Supplementaray Document for GaKCo: a Fast Gapped *k*-mer string Kernel using Counting

Ritambhara Singh, Arshdeep Sekhon, Jack Lanchantin,
Kamran Kowsari, Beilun Wang and Yanjun Qi

Department of Computer Science, University of Virginia (yanjun@virginia.edu)

S:1 More Details of GaKCo Algorithm

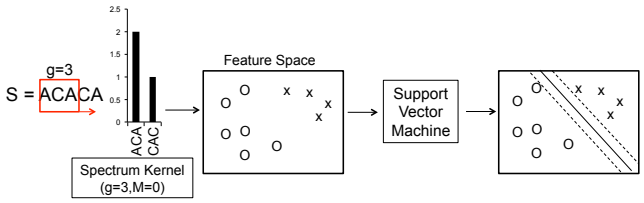

Fig. S:1. Overview of the String (Spectrum) Kernel + SVM classifier.

S:1.1 Formal proof regarding Hamming Distance Property

Let hamming distance between strings x and y be $d(x, y)$. Assuming both x and y are composed of n characters, then hamming distance is formally defined as [2]:

$$d(x, y) = \Sigma_{i=0}^{n} neq(x_i, y_i) \tag{S:1–1}$$

where, if a and b are two characters,

$$neq(a, b) = \begin{cases} 0, & \text{if } a = b \\ 1, & \text{otherwise} \end{cases} \tag{S:1–2}$$

Property: Given, there are two strings x and y (composed of n characters each) and characters from p positions are removed to obtain strings x' and y' with $(n-p)$ characters. If the hamming distance between x' and y', $d(x', y') = 0$ then the hamming distance between original x and y, $d(x, y) \leq p$.

Proof by example:

Let $p = 2$. We first re-write Eq. S:1–1 as:

$$d(x,y) = \Sigma_{i=0}^{n-2} neq(x_i, y_i) + neq(x_{n-1}, y_{n-1}) + neq(x_n, y_n) \quad \text{(S:1–3)}$$

That is, we split the summation of $neq(.)$ function as summation of $neq(.)$ for $(n-2)$ characters plus the sum of $neq(.)$ for the $(n-1)^{th}$ and last n^{th} character for x and y.

The term $\Sigma_{i=0}^{n-2} neq(x_i, y_i)$ represents the hamming distance $d(x', y')$ for $p = 2$ positions removed. Therefore:

$$d(x,y) = d(x', y') + neq(x_{n-1}, y_{n-1}) + neq(x_n, y_n) \quad \text{(S:1–4)}$$

Now if $d(x', y') = 0$ then

$$d(x,y) = neq(x_{n-1}, y_{n-1}) + neq(x_n, y_n) \quad \text{(S:1–5)}$$

Based on Eq. S:1–2, $d(x,y) = \{(0+0), (0+1), (1+0), (1+1)\}$ as these are all the possible values of $neq(.)$ function.

Therefore, $d(x,y) \leq 2$ if $d(x', y') = 0$ where x' and y' are x and y (respectively) with characters removed from $p = 2$ positions.

S:1.2 Justification of GaKCo's Sort and Count Method

A core piece of the GaKCo's kernel computation is counting the observed g-mers in the strings for which the kernel value is being computed. The final implementation of our algorithm uses a sorting-based counting method, but we did consider a hashing approach as well. There are straightforward time complexity justifications for choosing sorting over hashing, which we explain in this section.

A hash table, treated as an associative array, could easily be used to count instances of a g-mer. Given a g-mer, which consists of a g-length token and a reference to the original string number, we may write a simple hash function that executes in $\Theta(g)$ time (as we ought to consider every character in the string for a well-distributed hash). Also, given that the total number of strings is N of average length l, then the total number of g-mers is $\sim Nl$. If we accept the "typical-case" runtime of insertion into a hash table, which is $\Theta(1)$, then to count every g-mer we must perform at least $\Theta(g \cdot Nl)$ steps: for each of the total Nl g-mers, we do g work to hash, insert, and update the associated value.

At first consideration, a sorting-based approach would seem to be strictly worse, as any swapping sort would take $\Theta(g \cdot (Nl) \lg(Nl))$ time. However, using a non-swapping sort, in our case, gives us $\Theta(g \cdot Nl)$ time, which is the same as we derived for the above hashing method. However, the sorting requires nearly exactly $g \cdot Nl$ steps, while the hashing approach needs more steps to resolve any possible collisions. To confirm our theoretical justification, we implemented hashing approach and found that our sorting method was, indeed, faster than hashing.

S:2 Connecting to Previous Studies

String Kernels Aside from the spectrum kernel [14] and gapped k-mer kernel [8], a few other notable string kernels include (but are not limited to): (1) (k, m)-*Mismatch Kernel.* This kernel calculates the dot product of contiguous k-mer counts with m mismatches allowed. The cumulative matching statistic idea was first proposed by [11] for this kernel. [1] (2) *Substring Kernel.* It measures the similarity between sequences based on common co-occurrence of exact matching subpatterns (e.g., substrings) [24]. (3) *Profile Kernel.* This method uses the notion of similarity based on a probabilistic model (e.g. profile) [10]. (4) *Cluster Kernel.* The "sequence neighborhood" kernel or "cluster" kernel [6] is a semi-supervised extension of the string kernel. It replaces every sequence with a set of "similar" (neighboring) sequences and obtains a new representation. Then, it averages over the representations of these contiguous sequences found in the unlabeled data using a sequence similarity measure.

All string kernels calculate the feature representation $\phi(\cdot)$ using the counts of k-mer occurrence. Thus, in the following paragraph we will briefly discuss notable methods that count the occurrence of k-mers (mostly in the bioinformatics literature).

k-mer counting methods k-mer (or in our case, g-mer) counting is the method by which we determine the number of matching or unique k-mers (or g-mers) in any text or pattern. Tools handling large text datasets need to filter out these unique k-mers (or g-mers) to reduce the processing or counting time. GaKCo uses a 'sort and count' method for calculating the number of matching g-mers to compute the mismatch profile. This is a widely used method that lists all the g-mers, sorts them lexicographically and counts all the consecutive matching entries while skipping the unique g-mers. It has been used previously in tools used for genome assembly [17], discovery of motifs (or most common fixed length patterns) [20], and string kernel calculation [11].

BioSequence Classification with Deep Learning In recent years, deep learning models have become popular in the bioinformatics community, owing to their ability to extract meaningful representations from large labeled datasets (e.g., with sample size ∼30,000 sequences). For example, Qi et al. [19] used a deep multi-layer perceptron (MLP) architecture with multitask learning to perform sequence-based protein structure prediction. Zhou et al. [26] created a generative stochastic network to predict secondary structure on the same data as used by Qi et al. [19]. Recently, Lin et al. [15] outperformed all the state-of-the-art works for protein property prediction task by using a deep convolutional neural network architecture. Later, Alipanahi et al. [1] applied a convolutional neural network model for predicting sequence specificity of DNA and RNA-binding proteins as well as generating motifs, or consensus patterns, from the features that

[1] Because [11] uses all possible k-mers built from the dictionary with m mismatches as the feature space, the authors [11] need to precompute a complex weight matrix to incorporate all possible k-mers with m mismatches.

were learned by their model. Lanchantin et al. [13] proposed a deep convolutional/highway MLP framework for the same task and demonstrated improved performance. In the field of natural language processing, multiple works like [23] have used deep learning models for document [25] or sentiment [22] classification.

S:3 Details about the Datasets

S:3.1 Benchmark Tasks of Sequence Classification

DNA and Protein Sequence Classification Studying DNA and Protein sequences gives us deeper insight into the biological processes that can, in turn, help us understand cell development and diseases. Two major tasks essential in the field are Transcription Factor Binding Site (TFBS) Prediction and Remote Protein Homology Prediction.

Transcription factors (TFs) are regulatory proteins that bind to functional sites of DNA to control the regulation of genes. Each different TF binds to specific locations (or sites) on a genomic sequence to regulate cell machinery. Owing to the development of chromatin immunoprecipitation and massively parallel DNA sequencing (ChIP-seq) technologies [18], maps of genome-wide binding sites are currently available for multiple TFs across different organisms. Because ChIP-seq experiments are slow and expensive, computational methods to identify TFBSs accurately are essential for understanding cell regulation.

Remote Protein Homology Prediction, i.e. classification of protein sequences according to their biological function or structure, plays a significant role in drug development. Protein sequences that are a part of the same protein superfamily are evolutionally related and functionally and structurally relevant to each other [3]. Protein sequences with feature patterns showing high homology are classified into the same superfamily. Once assigned a family, the properties of the protein can be easily narrowed down by analyzing only the superfamily to which it belongs.

Researchers have formulated both these tasks as classification tasks, where knowing a DNA or protein sequence, we would like to classify it as a binding site or non-binding site for TF prediction and belonging or not belonging to a protein family for homology prediction respectively.

Text Classification Text classification incorporates multiple tasks like assigning subject categories or topics to documents, spam detection, language detection, sentiment analysis, etc. Generally, given a document and a fixed number of classes, the classification model has to predict the class that is most relevant to that document. Several recent studies have discovered that character-based representation provides straightforward and powerful models for relation extraction [21], sentiment classification [25], and transition based parsing [4]. Lodhi et. al. [16] first used string kernels with character level features for text categorization. However, their kernel computation used dynamic programming which was computationally intensive. Over recent years, more efficient string kernel methods

have been devised [14, 10–12, 8]. Therefore, we use simple character-based text input for document and sentiment classification tasks.

S:3.2 19 Datasets used in Evaluations

ENCODE DNA Sequences: Transcription factors (TFs) are regulatory proteins that bind to functional sites of DNA to control the regulation of genes. Each different TF binds to specific locations (or sites) on a genomic sequence to regulate cell machinery. Maps of genome-wide binding sites are currently available for multiple TFs for human genome via the ENCODE [7] database. These "maps" mark the positions of the TF binding sites. We select 100 basepair sequences overlapping the binding sites as positive sequences and randomly select non-binding sequences from the human genome as negative sequences. We perform this selection for five different transcription factors (CTCF, EP300, JUND, RAD21, and SIN3A) from the K562 (leukemia) cell type, resulting in five different prediction tasks. For these tasks, we use a dictionary size of 5 ($\Sigma = 5$); four nucleotide symbols — A, T, C, G — and a "unknown" character 'N' for nucleotides that were not read by the sequencing machines. Therefore, the dictionary is {A,T,C,G,N}.

SCOP Protein Sequences: Remote Protein Homology Prediction, i.e. classification of protein sequences according to their biological function or structure, plays a significant role in drug development. Protein sequences that are a part of the same protein superfamily are evolutionally related and functionally and structurally relevant to each other [3]. The SCOP domain database consists of protein domains, no two of which have 90% or more residual identity [9]. It is hierarchically divided into folds, superfamilies, and finally families. We use 12 sets of samples (listed in Supplementary Table) and select positive test sequences (for each sample) from 1 protein family of a particular superfamily. We obtain the positive training sequences from remaining families in that superfamily. We select negative training and test sequences from non-overlapping folds outside the positive sequence fold. We use the dictionary size of 20 ($\Sigma = 20$) as there are 20 amino acid symbols that make up a protein sequence. Therefore the dictionary is {A,C,D,E,F,G,H,I,K,L,M,N,P,Q,R,S,T,V,W,Y}.

WebKB and Sentiment Classification Datasets: Text classification incorporates multiple tasks like assigning subject categories or topics to documents, spam detection, language detection, sentiment analysis, etc. Several recent studies have discovered that character-based representation provides straightforward and powerful models for relation extraction [21], sentiment classification [25], and transition based parsing [4]. We downloaded the processed WebKB datasets (removed stop/short words, stemming, etc.) from [5]. This task is a multi-class classification task of webpages with four classes: project, course, faculty, and student. For the sentiment analysis experiments, we used the Stanford sentiment treebank dataset [22]. This dataset provides a score for each sentence between 0 and 1 with $[0, 0.4]$ being negative sentiment and $[0.6, 1.0]$ being positive. We combined the validation set in the original treebank dataset with the training set. We use the dictionary size of 36 ($\Sigma = 36$) since we use character-based input. The dictionary includes all the alphabets [A-Z] and numbers [0-9].

S:4 Empirical Performance of GaKCo versus Neural Networks

Recently, Deep Neural Networks (NNs) have provided state-of-the-art performances for various sequence classification tasks like analyzing DNA [1, 13], proteins [19, 26], and natural language [25, 22] sequences. Despite their superior performance in accuracy and speed (e.g. through GPUs and mini-batches) such NN systems usually require a significant number of training samples. This requirement can be unfeasible for many datasets, especially in the medical research domains. Here, the number of training sequences per experiment can be as low as tens or hundreds due to cost and time constraints. We compare GaKCo's empirical performance with a state-of-the-art deep convolutional neural network (CNN) model [13]. On datasets with few training samples, GaKCo achieves an average accuracy improvement of 20% over the CNN model (see Fig. S:2) making it an appealing tool when the training samples are scarce. Besides, GaKCo includes only two hyperparameters (g and k) for tuning [2]. This feature is desirable when comparing with NN systems for which figuring out the optimal network model and hyperparameters can be a daunting task.

More concretely, we compare GaKCo's empirical performance with a state-of-the-art CNN model from [13]. Fig. S:2 (b) shows the differences in AUC Scores (or micro-averaged F1-score for Web-KB) of GaKCo and CNN [13]. For 16/19 tasks, GaKCo outperforms the CNN model with an average of $\sim$ 20% accuracy. This result can be explained by the fact that CNNs trained with a small number of samples (1000-10,000 sequences) often exhibit unstable behavior in performance.

For three datasets - SIN3A (DNA), 1.1 (protein), and Web-KB (text), we observe that the empirical performance of GaKCo and CNN is similar. Therefore, we further explore these datasets in Fig. S:2(c). Here, we plot the AUC scores or micro-averaged F1 scores (Web-KB) for varying number of training sample ($N = \{100, 250, 500$ and $750\}$ sequences). We randomly select these samples from the training set and use the original test set of the respective datasets. The results are averaged over three runs of the experiment. Our aim is to find the threshold (number of training samples) for which CNN gives a lower performance to GaKCo for these three datasets. Fig. S:2(c) presents the averaged AUC scores or micro-averaged F1 score (Web-KB). We see that the threshold for which CNN gives a lower performance to GaKCo is 750 sequences in the training set. We also observe that the variance in performance is high for NN (represented by error bars) across the three runs.

[2] There is also one C parameter for tuning SVM training (while using linear kernel)

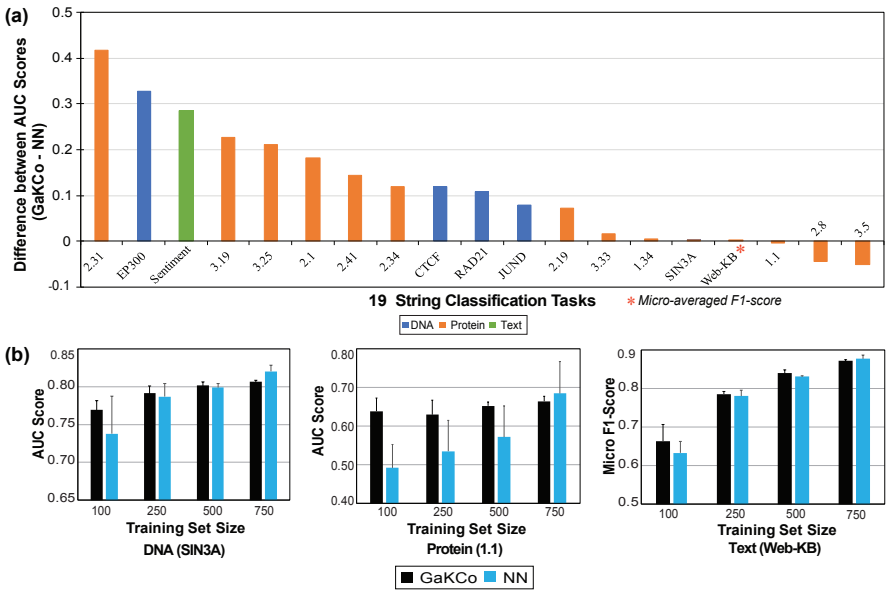

Fig. S:2. (a) Kernel calculation time (X-axis) and the memory usage (Y-axis) (lower is better) for both GaKCo and gkm-SVM for all 19 classification tasks. For 17/19 tasks, GaKCo's memory usage is lesser or comparable to gkm-SVM with lower kernel calculation time. Therefore, it is both time and memory efficient. (b) Differences in AUC Scores (or micro-averaged F1-score for Web-KB) between GaKCo and state-of-the-art CNN model [13]. For 16/19 tasks, GaKCo outperforms CNN with an average of $\sim 20\%$ accuracy. (c)Averaged AUC scores, across 3 runs, for SIN3A (DNA) and 1.1 (protein), and micro-averaged F1 scores for Web-KB (text) while varying number of sequences ($N = \{100, 250, 500,$ and $750\}$). For a threshold value of 750 sequences in the training set, CNN achieves lower empirical performance to GaKCo.

S:5 Other Experiments

S:5.1 Time profiles for different functions of GaKCo

Table S:1 shows the GaKCo average time profiled for sorting vs. count and update function when calculating cumulative mismatch profile for EP300 DNA dataset ($m = 7$).

Table S:1. The GaKCo sorting time vs Count and Update time averaged over all iterations for EP300 DNA dataset (g=10 and k=3).

Algorithmic Step	Average Time (in seconds)
Sort	0.059
Count and Update	2.26

S:5.2 AUC scores for the best performing parameters

Different handling of dictionaries Table S:2 summarizes the AUC scores for all datasets. The current gkm-SVM implementation [8] while reading the input ignores an unknown character. GaKCo maps it to another 'UNK' character. For example, the dataset may contain an extra 'X' character, which is not a part of the dictionary. To ensure consistent empirical performance between GaKCo and gkm-SVM, the user will have to add the extra 'X' character to the dictionary of gkm-SVM.

Table S:2. Summary of GaKCo-SVM, gkm-SVM and CNN-AUC scores for all datasets. For Web-KB we report the micro-averaged F1-Score since it is a multi classification task with four classes: student, faculty, project and course.

Prediction Task	**Sample properties**			**Best Parameters**			**AUC**		
Datasets	N	Σ	Max(l)	g	k	c	GaKCO-AUC	gkm-SVM-AUC	NN-AUC
1.1	3574	20	905	7	5	0.01	0.7453	0.7448	0.7484
1.34	3312			10	1	0.1	0.9903	0.9903	0.9858
2.19	2560			7	1	100	0.8951	0.8951	0.822
2.31	3500			10	7	10	0.9484	0.9497	0.5317
2.1	7062			10	3	10	0.979	0.9895	0.7970
2.34	2738			7	6	0.01	0.8664	0.8660	0.7477
2.41	2646			10	6	0.01	0.7925	0.7925	0.6484
2.8	2480			10	1	10	0.6367	0.6367	0.6801
3.19	3341			8	1	0.1	0.9326	0.9326	0.7050
3.25	3637			10	8	1	0.7967	0.7962	0.5848
3.33	2918			10	5	0.01	0.9018	0.9018	0.8843
3.50	2549			10	7	0.01	0.7768	0.7772	0.8265
CTCF	4000	5	100	10	5	1	0.902	0.902	0.7834
EP300				10	5	1	0.942	0.942	0.6138
JUND				10	7	1	0.91	0.91	0.8317
RAD21				10	5	1	0.901	0.901	0.7937
SIN3A				10	7	1	0.834	0.834	0.8309
Sentiment	9217	36	260	8	4	1	0.8154	0.81	0.5303
WebKB (F1-score)	4163	36	14218	8	5	1	0.9153	0.9116	0.9147

S:5.3 Running Time Results of GaKCo with one level of parallelization versus GaKCo with two levels of parallelization

As explained earlier, our GaKCo implementation utilizes the parallelizability of GaKCo over iterations over m mismatches. It is possible to parallelize GaKCo on another level as the calculation of the cumulative mismatch profile C_i is also independent for the $\binom{g}{i}$ iterations i.e. the Step 4 in Fig 2 (Main Text) can be done independently over all $\binom{g}{i}$ positions. We also performed similar experiments for a

[4] * indicates memory issues

Table S:3. GaKCo-Parallel (Single Level Parallelization) vs GaKCo-Parallel+ (two levels of parallelization) for WebKB dataset

Levels of Parallelization	Time (seconds)	Memory(GB)
GaKCo-Parallel	751	1.63
GaKCo-Parallel+	58	24.44

Table S:4. GaKCo-Parallel(Single Level Parallelization) vs GaKCo-Parallel+ (two levels of parallelization). The time is in seconds[4]

Dataset	GaKCo-Parallel	GaKCo-Parallel+
1.1	31	13
1.34	266	63
2.19	79	28
2.31	120	17
2.1	974	*
2.34	10	7
2.41	90	19
2.8	184	43
3.19	175	35
3.25	54	15
3.33	151	32
3.50	58	10
WebKB	751	58
Sentiment	522	137

two-level parallelization implementation of GaKCo. As summarized in Table S:3 and Table S:4, the kernel calculation time decreases manifold, but this speed-up comes at the cost of increased memory usage. If memory is not a constraint, our GaKCo-Parallel+ implementation can be used to further speed up kernel calculation.

S:5.4 GaKCo is both time and memory efficient:

Fig. S:3 shows points for the kernel calculation time (X-axis) versus the memory usage (Y-axis) for both GaKCo and gkm-SVM for all 19 classification tasks. We observe that most of these points representing GaKCo lie in the **lower-left quadrant** indicating that it is both time and memory efficient. For 17/19 tasks, its memory usage is lesser or comparable to gkm-SVM with faster kernel calculation time. Therefore, GaKCo's time improvement over the baseline is achieved with almost no added memory cost.

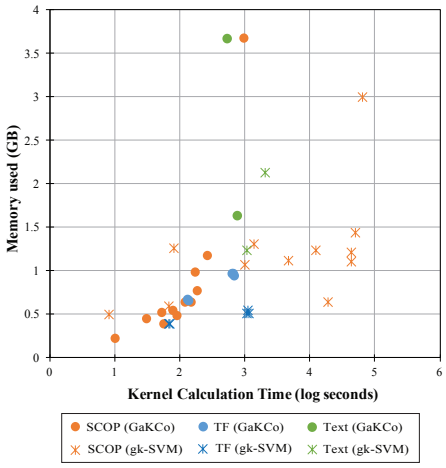

Fig. S:3. Kernel calculation time (X-axis) and the memory usage (Y-axis) (lower is better) for both GaKCo and gkm-SVM for all 19 classification tasks. For 17/19 tasks, GaKCo's memory usage is lesser or comparable to gkm-SVM with lower kernel calculation time. Therefore, it is both time and memory efficient.

References

1. Babak Alipanahi, Andrew Delong, Matthew T Weirauch, and Brendan J Frey. Predicting the sequence specificities of DNA- and RNA- binding proteins by deep learning. *Nature Biotechnology*, 2015.
2. Amihood Amir, Moshe Lewenstein, and Ely Porat. Faster algorithms for string matching with k mismatches. *Journal of Algorithms*, 50(2):257–275, 2004.
3. Pierre Baldi and Søren Brunak. *Bioinformatics: the machine learning approach*. MIT press, 2001.
4. Miguel Ballesteros, Chris Dyer, and Noah A Smith. Improved transition-based parsing by modeling characters instead of words with lstms. *arXiv preprint arXiv:1508.00657*, 2015.
5. Ana Cardoso-Cachopo. Improving Methods for Single-label Text, Categorization. PdD Thesis, Instituto Superior Tecnico, Universidade Tecnica de Lisboa, 2007.
6. Olivier Chapelle, Jason Weston, and Bernhard Schölkopf. Cluster kernels for semi-supervised learning. In *Advances in neural information processing systems*, pages 585–592, 2002.
7. ENCODE Project Consortium et al. An integrated encyclopedia of dna elements in the human genome. *Nature*, 489(7414):57–74, 2012.
8. Mahmoud Ghandi, Dongwon Lee, Morteza Mohammad-Noori, and Michael A Beer. Enhanced regulatory sequence prediction using gapped k-mer features. *PLoS Comput Biol*, 10(7):e1003711, 2014.
9. Tommi Jaakkola, Mark Diekhans, and David Haussler. A discriminative framework for detecting remote protein homologies. *Journal of computational biology*, 7(1-2):95–114, 2000.

10. Rui Kuang, Eugene Ie, Ke Wang, Kai Wang, Mahira Siddiqi, Yoav Freund, and Christina Leslie. Profile-based string kernels for remote homology detection and motif extraction. *Journal of bioinformatics and computational biology*, 3(03):527–550, 2005.
11. Pavel P. Kuksa, Pai-Hsi Huang, and Vladimir Pavlovic. Scalable algorithms for string kernels with inexact matching. In *NIPS'08*, pages 881–888, 2008.
12. Pavel P. Kuksa, Pai-Hsi Huang, and Vladimir Pavlovic. Efficient use of unlabeled data for protein sequence classification: a comparative study. *BMC Bioinformatics*, 10(S-4), 2009.
13. Jack Lanchantin, Ritambhara Singh, Beilun Wang, and Yanjun Qi. Deep motif dashboard: Visualizing and understanding genomic sequences using deep neural networks. *arXiv preprint arXiv:1608.03644*, 2016.
14. Christina Leslie and Rui Kuang. Fast string kernels using inexact matching for protein sequences. *The Journal of Machine Learning Research*, 5:1435–1455, 2004.
15. Zeming Lin, Jack Lanchantin, and Yanjun Qi. MUST-CNN: A multilayer shift-and-stitch deep convolutional architecture for sequence-based protein structure prediction. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI-16)*, 2016.
16. Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, 2(Feb):419–444, 2002.
17. Jason R Miller, Arthur L Delcher, Sergey Koren, Eli Venter, Brian P Walenz, Anushka Brownley, Justin Johnson, Kelvin Li, Clark Mobarry, and Granger Sutton. Aggressive assembly of pyrosequencing reads with mates. *Bioinformatics*, 24(24):2818–2824, 2008.
18. Peter J Park. Chip–seq: advantages and challenges of a maturing technology. *Nature Reviews Genetics*, 10(10):669–680, 2009.
19. Yanjun Qi, Merja Oja, Jason Weston, and William Stafford Noble. A unified multitask architecture for predicting local protein properties. *PloS One*, 7(3):e32235, 2012.
20. Sanguthevar Rajasekaran, Sudha Balla, and C-H Huang. Exact algorithms for planted motif problems. *Journal of Computational Biology*, 12(8):1117–1128, 2005.
21. Ritambhara Singh and Yanjun Qi. Character based string kernels for bio-entity relation detection. *ACL 2016*, page 66, 2016.
22. Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, Christopher Potts, et al. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, volume 1631, page 1642. Citeseer, 2013.
23. Duyu Tang, Bing Qin, and Ting Liu. Document modeling with gated recurrent neural network for sentiment classification. In *EMNLP*, pages 1422–1432, 2015.
24. SVN Vishwanathan, Alexander Johannes Smola, et al. Fast kernels for string and tree matching. *Kernel methods in computational biology*, pages 113–130, 2004.
25. Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657, 2015.
26. Jian Zhou and Olga G Troyanskaya. Deep supervised and convolutional generative stochastic network for protein secondary structure prediction. *arXiv preprint arXiv:1403.1347*, 2014.