Efficient Evaluation of Large Sequence Kernels

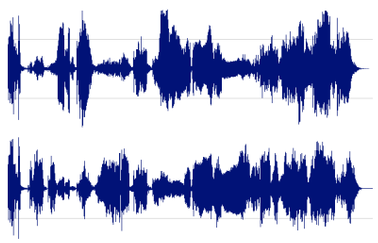
Pavel P. Kuksa, NEC Laboratories America Inc.
Vladimir Pavlovic, Rutgers University

Paper **552**

What are we after?

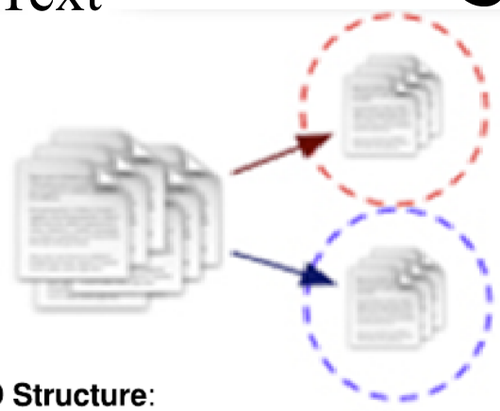
- **Sequence Classification**

Audio/Music



Music Genre
Artist
etc

Text



GTATATGATCAGGACT
AGTTGGTACAGCTCTT
AGACTATTAATTCGAG
CTGAACTAGGCCAAC
CAGGAGCTCTTCTGG
GGGATGATCAATTATA
TAATGTAATTGTAACC

String kernels

Seaquence X Seaquence Y

?

ATGGTTAAATTCTCCT
CAGAACCACATTTGG
CTCAGGTAGTCGCAG
AAGACCTTCTTTCTCC

Bio-informatics

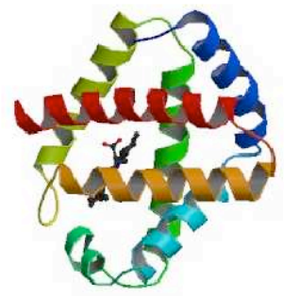
Sequence
VDAAVAKVCGSEAIKANLRRSWGVLSADIEA
TGLMLMSNLFTLRPDTKTYFTRLGDVQKGK
ANSKLRGHAITLTYALNNFVDSLDDPSRLKC
VVEKFAVNHINRKISGDAFGAIVEPMKETLKA
RMGNYYSDDVAGAWAALVGVVQAAL

predict

Class:
Globin-like

Function:
Oxygen transport

3D Structure:



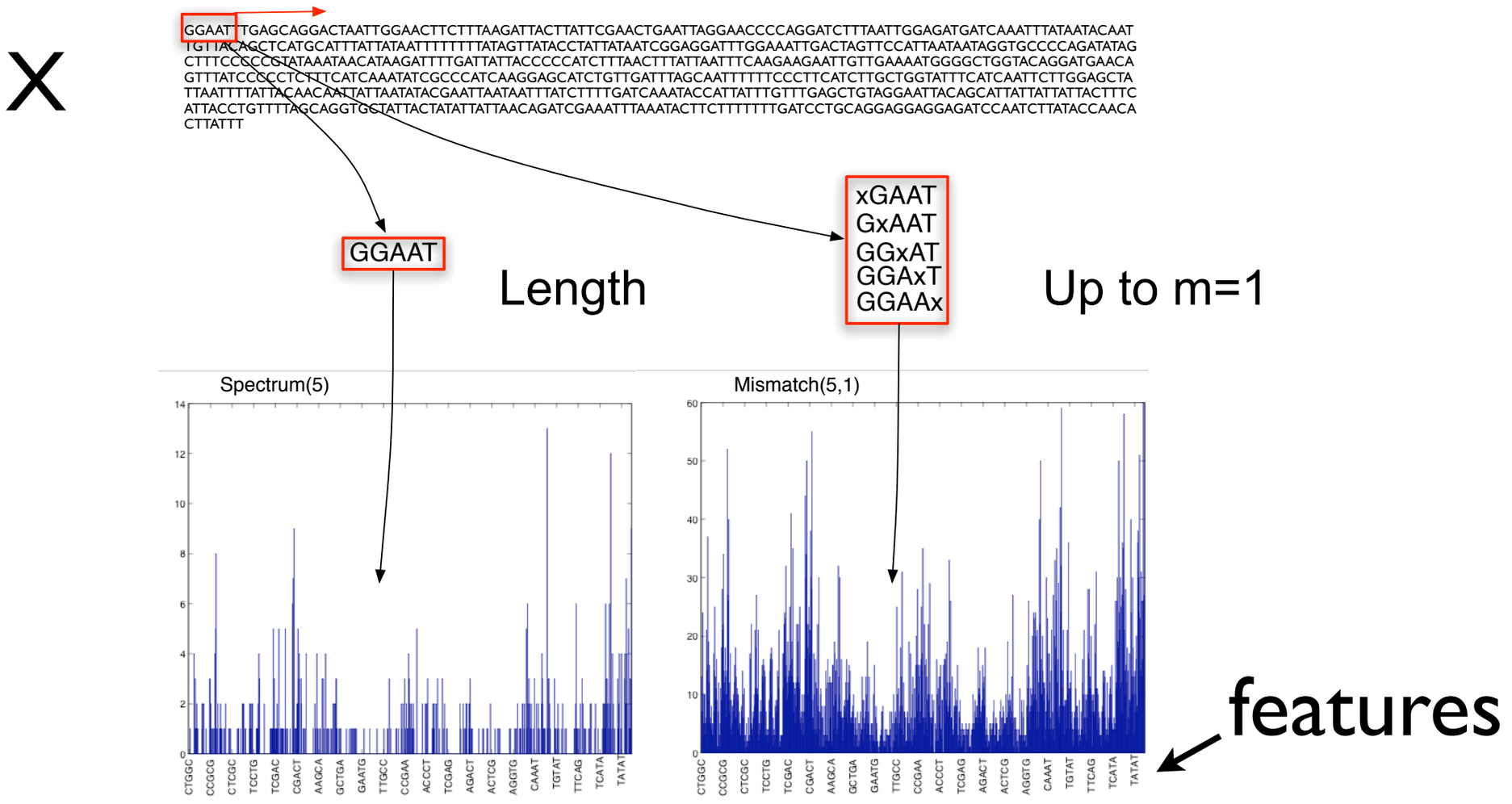
K(X,Y) = <F(X), F(Y)>

^{2} seq. feature vector

- String kernels
  - Superior for classification in diverse settings: text, bioinformatics, music etc
  - Similarity-based inference: kernel $K(X,Y)=\langle F(X),F(Y)\rangle$
  - Classification: $\mathbf{f}(X) = \sum_{X_i \in \mathbf{SV}} \alpha_i K(X,X_i)$

Inexact String Matching Kernels

- However, computation is challenging as sequence matching is based on co-occurrence of sub-patterns (eg, *k*-mers) *with mismatches*
- *Expensive* $O(k^m|\Sigma|^m n)$ evaluation using trie-based approach
  - not applicable for *large k and m*
  - limits feasible set of kernel parameters to *small k,m*



Similar X and Y ~ Similar F(X) and F(Y)

Preview of the Results

1. Theoretical results:

   - significantly improved computation for *inexact string matching* kernels with *large substrings (k)* and *many mismatches (m)*
   - $\left(\frac{|\Sigma|}{m+2}\right)^m$ *theoretical speed-up* (alphabet size $|\sum| \gg m$)

2. Empirical results:

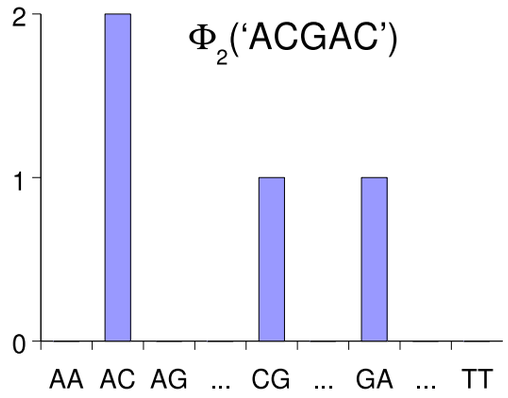
   - *orders-of-magnitude* speed-ups (~200x)
   - significant *classification performance gains* (~30%) with *large-(k,m)* kernels over state-of-the-art on *bioinformatics* and *music classification*

Background: String Kernels

Spectrum Kernels

- Measure similarity between sequences based on co-occurrence of fixed-length substrings (k-mers)
- Feature map for exact spectrum kernel (no mismatches)

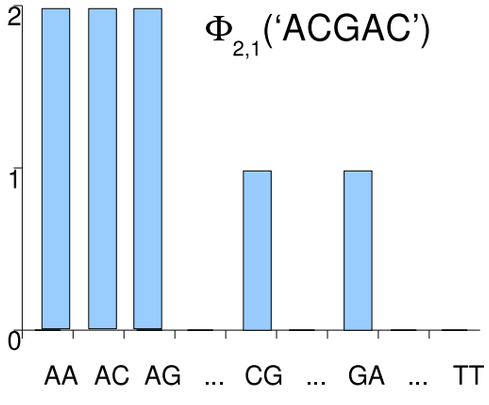
$$\Phi(X) = \left(\sum_{\substack{\alpha \in X \\ |\alpha| = k}} I(\alpha, \gamma) \right)_{\gamma \in \Sigma^k}$$



X="ACGAC"
k=2

- Feature map for mismatch kernel (*m* mismatches)

$$\Phi(X) = \left(\sum_{\substack{\alpha \in X \\ |\alpha| = k}} I_m(\alpha, \gamma) \right)_{\gamma \in \Sigma^k}$$



X="ACGAC"
k=2, *m*=1

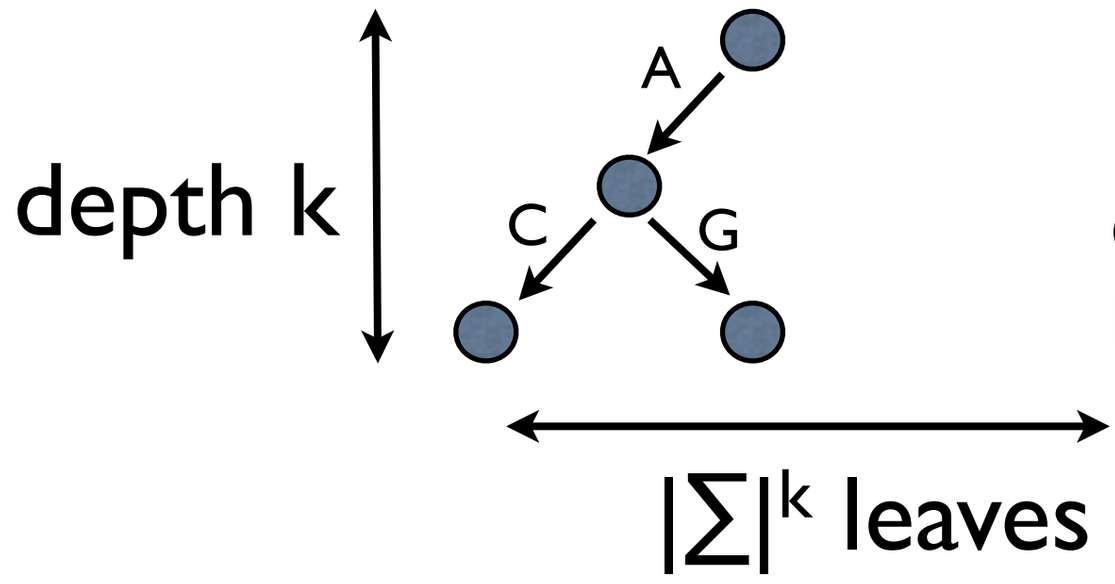
- more dense representation (up to $k^m|\sum|^m$ more dense)

What are the best algorithms for string kernels?

- Suffix trees (Vishwanathan & Smola, 2002)
  - linear-time all-substring kernel (exact!)
- (Sparse) dynamic programming (Rousu, 2005): eg, gapped kernel (gapped)
- ***Mismatch trie*** (Leslie, 2004)
  - applies to mismatch, spectrum, gapped kernels, etc (*inexact matching!!!*)
  - Complexity $O(k^{m+1}|\Sigma|^m(|X| + |Y|))$
    - limits applicability to small k,m,|∑|
- **Sufficient statistics** (SS) (Kuksa, 2008)
  - more efficient (no |∑| !) $O(k^m|\Sigma|^m n) \Rightarrow O(c_{k,m} n)$

Mismatch Trie Algorithm

- Depth-first search over complete tree with leaves/paths for all possible k-length substrings



$$O(k^{m+1}|\Sigma|^m(|X| + |Y|))$$

O(k^m |∑|^m) - number of substrings at Hamming distance of at most *m*

- Problematic for large-∑ inputs and relaxed matching (larger m)

Want to address this

Mismatch Kernel

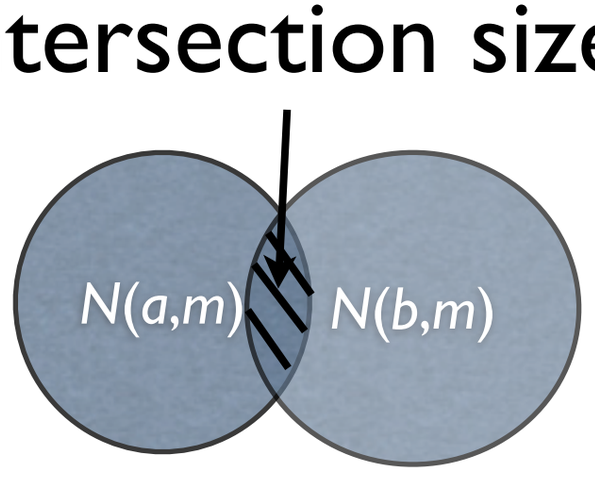
Main issue: *denser* substring spectrum, many more features due to approximate matching (m>0)

$$I_m(\alpha,\gamma) = \begin{cases} 1, d(\alpha,\gamma) \leq m \\ 0, \text{otherwise} \end{cases}$$

introduces for every *a* mismatch neighborhood *N*(*a*,*m*) of size $O(k^m|\Sigma|^m)$

$$\begin{aligned} K(X,Y|k,m) &= \sum_{\gamma\in\Sigma^k} \Phi_{k,m}(\gamma|X)\Phi_{k,m}(\gamma|Y). \\ &= \sum_{\alpha\in X}\sum_{\beta\in Y}\sum_{\gamma\in\Sigma^k} I_m(\alpha,\gamma)I_m(\beta,\gamma) \end{aligned}$$

intersection size for two mismatch neighborhoods



Observation I: number of substrings within distance *m* from both *a* and *b* is *independent* of *a* and *b*

Sufficient Statistics for String Kernels

Observation II: kernel is incremented only by $\min(2m,k)+1$ distinct values corresponding to Hamming distances 0-2*m*

$$K(X,Y|m,k) = \sum_{i_x=1}^{n_x-k+1} \sum_{i_y=1}^{n_y-k+1} \mathcal{I}(x_{i_x:i_x+k-1}, y_{i_y:i_y+k-1}) = \sum_{i=0}^{\min(2m,k)} M_i \mathcal{I}_i$$

Matching (sufficient) statistics:
M_i = number of pairs of substrings (a in X, b in Y) at Hamming distance d(a,b)=i

Matching statistics M_i can be computed in linear time! (NIPS, 2008)

How to systematically and efficiently compute int. sizes I_i?

Computing intersection sizes

- Intersection size I(a,b)
  - number of k-mers at Hamming distance ≤ m from both a and b
  - I(a,b) = |{c|h(a,c)≤m and h(b,c)≤m}|
- min(2m+1,k) distinct intersection sizes

  $I_0, I_1, \ldots, I_{min(2m+1,k)}$
  - I_i - intersection size for Hamming distance i
- *Intersection size computation problem*:

> Given |∑|,k, and m, *compute intersection sizes for Hamming distances 0...min(2m,k)*

Intersection size computation

1. Brute-force $O(k^m|\sum|^m)$: 2*m* trie traversals, only for small $|\sum|$,k,m
2. Analytical (closed-form) solution
    - No systematic way
    - Currently only for small *m*<4
        - Eg. as a function of *k* and *m (NIPS, 2008)*

$$\underset{(m=2)}{\mathcal{I}(a,b)} = \begin{cases} |N_{k,m}|, d(a,b) = 0 \\ 1 + k(|\Sigma| - 1) + (k-1)(|\Sigma| - 1)^2, d(a,b) = 1 \\ 1 + 2(k-1)(|\Sigma| - 1) + (|\Sigma| - 1)^2, d(a,b) = 2 \\ 6(|\Sigma| - 1), d(a,b) = 3 \\ \binom{4}{2}, d(a,b) = 4 \end{cases}$$

Intersection size computation: reduction-based method

- Observation: Intersection size can be found in a *weighted form* for every Hamming distance $0 \leq h \leq 2m$

$$\mathcal{I}(a,b) = \sum_{i=0}^{m} w_i (|\Sigma| - 1)^i$$

- Coefficients w_i, i=0...m, are *not* known but are *the same* (!!) for any $|\Sigma|$

$\Rightarrow$ w_i (and I_i) can be found by solving a *linear system* of $(m+1)$ equations!

$$Aw = \mathcal{I}$$

Intersection size computation: reduction-based

$$Aw = \mathcal{I}$$

|∑|=2 →

|∑|=m+2 →

$$A = \begin{pmatrix} 1^0 & 1^1 & 1^2 & \dots & 1^m \\ 2^0 & 2^1 & 2^2 & \dots & 2^m \\ & & & \dots & \\ (m+1)^0 & (m+1)^1 & (m+1)^2 & \dots & (m+1)^m \end{pmatrix}$$

$$\mathcal{I} = (I_0, I_1, \dots, I_m)^T$$

- $I_i, i = 0 \dots m$, intersection size for a particular setting of *k*,*m*,*h*,|∑|=2,3,...,*m*+2
- Note: this reduces computation for *large* |∑| to a set of (m+1) *less complex* intersection size computations for *smaller* alphabet sizes

Intersection size computation: reduction-based method

- $I=I_0...I_{m+1}$ can be feasibly computed using explicit trie traversal ($|\sum|\leq m+2$)!
- the size of the trie is only $(m+2)^k$ vs $|\sum|^k$
- eg. even for small $|\sum|=20$, k=13, m=6 the size of the trie is

$$20^{13}/(6+2)^{13}=149011 \text{ times smaller!}$$

intersection size computation: algorithm

- $O((2m+1)k^{m+1}(m+2)^m)$ vs $O((2m+1)k^{m+1}|\sum|^m)$ using trie traversal
- factor of $\left(\frac{|\Sigma|}{m+2}\right)^m$ speed-up ($|\sum| \gg m$)!

Algorithm 1: Intersection size computation

Input: kernel parameters k and m, alphabet size $|\Sigma|$

1: **for** $d = 0$ to $\min(k, 2m)$ **do**
2: {for each Hamming distance d}
3: **for** $i = 2$ to $m + 2$ **do**
4: {for each alphabet size $i = 2 \ldots m + 2$}
5: Create a pair of k-mers a and b with Hamming distance $h(a, b) = d$
6: Obtain intersection size I_i by computing (k, m)-mismatch kernel $K_{k,m}(a, b)$ using trie traversal over i^k tree.
7: **end for**
8: Set $(m + 1) \times (m + 1)$ matrix A with $A_{ij} = (i + 1)^{(j-1)}$, $i = 1, \ldots, m + 1$, $j = 1, \ldots, m + 1$
9: Solve $Aw = I$ to obtain weight vector w
10: Set $(d + 1)$th row of $W_{k,m}$ to w
11: **end for**
12: Compute vector of intersection sizes $\mathcal{I}_{k,m,|\Sigma|} = W_{k,m}((|\Sigma| - 1)^0 \ldots (\Sigma - 1)^m)^T$

Output: Weight matrix $W_{k,m}$ of size $(\min(k, 2m) + 1, m + 1)$, vector of intersection sizes (lookup table) $\mathcal{I}_{k,m,|\Sigma|}$

Weight matrices: Examples

$$W_{9,4} = \begin{pmatrix} 1 & 9 & 36 & 84 & 126 \\ 1 & 9 & 36 & 84 & 56 \\ 1 & 9 & 36 & 119 & 21 \\ 1 & 9 & 21 & 109 & 6 \\ 1 & 14 & 66 & 64 & 1 \\ 0 & -30 & 120 & 20 & 0 \\ 20 & 0 & 90 & 0 & 0 \\ -70 & 140 & 0 & 0 & 0 \\ 70 & 0 & 0 & 0 & 0 \end{pmatrix},$$

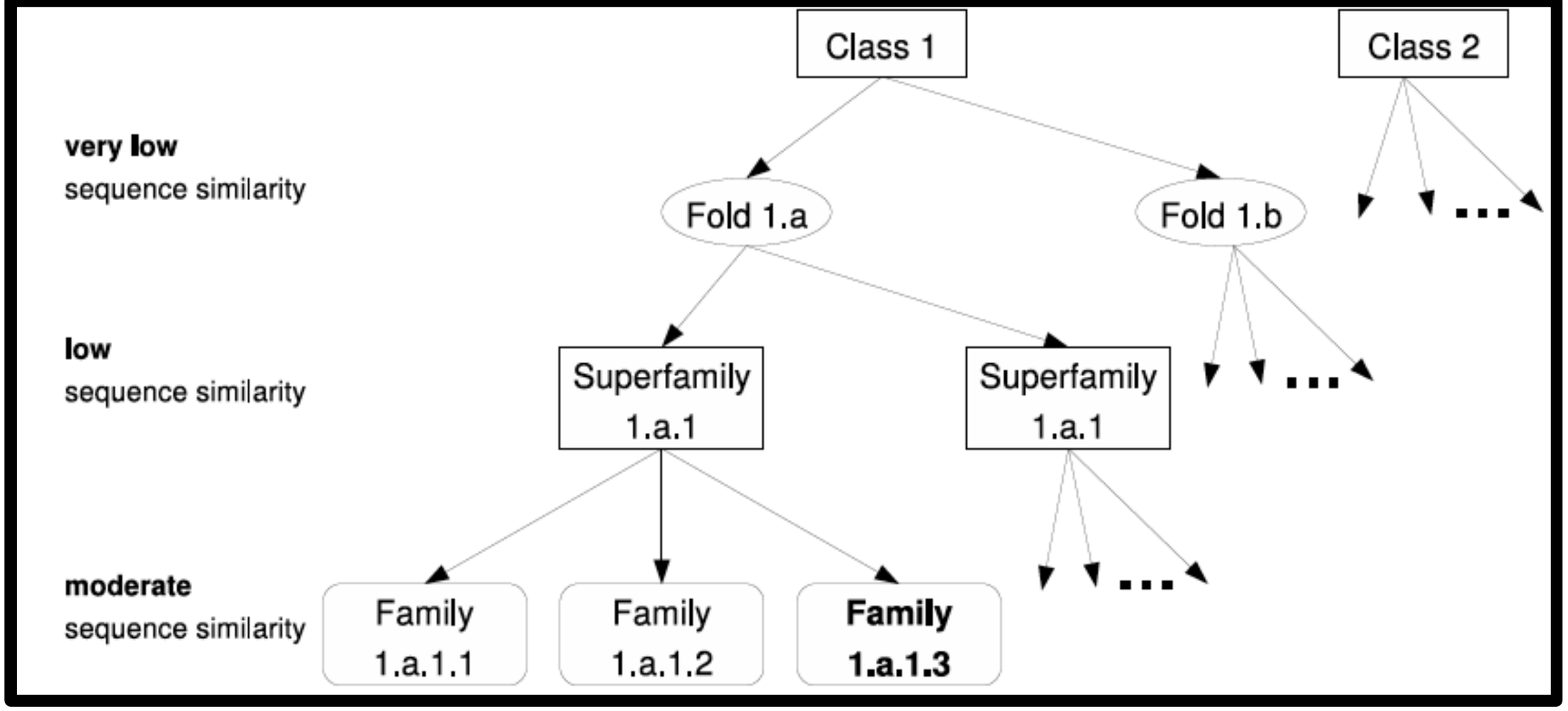
$$W_{10,5} = \begin{pmatrix} 1 & 10 & 45 & 120 & 210 & 252 \\ 1 & 10 & 45 & 120 & 210 & 126 \\ 1 & 10 & 45 & 120 & 280 & 56 \\ 1 & 10 & 45 & 85 & 280 & 21 \\ 1 & 10 & 60 & 160 & 205 & 6 \\ 1 & 5 & -40 & 310 & 105 & 1 \\ 2 & 80 & -90 & 360 & 30 & 0 \\ -28 & -70 & 210 & 210 & 0 & 0 \\ 182 & -420 & 560 & 0 & 0 & 0 \\ -378 & 630 & 0 & 0 & 0 & 0 \\ 252 & 0 & 0 & 0 & 0 & 0 \end{pmatrix},$$

Evaluation:

1. Protein structural classification
   1.1. Remote homology prediction
   1.2. Protein fold prediction
2. Music classification
3. Model selection (kernel parameter selection)

Protein Classification

Structural Classification of Proteins (SCOP)



Remote homology prediction

- 54 classification problems

Large-(*k*,*m*) performance

Kernel	Mean ROC	Mean ROC50
mismatch(5,1)	87.75	41.92
mismatch(5,2)	90.67	49.09
mismatch(6,2)	90.74	49.66
mismatch(6,3)	90.98	49.36
mismatch(7,3)	91.31	52.00
mismatch(7,4)	90.84	49.29
mismatch(9,3)	90.27	47.00
mismatch(9,4)	91.45	53.51
mismatch(10,4)	91.02	50.49
mismatch(10,5)	**91.60**	**53.78**
mismatch(13,6)	90.98	50.11

Comparison with state-of-the-art

method	ROC	ROC50
PSI-BLAST [6]	74.29	29.25
SVM-Pairwise (Smith-Waterman) [13]	89.30	43.40
Subsequence kernel [14]	87.23	40.37
SSSK [7]	91.48	51.18
SEQL [5]	**92.20**	52.37
Mismatch(k=10,m=5)	91.60	**53.78**

- Mean ROC50 41.92 → 53.78 (28% rel. improvement)
- Large-(*k*,*m*) kernel compared to typically used small *k* and *m*

Comparison of (k,m)-kernels

- Statistical significance of the observed differences on remote homology prediction task

	(5,1)	(5,2)	(6,2)	(6,3)	(7,4)	(9,4)
(5,2)	3.4e-04					
(6,2)	3.7e-06	3.8e-01				
(6,3)	1.4e-03	7.2e-01	7.7e-01			
(7,4)	2.2e-03	7.0e-01	9.8e-01	8.7e-01		
(9,4)	1.6e-06	3.2e-03	7.3e-04	1.4e-02	2.9e-02	
(10,5)	3.1e-06	1.8e-03	1.5e-03	2.5e-03	9.4e-03	5.6e-01

- Large (k,m) kernels perform better

Protein fold prediction

- multi-class, 27-folds

method	Error	F1
(*k*=5,*m*=1)	51.17	61.69
(*k*=7,*m*=3)	43.86	**65.55**
(*k*=10,*m*=6)	**42.82**	62.36

Music Classification

- multi-class (10 genres)

method	Error	F1
(k=5,m=1)	34.8	65.36
(k=7,m=4)	**31.1**	**68.96**
(k=10,m=4	31.7	68.29

- Kernels with moderately long k and larger values of m perform better than kernels with small k,m

Kernel parameter selection

- Parameter selection by validation

Kernel	ROC (Test)	ROC50 (Test)	ROC (Validation)	ROC50 (Validation)
mismatch(5,1)	85.75	39.47	88.28	73.21
mismatch(5,2)	89.62	48.10	90.36	77.17
mismatch(6,2)	89.26	48.72	90.61	77.89
mismatch(6,3)	89.95	48.38	90.32	77.05
mismatch(7,3)	90.08	51.64	91.17	79.21
mismatch(7,4)	89.79	47.74	90.04	76.40
mismatch(9,3)	87.14	44.27	89.55	76.47
mismatch(9,4)	89.97	**52.59**	91.90	**81.57**
mismatch(10,4)	88.77	48.47	90.90	79.15
mismatch(10,5)	90.40	**52.96**	**92.34**	**81.30**
mismatch(11,5)	89.68	50.14	91.52	80.25
mismatch(11,6)	**90.58**	52.72	**92.38**	81.23
mismatch(13,5)	86.57	42.56	87.79	70.12

selected kernels

- Kernels selected by validation (*k*=10,*m*=5 and *k*=9,*m*=4) also perform best on *test* set

Running time

Speed-up (running time ratio $T_{trie}/T_{lin.sys}$)

(k,m)	$\|\Sigma\| = 20$	$\|\Sigma\| = 100$
(5,2)	6	254
(5,3)	27	5931
(6,2)	9	393
(6,3)	45	10118
(7,2)	13	571
(7,3)	57	12848
(7,4)	120	n/a / $8 \times 10^{4\dagger}$
(9,4)	132	n/a / $8 \times 10^{4\dagger}$
(9,5)	197	n/a / $6 \times 10^{5\dagger}$
(10,5)	208	n/a / $6 \times 10^{5\dagger}$

† expected running time ratio only due to excessive running time for trie-based method

speed-up:
~10x for *m*=2-3
~100x *m*>3

- Observed speed-ups are on the order of $\left(\frac{|\Sigma|}{m+2}\right)^m$ as expected from theoretical analysis
- E.g. for protein alphabet, $|\sum|=20$, $m=5$
  speed-up is **208**x vs $20^5/(5+2)^5=190$x expected

Conclusions

- Efficient algorithm for computation of large-(k,m) inexact matching string kernels
- Large-(k,m) kernels can improve performance
- Can now explore larger set of kernels with a wide range of parameters (better model selection, performance)

References

1. Pavel P. Kuksa, Pai-Hsi Huang, Vladimir Pavlovic. *Scalable Algorithms for String Kernels with Inexact Matching.* NIPS, 2008

2. Christina Leslie and Rui Kuang. *Fast string kernels using inexact matching for protein sequences*. JMLR, 2004

3. Pavel P. Kuksa and Vladimir Pavlovic. *Spatial Representation for Efficient Sequence Classification*. ICPR, 2010

4. Christina S. Leslie, Eleazar Eskin, Jason Weston, and William Stafford Noble. *Mismatch string kernels for svm protein classification*. NIPS , 2002