kernel matrix** *is a matrix where entry (i, j) is the inner product between the support count vectors of S_i and S_j.*

With these definitions, we can formally define the glk-kernel problem:

Definition 6. **the glk-kernel problem -** *Given a set of input sequences $S_1, S_2, ...,$ the* **glk-kernel problem** *is to compute the glk kernel matrix of these sequences.*

A simple algorithm for computing the glk-kernel of a set of sequences is given in the Supplementary Materials (Section S1.4) .

2.3 glk patterns with consecutive indels

Since our main purpose of proposing the glk patterns is to study binding motifs of TF dimers, we are particularly interested in glk patterns that contain a single consecutive block of indels. We call them the glk-ci patterns:

Definition 7. **glk-ci pattern -** *A* **glk-ci (glk with consecutive indels) pattern** *is a glk pattern in which all indels appear in a consecutive region.*

For example, **AN — —** is a glk-ci pattern with $g = 1$, $l = 2$ and $k = 1$, but **A — N—** is not a glk-ci pattern since the two indels are separated by a non-indel character.

The definition for a sequence to match a glk-ci pattern is the same as that for a general glk pattern.

Due to its definition, each glk-ci pattern is composed of two (possibly empty) non-indel regions:

Definition 8. **Head and tail of a glk-ci pattern -** *The head and tail of a glk-ci pattern p are respectively the sub-sequences of p before and after the indel region.*

For example, the head and tail of the glk-ci pattern **AN — —** are **AN** and **∅** (the empty sequence), respectively.

It is easy to determine whether a sequence matches a glk-ci pattern:

Lemma 2. *Suppose a glk-ci pattern p has h characters in its head and t characters in its tail. A sequence s of length n matches p if and only if i) s satisfies the length condition, ii) the first h characters of s match the head of p, and iii) the last t characters of s match the tail of p.*

The proof of it essentially follows the definition of glk-ci patterns.

Due to the lack of indels in the head, the first h characters of s match the head of p if and only if every non-wildcard character in the head of p is the same as the corresponding character in s at the same position. Tail match can be determined in a similar way. As a result, the number of glk-ci patterns that each sequence matches is easy to compute (Section S1.5).

Now, suppose we have two sequences s_1 and s_2 of lengths n_1 and n_2 with $g + k \leq n_1 \leq n_2 \leq g + l + k$, and we want to find out all the glk-ci patterns that they commonly match. This can be done by repeating the following procedure for all possibilities in the different steps:

1. Construct a sequence p initialized with n_2 indels, which will become the glk-ci pattern.
2. Determine the sizes of the head and tail regions, h and t, where $h + t = g + k$.
3. Pick in total k positions in the head and tail of p that s_1 and s_2 have the same character, i.e., $s_1[x] = s_2[x]$ in the head or $s_1[n_1 - y] = s_2[n_2 - y]$ in the tail, for any $x \in [1, h]$ and any $y \in [0, t - 1]$.
4. If less than k positions can be picked in the Step 3, this head and tail pair gives no glk-ci patterns commonly matched by s_1 and s_2. Otherwise, for each of the k positions picked, copy the character from s_1 to p.
5. Replace all unpicked positions in the head and tail of p by wildcards.
6. Insert an extra $g + l + k - n_2$ indels to p between the head and tail.

Steps 1-3 of the above procedure can also be used to compute the number of glk-ci patterns commonly matched by two sequences, since the remaining steps are fixed once the head and tail regions have been selected and the k positions from them have been picked. Therefore, to compute this number efficiently, it is only necessary to know how many combinations of the k positions can be picked for each pair of head and tail regions. It is wasteful to re-determine this combination number from scratch for every pair of head and tail regions, since each choice is highly similar to the previous one.

To see how these combination numbers can be determined efficiently, note that for each position x in s_1, if it appears in a head region, it must pair up with position x in s_2. Similarly, if position $n_1 - y$ in s_1 appears in a tail region, it must pair up with position $n_2 - y$ in s_2. For example, suppose s_1=**ACG** and s_2=**ACTG**, the **C** in s_1 must pair up with the **C** in s_2 if it is in the head region since they are both at the second position, and it must pair up with the **T** in s_2 if it is in the tail region since they are both at the second to last position. Whether a position can appear in the head or tail region depends on g and k. A position x cannot be in any head region if $x > g + k$, since the length of the head region can be at most $g + k$. Similarly, a position $n - y$ cannot be in any tail region if $y > g + k$ for the same reason.

Using these ideas, we can record whether each position in s_1 can contribute to the k positions in Step 3 for different pairs of head-tail locations. For example, suppose again that s_1=**ACG**, s_2=**ACTG**, $g = 1$ and $k = 1$ (and $l \geq 1$). There are 3 possible pairs of head-tail locations, namely those having sizes of (2, 0), (1, 1) and (0, 2), respectively. For the **A** in s_1, it contributes 1 to the head in both the first and second pairs since it pairs with the **A** in s_2. For the **C** in s_1, it contributes 1 to the head in the first pair since it pairs with the **C** in s_2, and 0 to the tail in the third pair since it pairs with the **T** in s_2. For the **G** in s_1, it contributes 1 to the tail in both the second and third pairs since it pairs with **G** in s_2. Therefore, the number of positions that can be picked in Step 3 for the three pairs is 1+1+0=2, 1+0+1=2 and 0+0+1=1, respectively. As a result, the total number of glk-ci patterns commonly matched by these two sequences is $\binom{2}{1} + \binom{2}{1} + \binom{1}{1} = 2 + 2 + 1 = 5$, regardless of the value of l, as long as $n_1 \leq n_2 \leq g + l + k$, i.e., $l \geq 2$.

Putting all these together systematically, Algorithm S2 can be used to compute the number of glk-ci patterns commonly matched by two sequences. In this algorithm, since the values $\binom{x}{k}$ for various x are commonly needed, they can be pre-computed for quick referencing. With this efficient algorithm, the kernel can in turn be computed efficiently using Algorithm S1.

2.4 A tree algorithm for the glk-ci-kernel problem

Algorithm S2 has a time complexity of $O((g + k)^2)$. For a sequence of length N, there are $(N-g-k+1)+...+(N-g-k-l+1) = O(Nl)$ sub-sequences that satisfy the length condition, assuming $N \gg g + k$. Therefore for two sequences each of length $O(N)$, the total time needed to compute their similarity is $O(N^2 l^2 (g + k)^2)$. Finally, if there are M sequences in total, the time needed to compute the whole kernel matrix would be $O(M^2 N^2 l^2 (g + k)^2)$, which is expensive when M or N is large.

Practically, this approach may lead to the running of Algorithm S2 on many sequence pairs that do not match any common glk-ci patterns. This can be avoided by using a tree structure to index head-tail combinations, similar to the tree proposed in [14] for computing the kernel based on gapped-kmers without indels.

Specifically, assume $l > 0$ (since the $l = 0$ case is much simpler) and we want to index sequence S. For each sub-sequence of S that satisfies the length condition, we consider all possible head-tail combinations of it and insert each of them into the tree. Each nucleotide is inserted as a node down a path, with the first nucleotide inserted as a direct child of the root. The node is given type "head" (resp. "tail") if this nucleotide is part of the head (resp. tail). When a nucleotide is inserted to the tree, it can reuse a node if both the node label and node type are consistent with this inserting nucleotide. Based on this procedure, each path from the root to a leaf corresponds to a particular (g+k)-mer, which is the non-indel part of exactly one possible glk-ci pattern. The ID of sequence S is further stored in the leaf node, together with the number of sub-sequence of S that contains this head-tail combination.

As an example, suppose $g = 1$, $l = 2$ and $k = 2$. For the sequence S_j=**ACTGC**, there are 6 sub-sequences satisfying the length condition and all the corresponding head-tail combinations are as follows:

1. **ACT**: (∅,**ACT**), (**A**,**CT**), (**AC**,**T**), (**ACT**,∅)
2. **ACTG**: (∅,**CTG**), (**A**,**TG**), (**AC**,**G**), (**ACT**,∅)
3. **ACTGC**: (∅,**TGC**), (**A**,**GC**), (**AC**,**C**), (**ACT**,∅)
4. **CTG**: (∅,**CTG**), (**C**,**TG**), (**CT**,**G**), (**CTG**,∅)
5. **CTGC**: (∅,**TGC**), (**C**,**GC**), (**CT**,**C**), (**CTG**,∅)
6. **TGC**: (∅,**TGC**), (**T**,**GC**), (**TG**,**C**), (**TGC**,∅)

The tree after inserting these head-tail combinations is shown in Figure 1.

Fig. 1. The tree after inserting all head-tail combinations of the sub-sequences of S_j=**ACTGC** that satisfy the length condition, for $g = 1$, $l = 2$ and $k = 2$. Head and tail nodes are shown with solid and dotted borders, respectively. The leaf nodes store both the sequence ID (j in this case) and the number of its sub-sequences that have this head-tail combination.

The whole tree-construction process is summarized in Algorithm S3.

The tree is used to compute the similarity between every pair of sequences as follows. For each sequence S_i, we take every sub-sequence s_i of it that satisfies the length condition, and performs a depth-first-search (DFS) of each head-tail combination in the whole tree. During the DFS, if the node type does not match, the sub-tree is abandoned. If the node type matches but the node label does not match, a mismatch counter is incremented. When the mismatch counter becomes larger than g, the sub-tree is again abandoned. Finally, if a leaf node is reached, the sub-sequences with non-zero counts at this node are concluded to share common glk-ci patterns with s_i. If a sequence S_j has a count of z_j at this node, the number of common glk-ci patterns between s_i and sub-sequences of S_j due to this head-tail combination is $z_j \binom{g+k-miss}{k}$, where $miss$ is the current value of the mismatch counter. The whole algorithm is shown in Algorithm S4.

2.5 Practical variations of the glk-ci-kernel problem

In practice, two variations of the glk-ci-kernel problem are also considered.

The first variation is disallowing the indel region to appear at either end of a glk-ci pattern, since those patterns are generally not interesting. It can be easily implemented by requiring both the head and tail of every glk-ci pattern to have at least one position. This variation usually reduces the number of possible glk-ci patterns and correspondingly the time needed for computing the kernel matrix dramatically. In this study, we implemented this variation by further requesting the indel to appear exactly in the middle of a pattern.

The second variation is permitting a glk-ci pattern to appear on either strand of a DNA sequence, which means a sequence can match both a glk-ci pattern and its reverse complement. This can be achieved by adding also the reverse complement of each sequence to the head-tail tree.

2.6 Testing the effectiveness of glk-ci patterns

To test the effectiveness of glk-ci patterns in capturing motif information, we used them as features to classify TF binding sequences using several datasets and compared the results with gapped k-mer features, gappy pair kernel [29; 30] and (on the CAP-SELEX dataset only due to long running time) the gap-weighted string kernel [31]. For both our method (glk-ci-SVM) and the previously proposed gapped k-mer SVM without indels (gkm-SVM), we trained and tested the SVMs using the R package provided by Ghandi et al. [32]. For the gappy pair kernel, we used the implementation in the R KeBABS package [33]. For the gap-weighted string kernel, we used the implementation in the R string kernels package (http://cran.r-project.org/src/contrib/Archive/stringkernels/).

In all the tests, we performed five-fold cross validation and used the area under the receiver-operator characteristics (AUROC) and area under the precision-recall curve (AUPR) to quantify classification performance.

We also tested the effectiveness of glk-ci patterns as general sequence features for unsupervised clustering. Applying this to the sequences bound by a TF according to ChIP-seq data, we discovered a subset of the sequences co-bound by another TF, thereby demonstrating the use of glk-ci as a discovery tool.

2.6.1 Simulated dataset for TF pairs

To explore the ability of glk-ci in detecting co-binding events, we generated simulated data as follows. We downloaded ChIP-seq peak files of some TFs from the Jaspar database [34], based on a report that these TFs form pairs with a preferred spacing between their binding sites [20]. For each TF pair, we generated 800 positive sequences with the two motifs placed with a distance between them drawn uniformly from a range (either [1,7], [1,11] or [1,15]), with the non-motif regions copied from other parts of the peak sequences. We then generated the same number of negative sequences by placing the two motifs following a different uniform distribution. Specifically, when motif distances for the positive sequences were drawn uniformly from the range [1, d], the distances for the negative sequences were drawn from [d+1, 2d]. Since these negative sequences also contained the two motifs, whether an SVM could distinguish between the positive and negative sequences would depend on the presence of information about the spacing between them in the kernel. We also generated a similar dataset with more negative sequences than positive sequences to test the performance of glk-ci-SVM in this situation.

In gkm-SVM, l denotes the total pattern length. Since in glk patterns l is the number of indels, to avoid confusion, we denote the former as tot. To identify the appropriate values of k and tot for gkm-SVM, we tried 5 different values of tot around the actual total span of the two motifs and the spacing between them, to give an advantage to this competing method. We then randomly selected a TF pair to test several values of k, and found that the best performance was achieved for k=6, which was then used in this set of experiments. For gappy pair kernel, we tried 3 different values of k (length of the kmers which are considered in pairs by this kernel), together with different values of m (maximal number of irrelevant positions between a pair of kmers) around the actual spacing. For glk-SVM, we set g to a small value of 2, and k=10 such that g+k is approximately the length of a motif. We also tried other values of g and k without this constraint of their sum. Since the indel parameter l is the most difficult one to set in practice, we developed a method that searches for an appropriate value of it automatically. Specifically, we found that in real data the maximum difference between the occurrence counts of a glk-ci pattern in the positive and negative sets correlates strongly with classification performance (Section S2.2). Therefore, our procedure tries different values of l based on the training data and uses the one that maximizes this occurrence count difference.

For this dataset, we used the version of glk-ci-SVM that disallows indels at pattern ends. Since all sequences were generated using the same strand of the motifs, it was not necessary to consider reverse complement.

2.6.2 CAP-SELEX dataset for TF pairs

While the simulated dataset allowed us to mimic the situation of co-binding TFs, the distance between the two motifs was generated by artificial distributions. To further test the performance of glk-ci-SVM on real co-binding data, we studied a CAP-SELEX dataset with 30 TF pairs [21]. We downloaded the sequencing reads from European Nucleotide Archive (accession PRJEB7934), and selected 800 reads (or all reads if total was

smaller than 800) for each TF pair containing the canonical binding motifs of both TFs as the positive sequences, following the approach adopted in the original paper [21]. We extended these original 40bp sequences to 80bp by inserting random nucleotides to the two ends (in order to have the same length for the positive and background sequences). For each positive sequence, we generated a corresponding background sequence by randomly moving the distance between the two canonical motifs following a uniform distribution of $[d_{min}, d_{max}]$, where d_{min} was set to the negative of the length of the shorter motif (in which case the two motifs would overlap and one of them would be trimmed because of that), and d_{max} was set to 80 minus the length of the shorter motif (in which case the two motifs would be maximally separated). It should be noted that the distance distribution of these background sequences overlaps the distance distribution of the positive sequences, making them more difficult to be distinguished as compared to the positive and negative sequences in the simulated dataset.

For gkm-SVM, we again used some random TF pairs to try out different parameter values and used the set that gave the best classification performance. For glk-ci-SVM, we set g=2, k=8 and used the automatic procedure to search for l. For this dataset, we used both the versions of glk-ci-SVM that allow and disallow indels at the end to compare their performance. For gappy pair kernel, we randomly selected some TF pairs to test several values of k, and found that the best performance was achieved for k=3, and set the value of m to be the same as l (since both of them represent the length of spacing allowed). As in the case of the simulated dataset, it was not necessary to consider reverse complement. For gap-weighted string kernel, we fixed the gap length penalty factor $lambda$=0.9 (which gave the best performance), and tried 4 values of $length$, the number of exact match characters, from 8 to 14. We also tried larger values of $length$ but the performance was not improved.

2.6.3 ENCODE ChIP-seq dataset for single TFs

The last dataset involved 497 sets of ChIP-seq data of individual DNA-binding proteins from different human cell lines produced by the ENCODE Consortium [35; 36]. It was included to test whether the glk-ci pattern is also useful for identifying sequences from ChIP-seq binding data. We used the processed data provided by Alipanahi et al. (2015) [37] (downloaded from http://tools.genes.toronto.edu/deepbind/nbtcode/). We took the sequences within the top 1000 peaks of each protein as the positive set, and generated a corresponding negative set by randomly shuffling the nucleotides in each positive sequence while preserving dinucleotide frequencies. For gkm-SVM, we used its default value of k=6 (which is close to the length of a typical motif) and tested four different settings, based on two values of the total pattern length (tot=9,10) and whether reverse complement is considered. For glk-ci-SVM, we also fixed $k = 6$ and used two different ways to set the values of g and l, namely 1) setting their total to be the same as the wildcard value of gkm-SVM: $(g, l) = \{(0, 3), (1, 2), (2, 1)\}$, or 2) fixing the g to be the same as the wildcard value of gkm-SVM, and varied the value of l: $l = \{1, 2, 3, 4, 5, 6, 7\}$. For this dataset, we used the version of glk-ci-SVM that allows indels to appear at pattern ends, since we expected many protein binding motifs to not contain a variable-length indel region in the middle.

2.6.4 Clustering with glk-ci patterns as sequence features

For the unsupervised clustering task, we randomly selected 8,000 ChIP-seq peak sequences of the TF CEBPB in the K562 cell line from the ENCODE dataset and used glk-ci patterns (with g=2, l=5 and k=8) to define their similarity matrix. We then applied spectral clustering on the data, using the implementation in the Python scikit-learn library [38]. For each cluster, we performed de novo motif analysis using HOMER [39] to see whether the sequences in the cluster are enriched in another motif.

3 Results

3.1 Simulated dataset

For the simulated dataset, our new glk-ci-SVM method generally performed better than the previous gkm-SVM method and gappy pair kernel, sometimes by a big margin (Figure 2, Sections S2.3, S2.4). This general trend was not affected by the method for aggregating results across parameter settings or by the evaluation measure. For each TF pair, gkm-SVM generally performed better when the spacing between the two motifs was small. In contrast, glk-ci-SVM consistently performed well regardless of the spacing.

Fig. 2. The AUROC (a) and AUPR (b) results of glk-ci-SVM, gappy pair kernel and gkm-SVM based on the simulated dataset.

To provide some insights regarding the most important features for separating the positive and negative sequences, we extracted the glk-ci patterns with the largest difference between their occurrence counts in the positive and negative sequences. Figure 3 shows an example for the ELF1-FOSL1 pair. It shows the 5 glk-ci patterns with the strongest differential occurrence, with the head and tail of these patterns aligned, and their indel region shown as a distribution based on the positive sequences that support these patterns. Quantified by the PWMSimilarity function of the TFBSTools R package [40], the head and tail sequences of these top glk patterns had a similarity with the corresponding motifs in Jaspar [34] from 0.953 to 0.993, showing that they are highly similar. It can also be seen that the distance between the head and tail vary a lot in the different positive sequences. The top 5 glk-ci patterns with the strongest differential occurrence of the other TF pairs are shown in Figure S1.

Fig. 3. Top 5 glk-ci patterns with strongest differential occurrence between the positive and negative sequences for the ELF1-FOSL1 pair based on the simulated data. The distributions show the numbers of supporting positive sequences with respect to the size of the indel region of the glk-ci patterns. The parameters of glk-ci-SVM were set to g=2, l=7 and k=12. The Jaspar motifs for ELF1 and FOSL1 are shown at the bottom for comparisons, with their similarities with the head and tail of the top glk patterns shown in the "Similarity" columns.

3.2 CAP-SELEX dataset for TF pairs

For the CAP-SELEX dataset, the version of glk-ci-SVM disallowing indels at pattern ends almost always outperformed the version allowing end indels (Figure 4a-b), confirming the usefulness of this variation in practice. This version also performed better than gkm-SVM (Wilcoxon two-sided signed-rank test p=1.24e-5 for AUROC and 4.72e-2 for AUPR, same test used for the p-values below). As an example, the TF pair TFAP2C-ONECUT2 had the largest performance difference between gkm-SVM and glk-ci-SVM disallowing end indels, with the AUROC values of 0.38 and 0.70, respectively. Consistent with this, the spacing between these two TFs in the positive sequences was fairly flexible, with an average of 9.9 nucleotides and a standard deviation of 4.0 nucleotides. In contrast, the spacing distribution of the TF pair POU2F1-TBX21 had the smallest standard deviation, and correspondingly the performance of gkm-SVM and glk-ci-SVM was similar, both separating positive and negative sequences almost perfectly.

Similarly, glk-ci-SVM outperformed gap-weighted string kernel generally (p=1.74e-4 for AUROC and 1.19e-3 for AUPR) (Figure 4c-d). For some TF pairs that glk-ci-SVM and gkm-SVM could predict almost perfectly, gap-weighted string kernel could not perform as well. For example, both glk-ci-SVM and gkm-SVM could achieve the AUROC

Fig. 4. Performance of glk-ci-SVM, gkm-SVM and gap-weighted string kernel based on the CAP-SELEX dataset. (a)-(b) Comparisons of glk-ci-SVM and gkm-SVM based on the AUROC (a) and AUPR (b) results of the two methods among their parameter settings. (c)-(d) Comparisons of glk-ci-SVM and gap-weighted string kernel based on the AUROC (c) and AUPR (d) results of the two methods. (e)-(f) Comparisons of glk-ci-SVM and gappy pair kernel based on the AUROC (e) and AUPR (f) results of the two methods.In each panel, the diagonal dotted line corresponds to the situation when the two comparing methods have the same performance.

value of 0.98 for the TF pair FOXJ3-TBX21, while gap-weighted string kernel could only reach 0.91. As for the comparison to gappy pair kernel, glk-ci-SVM also outperformed it generally (p=2.35e-6 for AUROC and 6.15e-4 for AUPR) (Figure 4e-f).

3.3 ENCODE ChIP-seq dataset for single TFs

Figure 5 compares the performance of glk-ci-SVM and gkm-SVM based on the ENCODE dataset for single TFs. Overall, the two methods had similar performance, in terms of both AUROC (panels a and b) and AUPR (panels c and d). However, when the performance of gkm-SVM was low (auroc $<$ 0.8), glk-ci-SVM provided better classification performance (p=2.36e-8 for AUROC and 1.78e-7 for AUPR), as reflected by having more points above the equal-performance line for lower values of gkm-SVM (panel a). This was not due to specific parameter settings, since the same trend is also observed with the average result of all the parameter settings (panel b).

Fig. 5. Performance of glk-ci-SVM and gkm-SVM based on the ENCODE dataset. (a-d) The best AUROC (a), average AUROC (b), best AUPR (c) and average AUPR (d) results of glk-ci-SVM and gkm-SVM among their parameter settings. (e-f) The best AUROC (e) and average AUROC (f) of glk-ci-SVM with or without considering reverse complement. In each panel, the diagonal line corresponds to the situation when the two methods have the same performance.

For example, for the ChIP-seq data of EZH2 in normal human astrocytes, gkm-SVM achieved an AUROC value of 0.78 while glk-ci-SVM achieved 0.80. This improvement is consistent with the fact that EZH2 can exist in both monomer and dimer forms [41], although the relative frequencies of the two forms when binding target DNA require further investigations.

In contrast, when a DNA-binding protein has a highly conserved motif or binds DNA alone with a single DNA-binding domain, glk-ci-SVM could not improve the performance of gkm-SVM. For example, the binding motif of CTCF is rigid and highly conserved. Among the different ChIP-seq datasets of this protein, the performance of gkm-SVM and glk-ci-SVM was highly similar, with both achieving an average AUROC higher than 0.95. In fact, for CTCF, even a position weight matrix or consensus sequence could also distinguish positive and negative sequences very well.

When comparing glk-ci-SVM with or without considering reverse complement, the performance was consistently better when reverse complement was considered (panels e and f), showing that this variation is practically useful when analyzing ChIP-seq data.

3.4 Clustering with glk-ci patterns as sequence features

For the unsupervised clustering task, when we produced 6 clusters, one of them showed very strong enrichment (p=1e-318) for a motif highly similar to the motif of the TEAD family (Figure 6). This cluster contained 361 sequences and 98.34% of them contained this motif. CEBPB was previously reported to co-bind with TEAD4 based on CAP-SELEX data [21], consistent with the high similarity between the motif identified from this cluster and the canonical TEAD4 motif (HOMER match score=0.52). Similar results were also obtained when we changed the number of clusters to 4, 5 or 7.

Interestingly, if we did not perform the clustering but performed a de novo motif discovery from all the TF binding sequences of CEBPB directly, there was no enrichment of the TEAD family motifs, showing that the co-binding occurred in only a subset of the CEBPB-bound sequences and the glk-ci features successfully identified this subset.

These results demonstrate that glk-ci patterns can be used to discover co-binding events without prior knowledge of the co-binding TFs.

Fig. 6. Sequence logos of (a) the motif identified de novo from the cluster, (b) the canonical TEAD3 motif and (c) the canonical TEAD4 motif.

3.5 Runtime Benchmark

To evaluate the running time of glk-ci-SVM (the version disallowing indels at pattern ends), we randomly selected one ENCODE ChIP-seq dataset, which contained 2000 peaks with each peak sequence of length 101bp. We used four different parameter settings ((g,l,k) = (2,5,6), (2,5,8), (2,10,6) and (3,6,6)) to benchmark the running time of glk-ci kernel calculation with different number of threads (from 1 to 16).

The results (Figure 7) show that the running time of glk-ci SVM increased linearly with respect to the numbers of indels allowed (l) and exact matches (k), which are consistent with our complexity analysis. If only one thread is used, the running time of glk-ci-SVM could be long when the parameter values are large, especially for the number of wildcards, g. This problem was alleviated by our multi-threading implementation. When 16 threads were used, the kernel calculation finished within 100 seconds even for large parameter values. Since we have shown above that good classification performance could already be achieved with a small value of g in practice, the running time of glk-ci SVM should not be a problem in real applications.

Fig. 7. Running time of glk-ci-SVM with different parameter settings on the benchmark data.

We also tested the scalability of glk-ci-SVM with respect to different number of sequences. The results (Section S2.5) show that the running time remains short with 8000 input sequences.

4 Discussion

In this study, we have proposed the glk and glk-ci patterns that contain flexible indels, and corresponding algorithms for computing the kernel matrix based on the support count vectors of all glk-ci patterns efficiently. Empirical results based on simulated and CAP-SELEX data confirmed that glk-ci patterns led to better classification performance than gapped k-mers that do not allow indels. In fact, glk patterns can be used not only for modeling CAP-SELEX data, but also for analyzing standard ChIP-seq data, in which case the positive sequences can be bound by the target protein alone, or both the target protein and other proteins in complex with it. By using glk patterns to define sequence similarity, it is possible to identify these different subsets of sequences by unsupervised clustering using any clustering method that requires only a similarity matrix as input. We also implemented a glk-ci pattern counting method, which extracts glk-ci patterns with the strongest differential occurrence in the positive

and negative sequences. It could provide important biological insights regarding the binding motifs of partner TFs and their spacing preference.

The ability of glk patterns in detecting a subset of binding sequences that involve co-binding of another factor can also be used for a different purpose, namely "cleaning up" the binding sequences by identifying the subset not bound by other co-factors, which can help better discover the binding motif of the factor when it binds DNA alone.

CAP-SELEX uses synthetic random sequences, which may identify co-bound sequences different from the ones *in vivo*. This limitation is tackled by new methods such as reChIP-seq [42], the data produced by which will be used to further evaluate glk patterns.

In this work, we have chosen SVM as the classifier. One key reason was that SVM only requires a kernel matrix as input rather than the individual feature vectors, which avoided materializing the long support count vectors of all possible glk patterns. In general, glk patterns can be efficiently used as sequence features with any machine learning methods that can take a similarity matrix as input.

Although we have only tested our method on transcription factor binding sequence prediction, especially for TF dimers with variable spacing, our method can also be adopted to other sequence analysis problems not limited to DNA sequences. The glk and glk-ci patterns we propose can be used as general sequence features, which may be useful in various sequence analysis tasks involving flexible regions. More generally, since the glk kernel includes many popular kernels as special cases, including the gkm kernel, spectrum kernel and some other string kernels, glk-SVM can be used as a general method in various classification tasks of biological sequences, including homology detection, metagenomic sequence classification, evaluation of evolutionary conservation and RNA binding proteins prediction.

Key Points

- We propose a new class of sequence patterns that flexibly model variable regions, and corresponding algorithms that identify TF dimer co-bound sequences using these patterns.
- We provide a pattern counting method that can be used for extracting important features and guiding parameter setting.
- We show that using features derived from our sequence patterns lead to better classification performance than patterns that do not explicitly model the variable regions for transcription factor dimer binding prediction.
- The sequence patterns we propose can be used as general sequence features in different tasks, including de novo discovery of co-binding events.

5 Acknowledgement

We would like to thank William Wenjie Li for helpful discussions. KYY is partially supported by the HKSAR General Research Fund 14170217.

References

1. Stormo GD, Schneider TD, Gold L, *et al.* Use of the 'perceptron' algorithm to distinguish translational initiation sites in e. coli. *Nucleic Acids Research* 1982;**10(9)**:2997–3011.
2. Zhang MQ, Marr TG. A weight array method for splicing signal analysis. *CABIOS* 1993;**9(5)**:499–509.
3. Ellrott K, Yang C, Sladek FM, *et al.* Identifying transcription factor binding sites through markov chain optimization. *Bioinformatics* 2002; **18**:S100–S109.
4. Gershenzon NI, Stormo GD, Ioshikhes IP. Computational technique for improvement of the position-weight matrices for the DNA/protein binding sites. *Nucleic Acids Research* 2005;**33**:2290–2301.
5. Siddharthan R. Dinucleotide weight matrices for predicting transcription factor binding sites: Generalizing the position weight matrix. *PLOS ONE* 2010;**5**:e9722.
6. Tomovic A, Oakeley EJ. Position dependencies in transcription factor binding sites. *Bioinformatics* 2007;**23**:933–941.
7. Leslie C, Eskin E, Noble WS. The spectrum kernel: A string kernel for SVM protein classification. In: *Pacific Symposium in Biocomputing*. 2001; pp. 564–575.
8. Leslie CS, Eskin E, Cohen A, *et al.* Mismatch string kernels for discriminative protein classification. *Bioinformatics* 2004;**20**:467–476.
9. Leslie C, Kuang R. Fast string kernels using inexact matching for protein sequences. *Journal of Machine Learning Research* 2004; **5**:1435–1455.
10. Smith T, Waterman M. Identification of common molecular subsequences. *Journal of Molecular Biology* 1981;**147(1)**:195 – 197.
11. Liao L, Noble WS. Combining pairwise sequence similarity and support vector machines for detecting remote protein evolutionary and structural relationships. *Journal of computational biology* 2003; **10(6)**:857–868.
12. Lee D, Karchin R, Beer MA. Discriminative prediction of mammalian enhancers from DNA sequences. *Genome Research* 2011;**21**:2167–2180.
13. Fletez-Brant C, Lee D, McCallion AS, *et al.* kmer-SVM: A web server for identifying predictive regulatory sequence features in genomic data sets. *Nucleic Acids Research* 2013;**41**:W544–W556.
14. Ghandi M, Lee D, Mohammad-Noori M, *et al.* Enhanced regulatory sequence prediction using gapped k-mer features. *PLOS Computational Biology* 2014;**10**:e1004035.
15. Lee D. LS-GKM: A new gkm-SVM for large-scale datasets. *Bioinformatics* 2016;**32**:2196–2198.
16. Ravasi T, Suzuki H, Cannistraci CV, *et al.* An atlas of combinatorial transcriptional regulation in mouse and man. *Cell* 2010;**140**:744–752.
17. Roulet E, Bucher P, Schneider R, *et al.* Experimental analysis and computer prediction of CTF/NFI transcription factor DNA binding sites. *Journal of Molecular Biology* 2000;**297**:833–848.
18. Siggers T, Gordân R. Protein-DNA binding: Complexities and multi-protein codes. *Nucleic Acids Research* 2014;**42**:2099–2111.
19. Riley T, Sontag E, Chen P, *et al.* Transcriptional control of human p53-regulated genes. *Nature Reviews Molecular Cell Biology* 2008; **9(5)**:402–412.
20. Guo Y, Mahony S, Gifford DK. High resolution genome wide binding event finding and motif discovery reveals transcription factor spatial binding constraints. *PLOS Computational Biology* 2012; **8(8)**:e1002638.
21. Jolma A, Yin Y, Nitta KR, *et al.* DNA-dependent formation of transcription factor pairs alters their binding specificity. *Nature* 2015; **527(7578)**:384–388.
22. Jankowski A, Szczurek E, Jauch R, *et al.* Comprehensive prediction in 78 human cell lines reveals rigidity and compactness of transcription factor dimers. *Genome Research* 2013;**23(8)**:1307–1318.
23. Lai FJ, Jhu MH, Chiu CC, *et al.* Identifying cooperative transcription factors in yeast using multiple data sources. *BMC systems biology* 2014;**8(5)**:S2.
24. Sharon E, Lubliner S, Segal E. A feature-based approach to modeling protein–dna interactions. *PLoS computational biology* 2008;

4(8):e1000154.

25. Pudimat R, Schukat-Talamazzini EG, Backofen R. A multiple-feature framework for modelling and predicting transcription factor binding sites. *Bioinformatics* 2005;**21(14)**:3082–3088.
26. Hu YJ. Finding subtle motifs with variable gaps in unaligned DNA sequences. *Computer Methods and Programs in Biomedicine* 2003; **70(1)**:11–20.
27. Leibovich L, Yakhini Z. Efficient motif search in ranked lists and applications to variable gap motifs. *Nucleic Acids Research* 2012; **40(13)**:5832–5847.
28. Frith MC, Saunders NFW, Kobe B, *et al.* Discovering sequence motifs with arbitrary insertions and deletions. *PLoS Computational Biology* 2008;**4**:5.
29. Kuksa P, Huang PH, Pavlovic V. A fast, large-scale learning method for protein sequence classification. In: *8th Int. Workshop on Data Mining in Bioinformatics*. 2008; pp. 29–37.
30. Hamilton M, Reddy A, Ben-Hur A. Kernel methods for calmodulin binding and binding site prediction. In: *Proceedings of the 2nd ACM Conference on Bioinformatics, Computational Biology and Biomedicine*. 2011; pp. 381–386.
31. Rousu J, Shawe-Taylor J. Efficient computation of gapped substring kernels on large alphabets. *Journal of Machine Learning Research* 2005;**6**:1323–1344.
32. Ghandi M, Mohammad-Noori M, Ghareghani N, *et al.* GkmSVM: An R package for gapped-kmer SVM. *Bioinformatics* 2016;**32**:2205–2207.
33. Palme J, Hochreiter S, Bodenhofer U. Kebabs: an r package for kernel-based analysis of biological sequences. *Bioinformatics* 2015; **31(15)**:2574–2576.
34. Mathelier A, Fornes O, Arenillas DJ, *et al.* JASPAR 2016: A major expansion and update of the open-access database of transcription factor binding profiles. *Nucleic Acids Research* 2016;**44**:D110–D115.
35. The ENCODE Project Consortium. An integrated encyclopedia of DNA elements in the human genome. *Nature* 2012;**489(7414)**:57–74.
36. Yip KY, Cheng C, Bhardwaj N, *et al.* Classification of human genomic regions based on experimentally-determined binding sites of more than 100 transcription-related factors. *Genome Biology* 2012;**13**:R48.
37. Alipanahi B, Delong A, Weirauch MT, *et al.* Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning. *Nature Biotechnology* 2015;**33(8)**:831–838.
38. Pedregosa F, Varoquaux G, Gramfort A, *et al.* Scikit-learn: Machine learning in python. *Journal of machine learning research* 2011; **12(Oct)**:2825–2830.
39. Heinz S, Benner C, Spann N, *et al.* Simple combinations of lineage-determining transcription factors prime cis-regulatory elements required for macrophage and b cell identities. *Molecular cell* 2010;**38(4)**:576–589.
40. Tan G, Lenhard B. Tfbstools: an r/bioconductor package for transcription factor binding site analysis. *Bioinformatics* 2016; **32(10)**:1555–1556.
41. Wu H, Zeng H, Dong A, *et al.* Structure of the catalytic domain of EZH2 reveals conformational plasticity in cofactor and substrate binding sites and explains oncogenic mutations. *PLOS ONE* 2013; **8**:e83737.
42. Kinkley S, Helmuth J, Polansky JK, *et al.* reChIP-seq reveals widespread bivalency of H3K4me3 and H3K27me3 in CD4+ memory T cells. *Nature Communications* 2016;**7**:12514.
43. Wasserman WW, Sandelin A. Applied bioinformatics for the identification of regulatory elements. *Nature Reviews Genetics* 2004; **5(4)**:276.

6 Figure Legends

Fig 1. The tree after inserting all head-tail combinations of the sub-sequences of S_j=**ACTGC** that satisfy the length condition, for $g = 1$, $l = 2$ and $k = 2$. Head and tail nodes are shown with solid and dotted borders, respectively. The leaf nodes store both the sequence ID (j in this case) and the number of its sub-sequences that have this head-tail combination.

Fig 2. The AUROC (a) and AUPR (b) results of glk-ci-SVM, gappy pair kernel and gkm-SVM based on the simulated dataset.

Fig 3. Top 5 glk-ci patterns with strongest differential occurrence between the positive and negative sequences for the ELF1-FOSL1 pair based on the simulated data. The distributions show the numbers of supporting positive sequences with respect to the size of the indel region of the glk-ci patterns. The parameters of glk-ci-SVM were set to g=2, l=7 and k=12. The Jaspar motifs for ELF1 and FOSL1 are shown at the bottom for comparisons, with their similarities with the head and tail of the top glk patterns shown in the "Similarity" columns.

Fig 4. Performance of glk-ci-SVM, gkm-SVM and gap-weighted string kernel based on the CAP-SELEX dataset. (a)-(b) Comparisons of glk-ci-SVM and gkm-SVM based on the AUROC (a) and AUPR (b) results of the two methods among their parameter settings. (c)-(d) Comparisons of glk-ci-SVM and gap-weighted string kernel based on the AUROC (c) and AUPR (d) results of the two methods. (e)-(f) Comparisons of glk-ci-SVM and gappy pair kernel based on the AUROC (e) and AUPR (f) results of the two methods.In each panel, the diagonal dotted line corresponds to the situation when the two comparing methods have the same performance.

Fig 5. Performance of glk-ci-SVM and gkm-SVM based on the ENCODE dataset. (a-d) The best AUROC (a), average AUROC (b), best AUPR (c) and average AUPR (d) results of glk-ci-SVM and gkm-SVM among their parameter settings. (e-f) The best AUROC (e) and average AUROC (f) of glk-ci-SVM with or without considering reverse complement. In each panel, the diagonal line corresponds to the situation when the two methods have the same performance.

Fig 6. Sequence logos of (a) the motif identified de novo from the cluster, (b) the canonical TEAD3 motif and (c) the canonical TEAD4 motif.

Fig 7. Running time of glk-ci-SVM with different parameter settings on the benchmark data.

Supplementary materials

S1 Supplementary methods

S1.1 All glk patterns that a sequence matches

The set of all glk patterns that a sequence (with length n, where $g+k \leq n \leq g+l+k$) matches can be determined by repeating the following procedure for all possibilities:

1. Choose g of the characters and turn them into wildcards.
2. Choose $n-g-k$ of the remaining characters and turn them into indels.
3. Insert $g+l+k-n$ additional indels anywhere.

For example, suppose $g=1$, $l=2$ and $k=1$, from the sequence s=**ACG** (where $n=3$), the glk patterns that s matches (16 of them in total, in arbitrary orders) are determined as follows:

- After Step 1: **ACN**, **ANG**, **NCG**
- After Step 2: **AN**−, **A** − **N**, **NC**−, −**CN**, **N** − **G**, −**NG**
- After Step 3: **AN** − −, **A** − **N**−, −**AN**−,**A** − −**N**, −**A** − **N**, **NC** − −, **N** − **C**−, −**NC**−, −**CN**−, −**C** − **N**, − − **CN**, **N** − **G**−, **N** − −**G**, −**N** − **G**, −**NG**−, − − **NG**

If we use the idea of this procedure to count the number of unique glk patterns that a sequence matches, since some glk patterns can be created by multiple ways of the three steps, it is necessary to avoid double counting. In the above example, **A** − **N**− can be created by substituting **C** in **ACN** by an indel in Step 2, followed by adding an indel at the end; Alternatively, it can be created by substituting **G** in **ANG** by an indel, followed by adding indel right after **A**. The only other glk pattern above that can be produced by multiple ways from s is −**N** − **G**.

Although this double counting issue makes it difficult to compute the exact number of unique glk patterns that a sequence matches, we can nonetheless get an upper bound of it easily by considering the special case that no double counting exists:

$$C_1(g,l,k,n) \;\leq\; \binom{n}{g}\binom{n-g}{n-g-k}\binom{2g+l+2k-n}{g+k}, \tag{1}$$

where each term on the right hand side respectively corresponds to one of the three steps, and the term for the third step comes from choosing $g+k$ non-indel characters as the "partitions" to separate the indels added in Step 3. Applying this formula to the above example, we get $\binom{3}{1}\binom{3-1}{3-1-1}\binom{2\times1+2\times1+2-3}{1+1}$ = $\binom{3}{1}\binom{2}{1}\binom{3}{2}$ = 3 × 2 × 3 = 18, which is slightly larger than the actual value of 16 due to the two double-counting cases.

A special case occurs when $n=g+l+k$, in which case Step 3 does not apply:

Lemma S1. *For a sequence s of length n where $n=g+l+k$, the number of glk patterns that s matches is exactly the value given by Formula 1.*

Proof. When $n=g+l+k$, all glk patterns are produced by substitutions with no insertions involved. There is thus a simple one-to-one correspondence between the characters in s and in each glk pattern, and each choice combination of Steps 1 and 2 would lead to a different glk pattern.

Another special case occurs when $n=g+k$, in which case Step 2 does not apply:

Lemma S2. *For a sequence s of length n where $n=g+k$, the number of glk patterns that s matches is exactly the value given by Formula 1.*

The proof of it is similar to the previous case, by having a one-to-one correspondence between the characters in s and the non-indel characters in p.

These two special cases imply that $l=1$ is a simple case:

Corollary S1. *When $l=1$, for every sequence s that satisfies the length condition, the number of glk patterns that it matches is given by Formula 1.*

This is because any s satisfying the length condition must have length either $g+k$ or $g+l+k$.

In general, when both Steps 2 and 3 apply, there are three sources of double counting. First, a nucleotide character in p could come from different instances of that character in s. Second, a wildcard character in p could come from different characters in s. Third, the indel characters in p could be added in either Step 2 or Step 3.

S1.2 glk patterns that two sequences commonly match

The set of all glk patterns that two sequences (with lengths n_1 and n_2, where $g+k \leq n_1 \leq n_2 \leq g+l+k$) commonly match can be determined by repeating the following procedure for all possibilities:

1. Form a global alignment between the two sequences with at least k match positions, $g+k$ non-indel (i.e., match or mismatch) positions, and n positions in total, where $n \leq g+l+k$.
2. Form a sequence p initialized with n indels, for constructing the glk pattern.
3. Choose k of the match positions in the alignment and copy these matched characters to the corresponding positions of p.
4. Choose g of the remaining match or mismatch positions and set the corresponding positions of p to wildcards.
5. Insert $g+l+k-n$ additional indels to p anywhere.

Step 1 is particularly difficult since it requires the enumeration of all alignments that satisfy the criteria.

In some special cases, the number of glk patterns that two sequences commonly match is easy to compute.

One special case is when $n_1 = n_2 = g + l + k$. In this case, there is only one possible alignment, namely one that has no indels inserted. In this case, the number of glk patterns that the two sequences commonly match depends on the number of mismatches m between them:

$$C_{2a}(g, l, k, n_1, m) \;=\; \binom{n_1 - m}{k}\binom{n_1 - k}{g} \tag{2}$$

S1.3 Whether two sequences commonly match at least one glk pattern

To quickly check whether two sequences commonly match at least one glk pattern without comparing their full sets of matching glk patterns, one way is to check whether the two sequences contain a common sub-sequence of length at least k, for otherwise it would be impossible to choose k match positions in any alignment between the two sequences in Step 3. This is a necessary but not sufficient condition. For instance, if the only alignments that give k match positions do not have at least g mismatch positions, then Step 4 would be impossible.

For any two sequences of lengths n_1 and n_2, whether this condition is satisfied can be determined in $O(n_1 n_2)$ time using the dynamic programming algorithm for longest common sub-sequence.

S1.4 A simple algorithm for computing the glk-kernel

Since each support count vector can be very long when g, l or k is large, and many support counts are zero for a given sequence, it is very inefficient to compute the glk-kernel by constructing the actual support count vectors. Following the idea proposed in the gapped k-mer SVM paper [14], one way to speed up is to consider only the glk patterns supported by each sequence. Since we have a (not very efficient) procedure for generating all glk patterns that each sequence matches, one simple algorithm is as follows:

Algorithm S1 A simple algorithm for computing the glk-kernel

function glk-kernel1($\{\mathbf{S}\}, g, l, k$) ▷ $\{\mathbf{S}\} = S_1, S_2, \ldots$: the set of input sequences
▷ g, l, k: specified values for defining glk patterns
  Define A as the kernel matrix initialized with all 0's
  for each pair of sequences $S_x, S_y \in \{\mathbf{S}\}$ **do**
    for each sub-sequence s_i of S_x with a length between $g + k$ and $g + l + k$ **do**
      for each sub-sequence s_j of S_y with a length between $g + k$ and $g + l + k$ **do**
        Define $\text{sim}_{i,j}$ as the number of glk patterns commonly supported by s_i and s_j
        $A[x, y]$ += $\text{sim}_{i,j}$
        $A[y, x]$ += $\text{sim}_{i,j}$
      end for
    end for
  end for
  Return A
end function

A simple variation of this algorithm is to check whether s_i and s_j match at least one common glk pattern, before intersecting their full sets of matching glk patterns. This variation would be useful when the extra checking time is small as compared to the time needed for taking the set intersection.

S1.5 Finding the set and number of glk-ci patterns that each sequence matches

The whole set of glk-ci patterns that a sequence s of length n ($g + k \le n \le g + l + k$) matches can be determined by repeating the following procedure for all possibilities:

1. Form a sequence p initialized with n indels, for constructing the glk-ci pattern.
2. Determine the head and tail regions, with a total of $g + k$ characters.
3. From the head and tail regions together, pick k positions and copy the corresponding characters from s to p.
4. Replace all unpicked positions in the head and tail of p by wildcards.
5. Insert an extra $g + l + k - n$ indels to p between the head and tail.

Since each choice combination from the first two steps would lead to a different glk-ci pattern, the total number of glk-ci patterns that a sequence matches can be easily computed:

$$Cci(g, l, k, n) = \begin{cases} \binom{g+k}{g}, & \text{if } l = 0 \\ (g + k + 1)\binom{g+k}{g}, & \text{if } l > 0 \end{cases} \tag{3}$$

In the special case of $l = 0$, the head and tail merge into a single section and thus it is only necessary to determine which g characters are to be copied to the glk-ci pattern. In other cases, first the distribution of characters into head and tail should be decided (with $g + k + 1$ choices), followed by choosing the g characters from these two segments together.

By definition, any glk pattern with $l = 0$ or $l = 1$ must also be a glk-ci pattern since it is impossible to have two disconnected indels given these values of l. When $l = 0$, any sequence that matches a glk pattern must have length $n = g + k$ due to the length condition, and thus by Lemma S2, Formula 1 gives the exact number of glk patterns that a sequence matches. Similarly, when $l = 1$, by Corollary S1, Formula 1 also gives the exact number.

Therefore, when $l = 0$ or $l = 1$, Formula 1 and Formula 3 should be identical. This is indeed the case: When $l = 0$, $n = g + k$,

$$\begin{aligned} & \binom{n}{g}\binom{n-g}{n-g-k}\binom{2g+l+2k-n}{g+k} \\ = \ & \binom{g+k}{g}\binom{k}{0}\binom{g+k}{g+k} \\ = \ & \binom{g+k}{g}; \end{aligned}$$

When $l = 1$, $n = g + k$ or $n = g + k + 1$. For $n = g + k$,

$$\begin{aligned} & \binom{n}{g}\binom{n-g}{n-g-k}\binom{2g+l+2k-n}{g+k} \\ = \ & \binom{g+k}{g}\binom{k}{0}\binom{g+k+1}{g+k} \\ = \ & \binom{g+k}{g}(g+k+1); \end{aligned}$$

And when $l = 1$ and $n = g + k + 1$,

$$\begin{aligned} & \binom{n}{g}\binom{n-g}{n-g-k}\binom{2g+l+2k-n}{g+k} \\ = \ & \binom{g+k+1}{g}\binom{k+1}{1}\binom{g+k}{g+k} \\ = \ & \binom{g+k+1}{g}(k+1) \\ = \ & \binom{g+k}{g}[(g+k+1)/(g+k+1-g)](k+1) \\ = \ & \binom{g+k}{g}(g+k+1). \end{aligned}$$

S1.6 Computing the number of glk-ci patterns commonly matched by two sequences

Algorithm S2 can be used to compute the number of glk-ci patterns commonly matched by two sequences.

S1.7 A tree-based algorithm for efficiently computing the number of glk-ci patterns commonly matched by two sequences

The tree-construction process is summarized in Algorithm S3. The use of it in computing the number of glk-ci patterns for each pair of sequences is described in Algorithm S4.

S1.8 Quantifying the similarity between a head/tail of a glk pattern and a sequence motif

We computed the similarity between the head/tail of a glk pattern (which is a k-mer possibly with wildcard characters) and a sequence motif as follows. First, we represented a head/tail as a position probability matrix, where for each non-wildcard position the specified nucleotides has a probability of 1 and all other nucleotides have a probability of 0, while for each wildcard position all four nucleotides have a probability of 0.25. This position probability matrix was then converted to a position weight matrix [43]. Finally, the similarity between this position weight matrix and the position weight matrix of a sequence motif was quantified by the PWMSimilarity function (with Pearson correlation as measurement) of the TFBSTools R package.

Algorithm S2 Algorithm for computing the number of glk-ci patterns commonly matched by two sequences

function Cci(s_1, s_2, g, l, k) ▷ s_2, s_2: the two sequences of lengths n_1 and n_2 (with $n_1 \leq n_2$)
▷ g, l, k: specified values for defining glk-ci patterns
  if $n_2 < g + k$ OR $n_1 > g + l + k$ **then**
    Return 0
  else if $n_1 = n_2 = g + k$ **then**
    Define count := 0 ▷ Number of positions with same characters in s_1 and s_2
    for i = 1 to n_2 **do**
      if $s_1[i] = s_2[i]$ **then**
        count += 1
      end if
    end for
    if $l > 0$ **then**
      Return $\binom{\text{count}}{k}$ * (g + k + 1)
    end if
    Return $\binom{\text{count}}{k}$
  else
    Define counts as an array of $g + k + 1$ integers all initialized to 0 ▷ First array index is 1 here
    ▷ Counting for the head region
    for i = 1 to $g + k$ **do** ▷ For each position of s_1 and s_2 counting forward
      if $s_1[i] = s_2[i]$ **then**
        for j = 1 to $g + k + 1 - i$ **do** ▷ Position i only contributes to these heads
          counts[j] += 1
        end for
      end if
    end for
    ▷ Counting for the tail region
    for i = 1 to $g + k$ **do** ▷ For each position of s_1 and s_2 counting backward
      if $s_1[n_1 - i + 1] = s_2[n_2 - i + 1]$ **then**
        for j = 1 to $g + k + 1 - i$ **do** ▷ Position i only contributes to these tails
          counts[$g + k + 2 - j$] += 1
        end for
      end if
    end for
    ▷ Compute the final answer
    Define answer := 0
    for j = 1 to $g + k + 1$ **do**
      answer += $\binom{\text{counts}[j]}{k}$
    end for
    Return answer
  end if
end function

Algorithm S3 Algorithm for building Head-Tail tree for a set of input sequences

function buildTree($\{\mathbf{S}\}, g, l, k$) ▷ $\{\mathbf{S}\} = S_1, S_2, ...$: the set of input sequences
  Define $Root$ as the root of the tree
  for each sequence $S_x \in \{\mathbf{S}\}$ **do**
    for each sub-sequence s_i of S_x with a length between $g + k$ and $g + l + k$ **do**
      Define $n := |s_i|$ ▷ Length of s_i
      for j = 0 to $g + k$ **do**
        Define $H := s_i[1..g + k - j]$ ▷ First $g + k - j$ nucleotides of s_i as head
        ▷ $s_i[a..b] = \emptyset$ if $a > b$
        Define $T := s_i[n - j + 1..n]$ ▷ Last j nucleotides of s_i as tail
        insertTree(i, H, T, $Root$) ▷ Insert this head-tail combination into tree
      end for
    end for
  end for
  Return $Root$
end function
function insertTree(i, H, T, $Root$) ▷ i: ID of the sequence
▷ H, T: head and tail part
  Define p := $Root$
  for each nucleotide x of H **do**
    if p does not have a child of type "head" and label x **then**
      Create a head node q with label x as a child of p
    end if
    p := child of p of type "head" and label x ▷ Must have one now
  end for
  for each nucleotide x of T **do**
    if p does not have a child of type "tail" and label x **then**
      Create a tail node q with label x as a child of p
    end if
    p := child of p of type "tail" and label x ▷ Must have one now
  end for
  Increment the counter of sequence S_i at node p by 1
end function

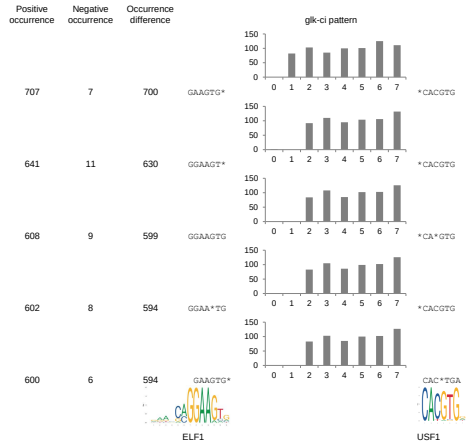
Algorithm S4 Algorithm for computing the kernel matrix among a set of input sequences using the Head-Tail tree

function computeKernel($\{\mathbf{S}\}, g, l, k, Root$)
▷ $\{\mathbf{S}\} = S_1, S_2, ...$: the set of input sequences
▷ $Root$: root node of the index tree
  Define Sim as the kernel matrix, initialized with all zeros
  for each sequence $S_x \in \{\mathbf{S}\}$ **do**
    for each sub-sequence s_i of S_x with a length between $g + k$ and $g + l + k$ **do**
      Define $n := |s_i|$ ▷ Length of s_i
      for j = 0 to $g + k$ **do**
        Define $H := s_i[1..g + k - j]$
▷ First $g + k - j$ nucleotides of s_i as head
▷ $s_i[a..b] = \emptyset$ if $a > b$
        Define $T := s_i[n - j + 1..n]$
▷ Last j nucleotides of s_i as tail
        dfsTree($i, H, T, g, k, Root, Sim$, 0)
▷ using DFS to find this head-tail combination in the tree
      end for
    end for
  end for
end function
function dfsTree($i, H, T, g, k, Root, Sim, mc$)
▷ i: sequence ID, H, T: head and tail of the sub-sequence
▷ Sim: the kernel matrix, mc: mismatch count
  if $Root$ is a leaf node **then**
    for each sequence ID $j \neq i$ with a non-zero count z_j at this node **do**
      $Sim[i, j] += z_j \binom{g+k-mc}{k}$
    end for
  else
    if $H \neq \emptyset$ **then**
      Define $type$="head"
      Define $nuc = H[1]$
      $H := H[2..|H|]$
    else
      Define $type$="tail"
      Define $nuc = T[1]$
      $T := T[2..|T|]$
    end if
    for each child node $Child$ of $Root$ **do**
      if $Child$.type = $type$ and ($mc < g$ or $Child$.label=nuc) **then**
        if $Child$.label=nuc **then**
          dfsTree($i, H, T, g, k, Child, Sim, mc$)
        else
          dfsTree($i, H, T, g, k, Child, Sim, mc + 1$)
        end if
      end if
    end for
  end if
end function

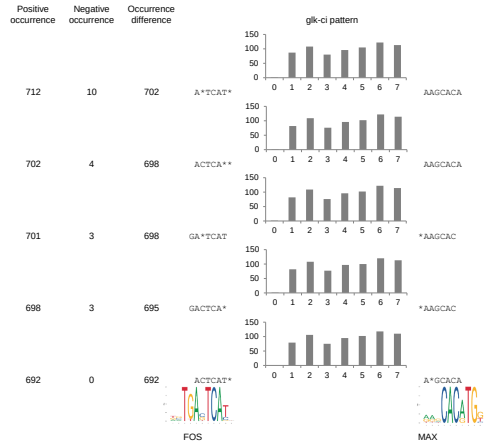
S2 Supplementary results

S2.1 Top 5 glk-ci patterns for the simulated data

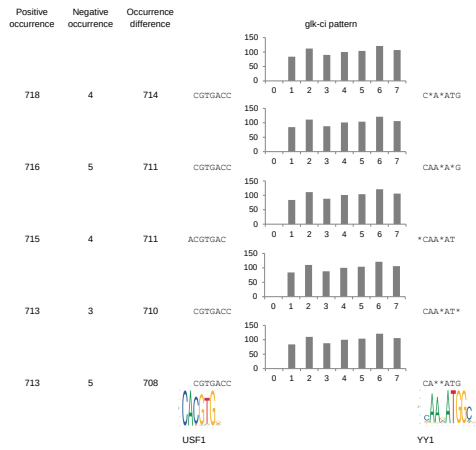
Figure S1 shows the 5 glk-ci patterns with the strongest differential occurrence for TF pairs (a) ELF1-USF1, (b) FOS-MAX, (c) USF1-YY1, (d) STAT1-MAX and (e) FOS-YY1. In each case, the head and tail of these patterns are aligned, and the indel region is shown as a distribution based on the positive sequences that support these patterns.



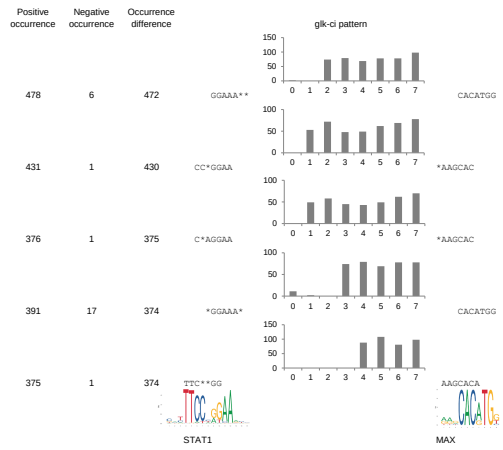
(a) ELF1 -USF1



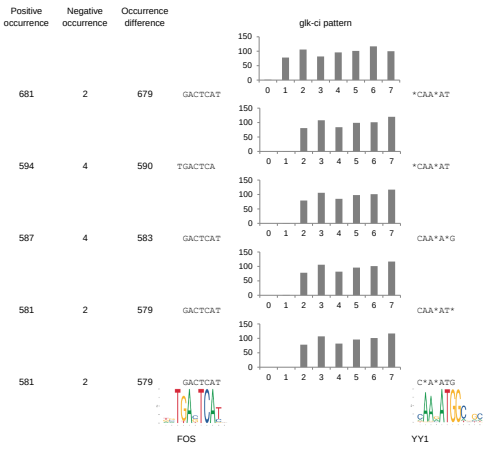
(b) FOS-MAX



(c) USF1-YY1



(d) STAT1-MAX



(e) FOS-YY1

Fig. S1. Top 5 glk-ci patterns with strongest differential occurrence between the positive and negative sequences for TF pairs (a) ELF1-USF1, (b) FOS-MAX, (c) USF1-YY1, (d) STAT1-MAX and (e) FOS-YY1, based on the simulated data. The distributions show the numbers of supporting positive sequences with respect to the size of the indel region of the glk-ci patterns. The parameters of glk-ci-SVM were set to g=2, l=7 and k=12. The corresponding Jaspar motifs are shown at the bottom for comparisons.

S2.2 Effects of parameter values

Figure S2 shows that the maximum difference between the occurrence counts of a glk-ci pattern in the positive and negative sequences correlates strongly with the classification performance.

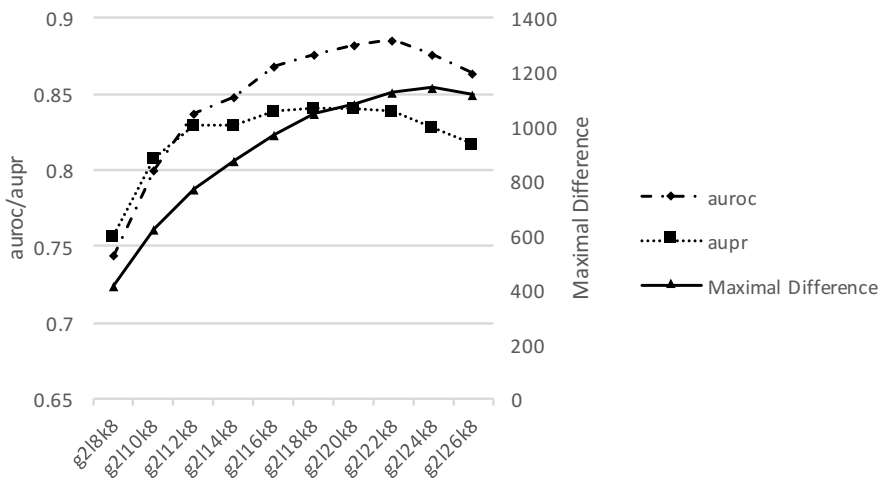


Fig. S2. Maximal difference between glk-ci pattern occurrence counts in the positive and negative sequences across different value of parameter l for the TF pair CUX1 and NHLH1.

We then investigated how the SVM performance was affected by the parameter setting. For the TF pair FOS-MAX, the two motifs had 11 and 10 positions, respectively. When the spacing range was set to [1,7], the total span of the two motifs and the spacing varied between 22 and 28 positions. For gkm-SVM, the values of tot considered and the resulting performance exhibited no clear relationships (Figure S3a). For glk-ci-SVM, we tried three sets of values for g and l, namely (2,7), (3,7) and (2,10). We used small values of g since they would require less running time, and it turned out that these small values were sufficient for achieving very high performance. We further tried several more combinations of these two parameters to see how the performance of glk-ci-SVM would change, and found that the performance was generally higher with a larger value of l (Figure S3b), suggesting that glk-ci patterns that modeled the negative sequences were also useful for the classification.

S2.3 Simulated dataset with more negative sequences than positive sequences

We generated an imbalanced simulated dataset. For each TF pair, the dataset contains 500 positive sequences and 2000 negative sequences. The motif distances for the positive sequences were drawn uniformly from the range [1, 15], and the distances for the negative sequences were drawn from [16, 30]. We set the value of g to 2, k to 10, and the value of l is the same as in the corresponding balanced dataset. Figure S4 shows the performances of glk-ci SVM on this imbalanced dataset.

S2.4 More parameter settings for the simulated dataset

For the experiments in the main text, we set g to 2 and k to 10. To test the performance when $g + k$ is different from the expected motif length, we also tested some additional values for the simulated dataset. From the results (Figure S5), the performance remained highly similar to the other parameter settings.

S2.5 Runtime scalability with respect to dataset size

To evaluate the scalability of glk-ci-SVM (the version disallowing indels at pattern ends) with respect to the dataset size, we randomly selected one ENCODE ChIP-seq dataset, and then generated four sub-datasets of different sizes (containing 2000, 4000, 6000 or 8000 random peak sequences each of length 101bp). We set $g = 2, l = 5, k = 6$, and used 32 threads. Figure S6 shows the running time of glk-ci SVM on thsee four datasets.

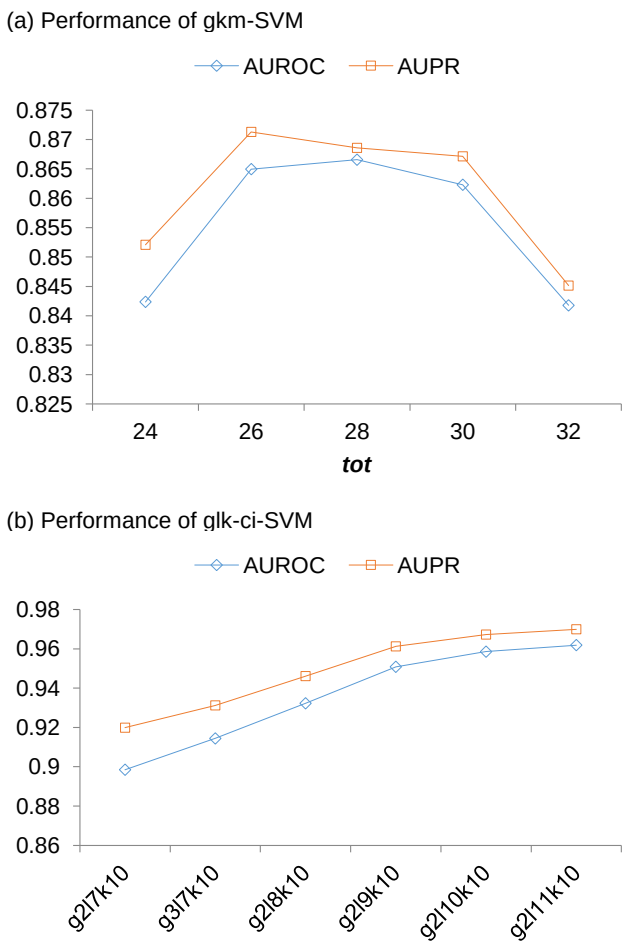


Fig. S3. Performance of gkm-SVM (a) and glk-ci-SVM (b) with different parameter settings based on the simulated data for the FOS-MAX pair.

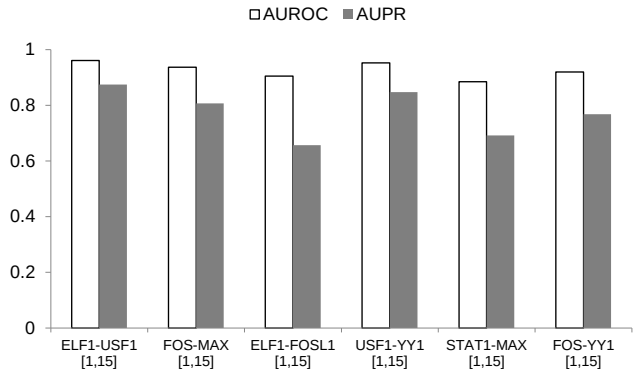


Fig. S4. The AUROC (white) and AUPR (grey) results of glk-ci SVM on the imbalanced dataset.

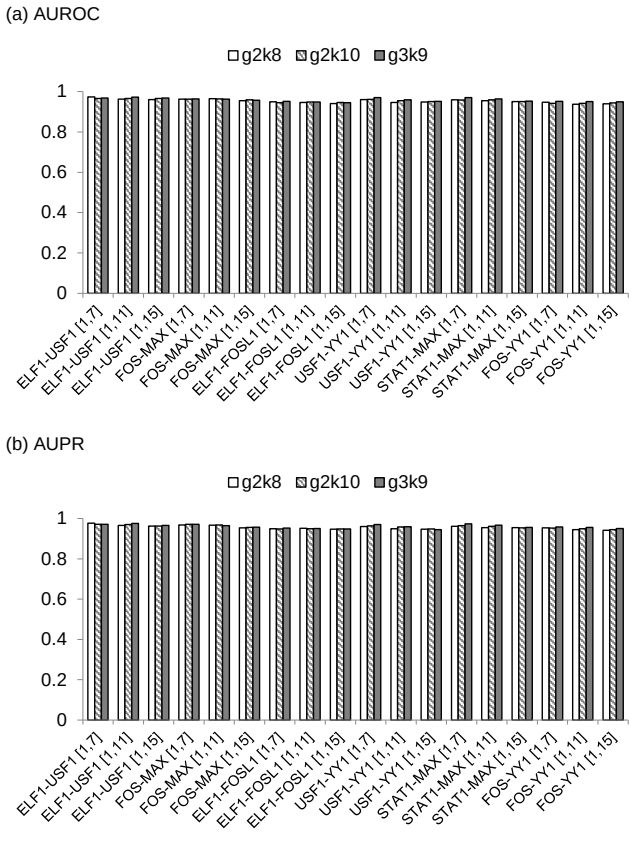


Fig. S5. The AUROC (a) and AUPR (b) results of additional parameter settings of g and k based on the simulated dataset.

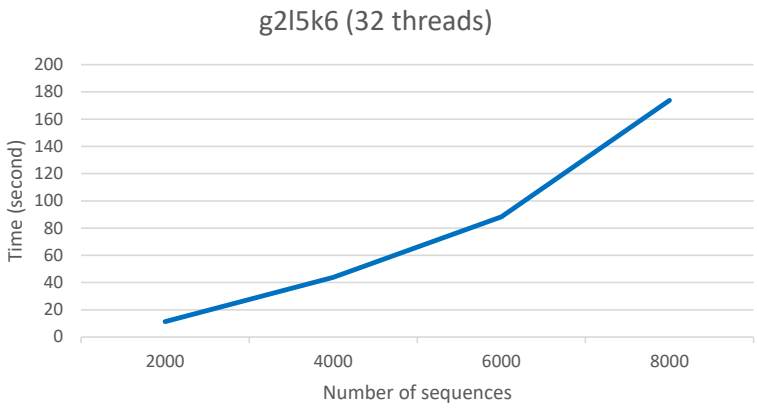


Fig. S6. Running time of glk-ci-SVM on the four benchmark datasets with increasing size.