FastSK: Fast Sequence Analysis with Gapped String Kernels

Submitted by

RITHIK ALIAS (Reg No: 31120032)

MOHAMED FAWAS T (Reg No: 31120027)

In partial fulfillment of the requirements for the award of Master Of Science in Computer Science with Specialization in Data Analytics Of

Cochin University of Science and Technology, Kochi

Conducted by

Indian Institute of Information Technology and Management-Kerala

Technocity, Pallippuram, Kerala 695302

ACKNOWLEDGEMENT

We would like to express our deepest gratitude to **Dr. Asharaf S**, our project guide for providing us with the necessary facilities for the completion of this project and our mentor **Nikhil V Chandran** for assisting us. We are thankful for the valuable discussions we had at each phase of the project and for being a very supportive and encouraging project guide and project mentor. We would also like to express our sincere and heartfelt gratitude to all people who were in one way or the other involved in my project. A lot of gratitude is reserved for the Director of the Institute for facilitating and nurturing an environment where I have been able to undertake this work. We would like to express my sincere thanks to all friends and classmates whose suggestions and creative criticism.

Names	**Registration Number**
Rithik Alias	**31120032**
Mohamed Fawas T	**31120027**

TABLE OF CONTENT

INTRODUCTION

String Kernel-Support Vector Machines (SK-SVM) are used to achieve strong prediction performance across a variety of sequence analysis tasks, with widespread use in bioinformatics and natural language processing (NLP). It had application in DNA regulatory element identification, and bio-medical named entity recognition.

String kernel methods use simple substring features to compute a similarity function between sequences. The similarity function defines an inner product space, where an SVMclassifier can be trained. The approach easily enables comparison of arbitrary length sequences, obviates sequence alignment issues, captures task-relevant pattern information, and is simpler than other pattern detection tools, such as position-weight matrices.

PREREQUISITES

1. Natural Language Processing

Natural Language Processing is a process of communicating with an intelligent system using the natural language and extracting meaningful data from it to finally represent the data in another form. The two components of NLP are Natural Language Understanding and Natural Language Generation. Natural Language Processing is applied in Sentimental Analysis, language translation applications, Speech Recognition and Information extraction etc.

SVM has been used in NLP for many tasks which resulted in classifying data with utmost accuracy in less time. Here NLP has helped in developing SVM readable inputs. But we are mostly focusing on SVM driven tasks here.

2.Support Vector Machines

SVM is a linear model used for classification and it can solve both linear and non-linear problems. The algorithm creates a hyperplane which separates the data into classes. SVM learn a linear predictive model f(x) =$\hat{y}$=x·w+b. Here x is the input sample features, y is the result output and w is the set of weights used for each feature vector whose linear combination predicts the value of y. Here b is the bias.

SVMs optimize the parameters w by learning a pair of max-margin hyperplanes given by:

$$x \cdot w + b = 1$$

$$x \cdot w + b = -1$$

This is achieved by minimizing the $||w||^2$, So we can maximize the distance between the planes, which is $2/||w||^2$.

A non-linear or kernelized SVM uses a kernel function to compute the pairwise similarities between samples.

In this case, the predictive class of a sample x is given by,

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i K(x_i, x) + b$$

Where x_i and y_i are the i^{th} training sample and its label, respectively. Each α_i is a weight, where if α_i != 0, α_i corresponds to a support vector and b is a learned additive bias.

3. K-mer

It refers to all subsequences of a string or sequence of length k. For example the sequence AGAT would have four monomers (A, G, A, and T), three 2-mers (AG, GA, AT), two 3-mers (AGA and GAT) and one 4-mer (AGAT). More generally, a sequence of length L will have L-k+1 k-mers and c^k total possible k-mers, where n is the number of possible monomers.

4. Gapped K-mer

A gapped k-mer refers to a subsequence containing k letters and m gaps. Hence the total length of the sub string with gaps is g = k + m. For example, A*AG*T is a gapped 4-mer containing 2 gaps (* is used to denote a gap).

5. String Kernels

String kernel methods compare arbitrary length sequences by mapping them into a fixed inner product feature space. the spectrum kernel function Ks provides a similarity score of two sequences x and y as the inner product of their spectrum feature vectors:

$$K_S(x, y) = \langle \phi_S(x), \phi_S(y) \rangle = \sum_{\alpha \in \Sigma^k} c_x(\alpha) c_y(\alpha)$$

Here S is the set of all strings composed from the alphabet Σ. φ_s maps x to a vector indexed by all possible length-k substrings from Σ_k . $c_x(\alpha)$ and $c_y(\alpha)$ return the counts of k-mer α in sequences x and y, respectively.

6. (k,m)-mismatch Kernel

The (k, m)-mismatch kernel retains the k parameter for substring lengths, while adding an m parameter to denote a number of mismatches permitted when comparing the k-mers of a pair of sequences. It permits up to m mismatches when determining if a pair of k-mers should contribute to the

similarity of their respective sequences. Under the (k, m)-mismatch feature map, a string x is mapped to a $|\Sigma|^k$ -dimensional space by,

$$\phi_{(k,m)}(x) = \left(\sum_{\alpha \in x} I_m(\alpha, \gamma) \right)_{\gamma \in \Sigma^k}$$

where $I_m(\alpha, \gamma) = 1$ if the kmer γ is in the "mismatch neighborhood" of α, denoted by $N_{k,m}(\alpha)$.

And the kernel function is given by,

$$K_{(k,m)}(x, y) = \sum_{\alpha \in \Sigma^k} c_x(\alpha; m) c_y(\alpha; m)$$

$\phi_{(k,m)}(x)$ is simply a count of how many times the ith possible k-mer occurs in x if we allow up to m mismatches. Intuitively, the similarity of sequences x and y is given by how many "neighboring" k-mers they share. Following this intuition, the trick is to compute the kernel function by counting how many k-mers from sequence x are contained in the mismatch neighborhoods of sequence y's k-mers.

This algorithm also have shortcomings, First, because the feature space is of size $|\Sigma|^k$, operating in this space becomes deleterious for even moderately sized Σ or k. Second, this is an extremely sparse feature space. Third, most implementations use trie-based data structures, which also grow exponentially with Σ and k. Hence we go for Gapped k-mer kernel.

LITERATURE REVIEW

Gapped k-mer kernels with Support Vector Machines (gkm-SVMs) have achieved strong predictive performance on regulatory DNA sequences on modestly-sized training sets. However, existing gkm-SVM algorithms suffer from

- Existing gkm-SVM kernels suffer from slow kernel computation time, as they depend exponentially on the sub-sequence feature length, number of mismatch positions and the task's alphabet size.
- Counting based methods rely on complex "mismatch statistics" to indirectly obtain feature counts.
- Deep learning models that need small scale datasets such as LSTM and character level CNNs are also trailing back with gkm-SVM in terms of performance and AUC score.

In this work by Derrick Blakely, Eamon Collins, Ritambhara Singh and Yanjun Qi, they introduced a fast and scalable algorithm for calculating gapped k-mer string kernels. The method, named FastSK , uses a simplified kernel formulation that decomposes the kernel calculation into a set of independent counting operations over the possible mismatch positions. This simplified decomposition allowed them to devise a fast Monte Carlo approximation that rapidly converges.

On 10 DNA transcription factor binding site (TFBS) prediction datasets, FastSK consistently matches or outperforms the state-of-the-art gkmSVM-2.0 algorithms in AUC, while achieving average speedups in kernel computation of ~ 100× and speedups of ~ 800× for large feature lengths. It is shown that FastSK outperforms character-level recurrent and convolutional neural networks across all 10 TFBS tasks. They then extend FastSK to 7 English language medical named entity recognition datasets and 10 protein remote homology detection datasets. FastSK consistently matches or outperforms these baselines.

ABSTRACT

Existing baseline models using gkm-SVM algorithms suffer from slow computation time since they depend exponentially on the sub sequence feature lengths. Whereas gapped k-mer kernels with Support Vector Machines (gkm-SVMs) have achieved strong predictive performance on regulatory DNA sequences on modestly-sized training sets.

_____A fast and scalable algorithm for calculating gapped k-mer string kernels is FastSK. It can scale to much greater feature lengths, allows us to consider more mismatches, and is performant on a variety of sequence analysis tasks.

The 3 main advantages of this algorithm on comparing with the baseline models are :

- The algorithm have a feature set that is not exponential in the alphabet size $|\Sigma|$.
- It is a fast kernel computation algorithm that is scalable in Σ, and conceptually simple.
- It scales to greater feature lengths and numbers of mismatches.

The backend of this application is built in C++ with a python interface. This algorithm is available as a Python package and as C++ source code. They bind the C++ backend and python interface using Pybind11 library.

MODEL DESCRIPTION

FastSK-exact Model architecture

In FastSK, our kernel function is determined by the co-occurrences of k-mers, except in our case the k-mers are not contiguous features. The gapped k-mer string kernel function is given by

$$K_{GSK}(x,y) = \sum_{\gamma \in \Theta_{g,m}} c_x(\gamma) c_y(\gamma)$$

Here $\Theta_{g,m}$ is the set of gapped k-mers with m mismatch positions appearing in the dataset. Unlike the spectrum methods, here don't consider the entire feature space. Also the function $c_x(\gamma)$ gives the count of gapped k-mer γ in x.

Then they decomposed the above equation into a summation of multiple independent counting operations, where each operation handles a combination of mismatch positions. Our kernel function is given by:

$$K_{FSK}(x,y) = \sum_{i=1}^{gCm} \sum_{\gamma \in \Theta_i} c_x(\gamma) c_y(\gamma)$$

Here Θ_i denotes the set of gapped k-mers induced by the ith combination of m mismatch positions.

Algorithm

Require: L, g, k (L=matrix of all g-mers from the dataset)

1: **procedure** CALCULATEKERNEL(L,g,k)

2: M ← g − k

3: N ←MISMATCHPROFILE(L,g,M)

4: K ← 0

5: **procedure** MISMATCHPROFILE(L,g,M)

6: $n_{pos} \leftarrow {}^{g}C_{m}$

7: **for** i : 0 ⟶ n_{pos} **do**

8: $P_i \leftarrow 0$

9: $L^i \leftarrow$ removePosition(L, i)

10: $L^i \leftarrow$ sort(L^i)

11: $P_i \leftarrow$ countAndUpdate(L^i)

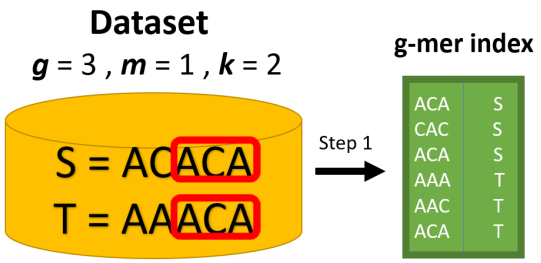
12: **for** i : 0 ⟶ n_{pos} **do**

13: $K \leftarrow K + P_i$

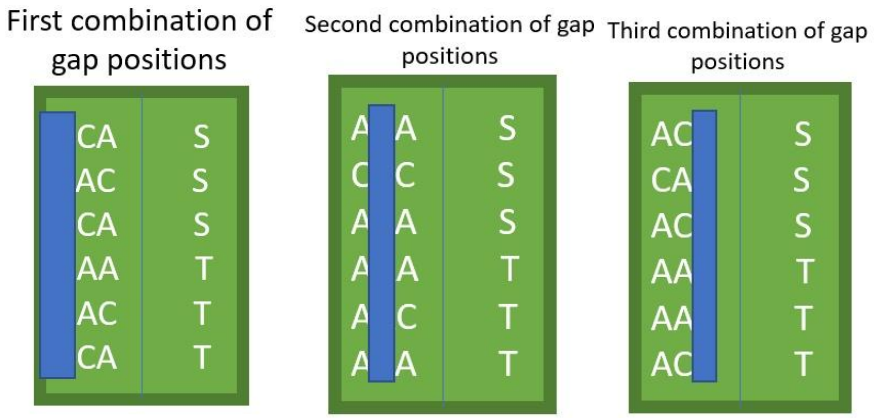
return K

The task of computing kernel function for 2 example sequences using the algorithm is explained the following steps:

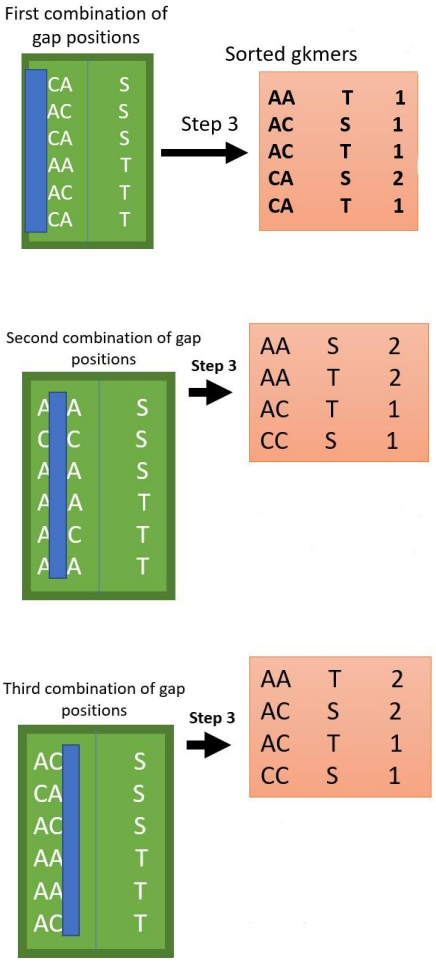
1. Let the datasets be S = ACACA and T = AAACA
2. First, we extract all g-mers from the dataset and store them in a g-mer index table.



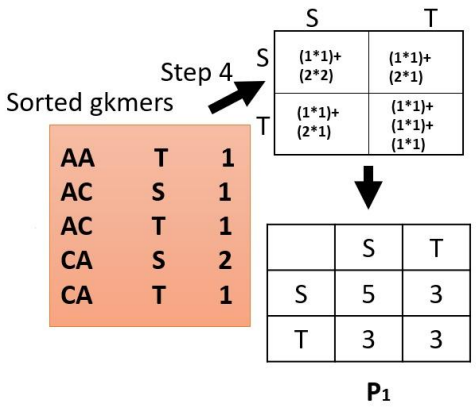
3. Then for each of the ${}^{g}C_{m}$ combinations(for m=1) of mismatch positions, we remove mismatch positions from the g-mers to obtain a set of g kmers.



4. Then we sort the g kmers and count when they are shared in common between pairs of sequences.



5. If a g kmer γ occurs in both sequences x and y, the product $c_x(\gamma)^* c_y(\gamma)$ is stored in a partial kernel matrix P_i in the corresponding cell for i = 1,2,...,${}^{g}C_m$.



AA	S	2
AA	T	2
AC	T	1
CC	S	1

Step 4

	S	T
S	5	4
T	4	5

P_2

AA	T	2
AC	S	2
AC	T	1
CC	S	1

Step 4

	S	T
S	5	2
T	2	5

P_3

Denoting P_i as a function, the partial similarity score of x and y is given by:

$$P_i(x, y) = \sum_{\gamma \in \Theta_i} c_x(\gamma) c_y(\gamma)$$

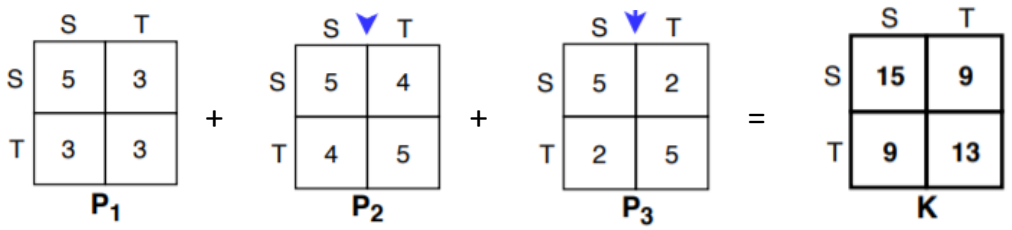
Here they computed each P_i independently. This way the algorithm is easy to parallelize and easy to approximate using random sampling.

6. Once each P_i for $i \in \{1,2,\ldots,{}^{g}C_m\}$ is computed, the full kernel matrix is given by:

$$K_{FSK} = \sum_{i=1}^{gCm} P_i$$

So in the example problem, full kernel matrix K can be computed as,

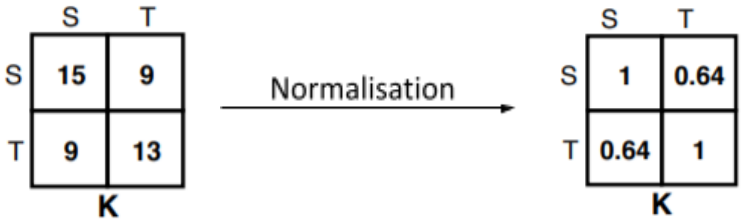
$P_1 + P_2 + P_3 =$



7. Finally, they normalized the kernel matrix using

$$K_{FSK}(x,y) \leftarrow \frac{K_{FSK}(x,y)}{\sqrt{K_{FSK}(x,x)K_{FSK}(y,y)}}$$

for a pair of sequences(x,y). For the given example problem it can be done as



FastSK-Approx (via fast Monte Carlo Approximation)

FastSK -Exact runs with a coefficient of ${}^{g}C_{m}$, it is exponential in g and m. As such, it is unable to handle features roughly of size g > 15. However, many TF binding sites are up to 20 base pairs. Even increasing gmer length is not optimal, a thorough grid-search must include large values of g in the search space in order to rule them out. . To solve this problem, they introduced a Monte Carlo approximation algorithm called FastSK-Approx. ox. FastSK-Approx is extremely fast even for large values of g, as it requires only a small random subset of the ${}^{g}C_{m}$ partial kernels P_i. It is roughly O(1) with respect to g.

To compute FastSK -Approx, they sampled possible mismatch combinations for up to $I_{max} \leq {}^{g}C_{m}$ iterations. That is, at iteration $1 \leq t \leq I_{max}$, we randomly sample (without replacement) a mismatch combination $i \leftarrow {}^{g}C_{m}$ and compute the corresponding partial kernel matrix P_i . They then computed the online mean kernel matrix $\overline{K}^{(t)}$ using P_i and $\overline{K}^{(t-1)}$. Furthermore, they computed

a matrix of online standard deviations corresponding to the entries of $\overline{K}^{(t)}$ and use the average of these values, which is denoted as σ (t) , to satisfy a convergence condition. Convergence is achieved when there is an approximately 95% probability that the online sample mean kernel $\overline{K}^{(t)}$ is within δ units of the true mean kernel matrix μ_K. Here, δ is a user-determined parameter.

According to the central limit theorem, we assume that for sufficiently large t, the sample mean kernel is normally distributed. Now, since its normally distributed we can stardize $\overline{K}^{(t)}$,

$$Z = \overline{K}^{(t)}\text{(standardized)} = \left| \frac{\overline{K}^{(t)} - \mu_K}{\sigma^{(t)}} \right|$$

Then, P[Z > 1.96] = 0.05, where 1.96 is the Z-score for a 95% confidence interval. Therefore, the convergence condition is satisfied when $1.96*\sigma^{(t)} < \delta$

It can be shown that FastSK -Approx converges rapidly, even for large values of g or m; it typically converges when t ≈ 50, which roughly corresponds to the number of samples needed to invoke the Central Limit Theorem. Furthermore, this means that FastSK -Approx is roughly O(1) with respect to g.

Datasets Used

- Here all of our datasets are in fasta format. A sequence in fasta format begins with a single-line description, followed by lines of sequence data.
- Here we have evaluated FastSK using 9 DNA sequence based transcription factor binding site classification datasets.

Dataset	Train	Test	Total
CTCF	2000	2000	4000
EP300	2000	2000	4000
JUND	2000	2000	4000
RAD21	2000	2000	4000
SIN3A	2000	2000	4000
Pbde	4500	5500	10000
EP300_47848	6506	724	7230
KAT2B	6318	702	7020
TP53	4432	494	4926

- Then we evaluated FastSK using 10 protein remote homology datasets from the SCOP project.

Dataset	Train	Test	Total
1.10	2339	1235	3574
1.34	2075	1237	3312
2.19	1345	1215	2560
2.31	2298	1202	3500
2.34	1501	1237	2738
2.41	1427	1219	2646
2.80	1241	1239	2480
3.19	2103	1238	3341
3.25	2395	1242	3637
3.33	1680	1238	2918

- Then we evaluated FastSK using 7 English-language medical named entity recognition datasets.

Dataset	Train	Test	Total
AImed	1500	1500	3000
BioInfer	2534	2534	5068
CC1-LLL	3785	330	4115
CC2-IEPA	3298	817	4115
CC3-HPRD50	3682	433	4115
DrugBank	2472	2472	4944
MedLine	635	635	1270

Libraries used

- os - Helps us to automatically perform many operating system tasks.
- re - This is used to work with regular expressions. It allows us to search a string for a match.
- setuptools - This is a package that is used by many other packages to handle their installation from source code .
- pybind11 - It is a lightweight header-only library that exposes C++ types in Python and vice versa. It is used to create Python bindings of existing C++ code.
- agparse - This lets your code accept command line arguments and makes the code easy to configure at runtime.
- Scikit-learn - It contains many tools for machine learning and statistical modeling. Here we have used a Linear SVC module which fits the support vector classifier to fit to the data we provide, returning a "best fit" hyperplane that divides, or categorizes, our data. From the metrics module we use the tools to calculate the ROC and AUC scores.
- numpy - It helps us to work with arrays and is utilized to perform a number of mathematical operations on matrix data structures.
- warnings - This warns the developer of situations that aren't necessarily exceptions.

Training

We basically did our testing on Intel® core™ i3-7130U CPU@2.70GHz x 4.

We were unable to detect the best parameters to use for each dataset using our hardware systems, So we used the best parameters which were already mentioned in the referred paper.

Then we conducted our experiment of running FastSK on 9 DNA sequence based transcription factor binding site classification datasets, 10 SCOP project protein remote homology detection datasets, and 7 medical named entity recognition datasets.

Experimental Evaluation

We got AUC scores and accuracy for each dataset as follows:

- For DNA sequence based transcription factor binding site classification datasets the results are obtained as follows

Dataset	Train	Test	Total	g	m	k	Accuracy	AUC
CTCF	2000	2000	4000	13	7	6	0.915500	0.969637
EP300	2000	2000	4000	10	4	6	0.952000	0.990690
JUND	2000	2000	4000	10	3	7	0.906000	0.967907
RAD21	2000	2000	4000	14	8	6	0.903000	0.969463
SIN3A	2000	2000	4000	8	2	6	0.836000	0.911383
Pbde	4500	5500	10000	5	1	4	0.784364	0.870143
EP300_47848	6506	724	7230	11	5	6	0.908840	0.952840
KAT2B	6318	702	7020	13	7	6	0.849003	0.922054
TP53	4432	494	4926	7	2	5	0.736842	0.810979

- For SCOP project protein remote homology detection datasets the results are obtained as follows

Dataset	Train	Test	Total	g	m	k	Accuracy	AUC
1.10	2339	1235	3574	8	4	4	0.993522	0.832111
1.34	2075	1237	3312	6	2	4	0.997575	1.000000
2.19	1345	1215	2560	8	4	4	0.992593	0.861894
2.31	2298	1202	3500	15	10	5	0.994176	0.999372
2.34	1501	1237	2738	6	0	6	0.995150	0.967506
2.41	1427	1219	2646	10	6	4	0.995078	0.928689
2.80	1241	1239	2480	12	8	4	0.993543	0.873680
3.19	2103	1238	3341	9	2	7	0.994346	0.483405
3.25	2395	1242	3637	15	9	6	0.989533	0.891810
3.33	1680	1238	2918	5	1	4	0.997577	0.995590

- For medical named entity recognition datasets the results are obtained as follows

Dataset	Train	Test	Total	g	m	k	Accuracy	AUC
AImed	1500	1500	3000	11	4	7	0.666667	0.436926
BioInfer	2534	2534	5068	5	4	1	0.621942	0.713633
CC1-LLL	3785	330	4115	5	2	3	0.454545	0.398362
CC2-IEPA	3298	817	4115	5	3	2	0.391677	0.335425
CC3-HPRD50	3682	433	4115	7	4	3	0.369515	0.360191
DrugBank	2472	2472	4944	10	2	8	0.581715	0.636212
MedLine	635	635	1270	5	2	3	0.899213	0.721596

Screenshots of training and testing:

Experimental results of SCOP project protein remote homology detection datasets

1.1

```
(base) axelwitsel@axelwitsel-Vostro-14-3468:~/Music/FastSK/project_tests$ python 1.1
.py
Length of shortest train sequence: 16
Length of shortest test sequence: 20
Dictionary size = 24 (+1 for unknown char).
g = 8, k = 4, 532908 features
Initializing kernel function
Computing approximate kernel...
Computing 70 mismatch profiles using 1 threads...
thread 0 converged in 63 iterations...
Thread 0 finished in 63 iterations...
Linear SVM:
        Acc = 0.9935222672064777, AUC = 0.8321108394458028
```

1.34

```
(base) axelwitsel@axelwitsel-Vostro-14-3468:~/Music/FastSK/project_tests$ python 1.34.
py
Length of shortest train sequence: 16
Length of shortest test sequence: 20
Dictionary size = 24 (+1 for unknown char).
g = 6, k = 4, 443827 features
Initializing kernel function
Computing approximate kernel...
Computing 15 mismatch profiles using 1 threads...
Thread 0 finished in 15 iterations...
Linear SVM:
        Acc = 0.9975747776879548, AUC = 1.0
```

2.8

```
(base) axelwitsel@axelwitsel-Vostro-14-3468:~/Music/FastSK/project_tests$ python 2.8.p
y
Length of shortest train sequence: 16
Length of shortest test sequence: 20
Dictionary size = 24 (+1 for unknown char).
g = 12, k = 4, 376964 features
Initializing kernel function
Computing approximate kernel...
Computing 495 mismatch profiles using 1 threads...
thread 0 converged in 47 iterations...
Thread 0 finished in 47 iterations...
Linear SVM:
        Acc = 0.993543179983858, AUC = 0.8736799350121852
```

2.19

```
(base) axelwitsel@axelwitsel-Vostro-14-3468:~/Music/FastSK/project_tests$ python 2.19.
py
Length of shortest train sequence: 16
Length of shortest test sequence: 20
Dictionary size = 24 (+1 for unknown char).
g = 8, k = 4, 399333 features
Initializing kernel function
Computing approximate kernel...
Computing 70 mismatch profiles using 1 threads...
thread 0 converged in 47 iterations...
Thread 0 finished in 47 iterations...
Linear SVM:
        Acc = 0.9925925925925926, AUC = 0.8618942325409988
```

2.31

```
(base) axelwitsel@axelwitsel-Vostro-14-3468:~/Music/FastSK/project_tests$ python 2.31.
py
Length of shortest train sequence: 16
Length of shortest test sequence: 20
Dictionary size = 24 (+1 for unknown char).
g = 15, k = 5, 567034 features
Initializing kernel function
Computing approximate kernel...
Computing 3003 mismatch profiles using 1 threads...
thread 0 converged in 38 iterations...
Thread 0 finished in 38 iterations...
Linear SVM:
        Acc = 0.9941763727121464, AUC = 0.9993718592964823
```

2.34

```
(base) axelwitsel@axelwitsel-Vostro-14-3468:~/Music/FastSK/project_tests$ python 2.34.
py
Length of shortest train sequence: 16
Length of shortest test sequence: 20
Dictionary size = 24 (+1 for unknown char).
g = 6, k = 6, 470969 features
Initializing kernel function
Computing approximate kernel...
Computing 1 mismatch profiles using 1 threads...
Thread 0 finished in 1 iterations...
Linear SVM:
        Acc = 0.9951495553759094, AUC = 0.9675060926076361
```

2.41

```
(base) axelwitsel@axelwitsel-Vostro-14-3468:~/Music/FastSK/project_tests$ python 2.41.
py
Length of shortest train sequence: 16
Length of shortest test sequence: 20
Dictionary size = 24 (+1 for unknown char).
g = 10, k = 4, 398067 features
Initializing kernel function
Computing approximate kernel...
Computing 210 mismatch profiles using 1 threads...
thread 0 converged in 45 iterations...
Thread 0 finished in 45 iterations...
Linear SVM:
        Acc = 0.9950779327317474, AUC = 0.928689200329761
```

3.19

```
(base) axelwitsel@axelwitsel-Vostro-14-3468:~/Music/FastSK/project_tests$ python 3.19.
py
Length of shortest train sequence: 16
Length of shortest test sequence: 20
Dictionary size = 24 (+1 for unknown char).
g = 9, k = 7, 561566 features
Initializing kernel function
Computing approximate kernel...
Computing 36 mismatch profiles using 1 threads...
thread 0 converged in 9 iterations...
Thread 0 finished in 9 iterations...
Linear SVM:
        Acc = 0.994345718901454, AUC = 0.48340489729604263
```

3.25

```
(base) axelwitsel@axelwitsel-Vostro-14-3468:~/Music/FastSK/project_tests$ python 3.25.
py
Length of shortest train sequence: 16
Length of shortest test sequence: 20
Dictionary size = 24 (+1 for unknown char).
g = 15, k = 6, 580745 features
Initializing kernel function
Computing approximate kernel...
Computing 5005 mismatch profiles using 1 threads...
thread 0 converged in 32 iterations...
Thread 0 finished in 32 iterations...
Linear SVM:
        Acc = 0.9895330112721417, AUC = 0.8918100583413338
```

3.33

```
(base) axelwitsel@axelwitsel-Vostro-14-3468:~/Music/FastSK/project_tests$ python 3.33.
py
Length of shortest train sequence: 16
Length of shortest test sequence: 20
Dictionary size = 24 (+1 for unknown char).
g = 5, k = 4, 434730 features
Initializing kernel function
Computing approximate kernel...
Computing 5 mismatch profiles using 1 threads...
Thread 0 finished in 5 iterations...
Linear SVM:
        Acc = 0.9975767366720517, AUC = 0.9955901125681792
```

Experimental results of DNA sequence based transcription factor binding site classification datasets

CTCF

```
(base) axelwitsel@axelwitsel-Vostro-14-3468:~/Music/FastSK/project_tests$ python CTCF.
py
Length of shortest train sequence: 100
Length of shortest test sequence: 100
Dictionary size = 5 (+1 for unknown char).
g = 13, k = 6, 352000 features
Initializing kernel function
Computing approximate kernel...
Computing 1716 mismatch profiles using 1 threads...
thread 0 converged in 127 iterations...
Thread 0 finished in 127 iterations...
Linear SVM:
        Acc = 0.9155, AUC = 0.969637
```

EP300

```
(base) axelwitsel@axelwitsel-Vostro-14-3468:~/Music/FastSK/project_tests$ python EP300
.py
Length of shortest train sequence: 100
Length of shortest test sequence: 100
Dictionary size = 5 (+1 for unknown char).
g = 10, k = 6, 364000 features
Initializing kernel function
Computing approximate kernel...
Computing 210 mismatch profiles using 1 threads...
thread 0 converged in 127 iterations...
Thread 0 finished in 127 iterations...
Linear SVM:
        Acc = 0.952, AUC = 0.9906900000000001
```

JUND

```
(base) axelwitsel@axelwitsel-Vostro-14-3468:~/Music/FastSK/project_tests$ python JUND.
py
Length of shortest train sequence: 100
Length of shortest test sequence: 100
Dictionary size = 5 (+1 for unknown char).
g = 10, k = 7, 364000 features
Initializing kernel function
Computing approximate kernel...
Computing 120 mismatch profiles using 1 threads...
thread 0 converged in 67 iterations...
Thread 0 finished in 67 iterations...
Linear SVM:
        Acc = 0.906, AUC = 0.9679069999999999
```

RAD21

```
(base) axelwitsel@axelwitsel-Vostro-14-3468:~/Music/FastSK/project_tests$ python RAD21
.py
Length of shortest train sequence: 100
Length of shortest test sequence: 100
Dictionary size = 5 (+1 for unknown char).
g = 14, k = 6, 348000 features
Initializing kernel function
Computing approximate kernel...
Computing 3003 mismatch profiles using 1 threads...
thread 0 converged in 126 iterations...
Thread 0 finished in 126 iterations...
Linear SVM:
        Acc = 0.903, AUC = 0.9694630000000001
```

SIN3A

```
(base) axelwitsel@axelwitsel-Vostro-14-3468:~/Music/FastSK/project_tests$ python SIN3A
.py
Length of shortest train sequence: 100
Length of shortest test sequence: 100
Dictionary size = 5 (+1 for unknown char).
g = 8, k = 6, 372000 features
Initializing kernel function
Computing approximate kernel...
Computing 28 mismatch profiles using 1 threads...
Thread 0 finished in 28 iterations...
Linear SVM:
        Acc = 0.836, AUC = 0.9113829999999998
```

Pbde

```
(base) axelwitsel@axelwitsel-Vostro-14-3468:~/Music/FastSK/project_tests$ python Pbde.
py
Length of shortest train sequence: 112
Length of shortest test sequence: 112
Dictionary size = 5 (+1 for unknown char).
g = 5, k = 4, 4864796 features
Initializing kernel function
Computing approximate kernel...
Computing 5 mismatch profiles using 1 threads...
Thread 0 finished in 5 iterations...
Linear SVM:
        Acc = 0.7843636363636364, AUC = 0.8701425454545455
```

EP300_47848

```
(base) axelwitsel@axelwitsel-Vostro-14-3468:~/Music/FastSK/project_tests$ python EP300
_47848.py
Length of shortest train sequence: 200
Length of shortest test sequence: 200
Dictionary size = 6 (+1 for unknown char).
g = 11, k = 6, 1373700 features
Initializing kernel function
Computing approximate kernel...
Computing 462 mismatch profiles using 1 threads...
thread 0 converged in 307 iterations...
Thread 0 finished in 307 iterations...
Linear SVM:
        Acc = 0.9088397790055248, AUC = 0.9528402673911054
```

KAT2B

```
(base) axelwitsel@axelwitsel-Vostro-14-3468:~/Music/FastSK/project_tests$ python KAT2B
.py
Length of shortest train sequence: 200
Length of shortest test sequence: 200
Dictionary size = 6 (+1 for unknown char).
g = 13, k = 6, 1319760 features
Initializing kernel function
Computing approximate kernel...
Computing 1716 mismatch profiles using 1 threads...
thread 0 converged in 338 iterations...
Thread 0 finished in 338 iterations...
Linear SVM:
        Acc = 0.8490028490028491, AUC = 0.9220542041054861
```

TP53

```
(base) axelwitsel@axelwitsel-Vostro-14-3468:~/Music/FastSK/project_tests$ python TP53.
py
Length of shortest train sequence: 200
Length of shortest test sequence: 200
Dictionary size = 6 (+1 for unknown char).
g = 7, k = 5, 955644 features
Initializing kernel function
Computing approximate kernel...
Computing 21 mismatch profiles using 1 threads...
Thread 0 finished in 21 iterations...
Linear SVM:
        Acc = 0.7368421052631579, AUC = 0.8109787080594666
```

Experimental results of medical named entity recognition datasets.

AImed

```
(base) axelwitsel@axelwitsel-Vostro-14-3468:~/Music/FastSK/project_tests$ python3 AImed.py
Length of shortest train sequence: 59
Length of shortest test sequence: 25
Dictionary size = 56 (+1 for unknown char).
g = 11, k = 7, 536508 features
Initializing kernel function
Computing approximate kernel...
Computing 330 mismatch profiles using 1 threads...
thread 0 converged in 70 iterations...
Thread 0 finished in 70 iterations...
Linear SVM:
        Acc = 0.6666666666666666, AUC = 0.443658
```

BioInfer

```
(base) axelwitsel@axelwitsel-Vostro-14-3468:~/Music/FastSK/project_tests$ python3 BioInfer.py
Length of shortest train sequence: 51
Length of shortest test sequence: 50
Dictionary size = 57 (+1 for unknown char).
g = 5, k = 1, 1037307 features
Initializing kernel function
Computing approximate kernel...
Computing 5 mismatch profiles using 1 threads...
Thread 0 finished in 5 iterations...
Linear SVM:
        Acc = 0.622336227308603, AUC = 0.7125751188726763
```

CC1-LLL

```
(base) axelwitsel@axelwitsel-Vostro-14-3468:~/Music/FastSK/project_tests$ python3 CC1-LLL.py
Length of shortest train sequence: 35
Length of shortest test sequence: 66
Dictionary size = 58 (+1 for unknown char).
g = 5, k = 3, 814942 features
Initializing kernel function
Computing approximate kernel...
Computing 10 mismatch profiles using 1 threads...
Thread 0 finished in 10 iterations...
Linear SVM:
        Acc = 0.44242424242424244, AUC = 0.391382603585072
```

CC2-IEPA

```
(base) axelwitsel@axelwitsel-Vostro-14-3468:~/Music/FastSK/project_tests$ python3 CC2-IEPA.py
Length of shortest train sequence: 51
Length of shortest test sequence: 35
Dictionary size = 58 (+1 for unknown char).
g = 5, k = 2, 814942 features
Initializing kernel function
Computing approximate kernel...
Computing 10 mismatch profiles using 1 threads...
Thread 0 finished in 10 iterations...
Linear SVM:
        Acc = 0.3818849449204406, AUC = 0.3320307177803926
```

CC3-HPRD50

```
(base) axelwitsel@axelwitsel-Vostro-14-3468:~/Music/FastSK/project_tests$ python3 CC3-HPRD50.py
Length of shortest train sequence: 49
Length of shortest test sequence: 53
Dictionary size = 58 (+1 for unknown char).
g = 7, k = 3, 814912 features
Initializing kernel function
Computing approximate kernel...
Computing 35 mismatch profiles using 1 threads...
Thread 0 finished in 35 iterations...
Linear SVM:
        Acc = 0.3672055427251732, AUC = 0.3680981595092025
```

DrugBank

```
(base) axelwitsel@axelwitsel-Vostro-14-3468:~/Music/FastSK/project_tests$ python3 DrugBank.py
Length of shortest train sequence: 21
Length of shortest test sequence: 25
Dictionary size = 56 (+1 for unknown char).
g = 10, k = 8, 787919 features
Initializing kernel function
Computing approximate kernel...
Computing 45 mismatch profiles using 1 threads...
Thread 0 finished in 45 iterations...
Linear SVM:
        Acc = 0.5821197411003236, AUC = 0.6362515840847917
```

MedLine

```
(base) axelwitsel@axelwitsel-Vostro-14-3468:~/Music/FastSK/project_tests$ python3 MedLine.py
Length of shortest train sequence: 36
Length of shortest test sequence: 39
Dictionary size = 56 (+1 for unknown char).
g = 5, k = 3, 206198 features
Initializing kernel function
Computing approximate kernel...
Computing 10 mismatch profiles using 1 threads...
Thread 0 finished in 10 iterations...
Linear SVM:
        Acc = 0.8992125984251969, AUC = 0.7216779772329246
```

Conclusion

1. From the experimental results we can see that FastSK performed very well for all the SCOP project protein remote homology detection datasets. All of them have a very good AUC score and accuracy.
2. From the experimental results we can see that FastSK performed very well for most of the DNA sequence based transcription factor binding site classification datasets except the SIN3A, Pbde,KAT2B, and TP53.
3. From the experimental results we can see that FastSK falls short for NLP datasets which are medical named entity recognition datasets.Accuracy and AUC score is below average for the datasets CC1-LLL, CC2-IEPA and CC3-HPRD50. Among other datasets FastSK performed well for the MedLine dataset.

References

1. Derrick Blakely, Eamon Collins, Ritambhara Singh, Yanjun Qi ,Department of Computer Science,University of Virginia.*FastSK: Fast Sequence Analysis with Gapped String Kernels.* (Main paper published)
2. Christina S. Leslie, Eleazar Eskin, and William Stafford Noble. *The spectrum kernel: A string kernel for svm protein classification.*
3. Ritambhara Singh, Arshdeep Sekhon, Kamran Kowsari, Jack Lanchantin, Beilun Wang, and Yanjun Qi. *GaKCo: A fast gapped k-mer string kernel using counting.*
4. Huma Lodhi,Craig Saunders,John Shawe-Taylor, Nello Cristianini and Chris Watkins. *Text Classification using String Kernels.*
5. gkmSVM: fast and accurate interpretation of nonlinear gapped k-mer SVMs by Avanti Shrikumar, Eva Prakash, Anshul Kundaje.