Weighted Finite-State Transducers in Computational Biology

Corinna Cortes
Google Research
corinna@google.com

Mehryar Mohri
Courant Institute
mohri@cs.nyu.edu

With contributions from Fernando Pereira, Patrick Haffner, and Michael Riley. Thanks also to Eleazar Eskin for helpful suggestions.
This Tutorial

- Weighted finite-state transducers and software library (FSM Library)
- Pairwise alignments for bioinformatics
- Kernels for computational biology
General Definitions

• **Weighted finite-state transducers:**

• **Kernels**: similarity measures between vectors, sequences or other structures.
  • Efficient computation of inner products in high-dimensional feature spaces.
  • Extensive use in modern machine learning.
  • Must satisfy a mathematical requirement (Mercer’s condition).
**Example**

- **Problem**: find the $n$ best pairwise alignments between two very large sets of sequences (e.g., $|\text{set}| > 10,000$).

- **Example**:

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a b a a b b a b a b b</td>
<td>b b b a a a b a b b</td>
</tr>
<tr>
<td></td>
<td>a b b a a a a</td>
<td>a a b b a a a b</td>
</tr>
<tr>
<td></td>
<td>a b b b a a a b</td>
<td>a a b b b b a a a</td>
</tr>
<tr>
<td></td>
<td>b b b a a b a b</td>
<td>b a b b b b a b</td>
</tr>
<tr>
<td></td>
<td>b b b a a b a b</td>
<td>b a a b b b a a</td>
</tr>
<tr>
<td>η</td>
<td>a b b a a a b</td>
<td>b a b a b b a a</td>
</tr>
</tbody>
</table>

ε a ε b b a a a
a a b b b a a a
Algorithms and Software Library (FSM Library)
Software Libraries

- **FSM Library**: Finite-State Machine Library. General software utilities for building, combining, optimizing, and searching weighted automata and transducers.


- **GRM Library**: Grammar Library. General software collection for constructing and modifying weighted automata and transducers representing grammars and statistical language models.

The FSM utilities construct, combine, minimize, and search *weighted finite-states machines* (FSMs).

**User Program Level:** Programs that read from and write to files or pipelines, *fsm(1):*

```bash
define intersect in1.fsm in2.fsm > out.fsm
```

**C(++) Library Level:** Library archive of C(++) functions that implements the user program level, *fsm(3):*

```c
Fsm in1 = FSMLoad("in1.fsm");
Fsm in2 = FSMLoad("in2.fsm");
Fsm out = FSMIntersect(fsm1, fsm2);
FSMDump("out.fsm", out);
```
• **Definition Level**: Specification of *labels*, of *costs*, and of types of FSM representations.
FSM File Types

• **Textual format**
  - automata/acceptor files,
  - transducer files,
  - symbols files.

• **Binary format**: *compiled* representation used by all FSM utilities.
Compiling, Printing, and Drawing FSMs

- **Compiling**
  - fsmcompile \(-s \text{tropical}\) \(-i\text{A.syms}\) \(<\text{A.txt} >\text{A.fsm}\)
  - fsmcompile \(-s \text{log}\) \(-i\text{A.syms}\) \(-o\text{A.syms}\) \(-t <\text{T.txt} >\text{T.fsm}\)

- **Printing**
  - fsmprint \(-i\text{A.syms}\) \(<\text{A.fsm} >\text{A.txt}\)
  - fsmprint \(-i\text{A.syms}\) \(-o\text{A.syms}\) \(<\text{T.fsm} | \text{dot -Tps} >\text{T.ps}\)

- **Drawing**
  - fsmdraw \(-i\text{A.syms}\) \(<\text{A.fsm} | \text{dot -Tps} >\text{A.ps}\)
  - fsmdraw \(-i\text{A.syms}\) \(-o\text{A.syms}\) \(<\text{T.fsm} | \text{dot -Tps} >\text{T.ps}\)
Weight Sets: Semirings

• A *semiring* $(\mathbb{K}, \oplus, \otimes, 0, 1)$ is a ring that may lack negation.

• **sum**: to compute the weight of a sequence (sum of the weights of the paths labeled with that sequence).

• **product**: to compute the weight of a path (product of the weights of constituent transitions).
Semirings - Examples

<table>
<thead>
<tr>
<th>Semiring</th>
<th>Set</th>
<th>⊕</th>
<th>⊗</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>{0, 1}</td>
<td>∨</td>
<td>∧</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Probability</td>
<td>(\mathbb{R}_+)</td>
<td>+</td>
<td>×</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Log</td>
<td>(\mathbb{R} \cup {-\infty, +\infty})</td>
<td>(\oplus_{\text{log}})</td>
<td>+</td>
<td>+(\infty)</td>
<td>0</td>
</tr>
<tr>
<td>Tropical</td>
<td>(\mathbb{R} \cup {-\infty, +\infty})</td>
<td>min</td>
<td>+</td>
<td>+(\infty)</td>
<td>0</td>
</tr>
</tbody>
</table>

with \(\oplus_{\text{log}}\) defined by: \(x \oplus_{\text{log}} y = -\log(e^{-x} + e^{-y})\).
Automata/Acceptors

• **Graphical Representation** \((A.ps)\)

```
0 red/.5
1 green/.3
1 blue/0
2 yellow/.6
0 0 red .5
0 1 green .3
1 2 blue
1 2 yellow .6
2 .8
```

• **Acceptor file** \((A.txt)\)

```
0 0 red .5
0 1 green .3
1 2 blue
1 2 yellow .6
2 .8
```

• **Symbols file** \((A.sym)\)

```
red 1
green 2
blue 3
yellow 4
```
Transducers

- **Graphical Representation** \((T.ps)\)

```
0 -> 1: green:blue/0.3
1 -> 2: blue:green/0
2 -> 0: yellow:red/0.6
```

- **Transducer file** \((T.txt)\)

```
0 0 red yellow .5
0 1 green blue .3
1 2 blue green
1 2 yellow red .6
2 .8
```

- **Symbols file** \((T.sym)\)

```
red 1
green 2
blue 3
yellow 4
```
Paths - Definitions and Notation

- **Path** $\pi$
  - origin or previous state: $p[\pi]$
  - destination or next state: $n[\pi]$
  - input label: $i[\pi]$
  - output label: $o[\pi]$

- **Sets of paths**
  - $P(R_1, R_2)$: paths from $R_1 \subseteq Q$ to $R_2 \subseteq Q$.
  - $P(R_1, x, R_2)$: paths in $P(R_1, R_2)$ with input label $x$.
  - $P(R_1, x, y, R_2)$: paths in $P(R_1, x, R_2)$ with output label $y$. 

![Diagram showing the notation of a path with arrows pointing from $p[\pi]$ to $n[\pi]$.](image-url)
General Definitions

• **Alphabets**: input $\Sigma$, output $\Delta$.

• **States**: $Q$, initial states $I$, final states $F$.

• **Transitions**: $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\}) \times K \times Q$

• **Weight functions**:
  
  • initial: $\lambda : I \rightarrow K$
  
  • final: $\rho : F \rightarrow K$
Automata and Transducers - Definitions

• **Automaton** \( A = (\Sigma, Q, I, F, E, \lambda, \rho) \)

\[ \forall x \in \Sigma^*, \]

\[ [A](x) = \bigoplus_{\pi \in P(I,x,F)} \lambda(p[\pi]) \otimes w[\pi] \otimes \rho(n[\pi]) \]

• **Transducer** \( T = (\Sigma, \Delta, Q, I, F, E, \lambda, \rho) \)

\[ \forall x \in \Sigma^*, y \in \Delta^*, \]

\[ [T](x, y) = \bigoplus_{\pi \in P(I,x,y,F)} \lambda(p[\pi]) \otimes w[\pi] \otimes \rho(n[\pi]) \]
Weighted Automata

\[[A](x) = \text{Sum of the weights of all successful paths labeled with } x\]

\[[A](a bb) = .1 \times .2 \times .3 \times .1 + .5 \times .3 \times .6 \times .1\]
Weighted Transducers

\[ [[T]](x, y) = \text{Sum of the weights of all successful paths with input } x \text{ and output } y \]

\[ [[T]](abb, baa) = .1 \times .2 \times .3 \times .1 + .5 \times .3 \times .6 \times .1 \]
Rational Operations

- **Sum**

\[
[T_1 \oplus T_2](x, y) = [T_1](x, y) \oplus [T_2](x, y)
\]

- **Product**

\[
[T_1 \otimes T_2](x, y) = \bigoplus_{x=x_1 x_2, y=y_1 y_2} [T_1](x_1, y_1) \otimes [T_2](x_2, y_2)
\]

- **Closure**

\[
[T^*](x, y) = \bigoplus_{n=0}^{\infty} [T]^n(x, y)
\]
• **Conditions** (on the closure operation): condition on $T$: e.g., weight of $\varepsilon$-cycles = $\overline{0}$ (*regulated transducers*), or semiring condition: e.g., $\overline{1} \oplus x = \overline{1}$ as with the tropical semiring (more generally *locally closed semirings*).

• **Complexity and implementation:**

  • linear-time complexity:

    $O((|E_1| + |Q_1|) + (|E_2| + |Q_2|))$ or $O(|Q| + |E|)$

  • lazy implementation.
Sum - Illustration

- **Program**: `fsmunion A.fsm B.fsm > C.fsm`

- **Graphical representation:**

```plaintext
A.fsa
0
red/0.5
1 green/0.3
2 /0.8 blue/0
yellow/0.6

B.fsa
0
1 /0
green/0.4
2 /0.3
blue/1.2

C.fsa
0
red/0.5
1 green/0.3
2 /0.8 blue/0
yellow/0.6
3 4 /0
green/0.4
5 /0.3
blue/1.2
6 eps/0
eps/0
```
Product - Illustration

- **Program**: `fsmconcat A.fsm B.fsm > C.fsm`

- **Graphical representation**:

  ![Diagram](image-url)
Closure - Illustration

- **Program:** fsmclosure B.fsm > C.fsm
- **Graphical representation:**

```
0 -> 1 /0 green/0.4
    -> 2 /0.3 blue/1.2

3 /0 -> 0 eps/0
    -> 1 /0 green/0.4
    -> 2 /0.3 eps/0
    -> 3 /0 eps/0.3
```

```
0 -> 1 /0 green/0.4
    -> 2 /0.3 blue/1.2

3 /0 -> 0 eps/0
    -> 1 /0 green/0.4
    -> 2 /0.3 eps/0.3
```
Elementary Unary Operations

- **Reversal**

  \[ \tilde{T}(x, y) = [T](\tilde{x}, \tilde{y}) \]

- **Inversion**

  \[ [T^{-1}](x, y) = [T](y, x) \]

- **Projection**

  \[ A(x) = \bigoplus_{y} [T](x, y) \]

- **Linear-time complexity, lazy implementation (not for reversal).**
Reversal - Illustration

- **Program**: fsmreverse A.fsm > C.fsm
- **Graphical representation**:
Inversion - Illustration

- **Program**: `fsminvert A.fsm > C.fsm`

- **Graphical representation**: 

```
A.fst
0   red:bird/0.5
   ↓
  1  green:pig/0.3
      ↓
  2   blue:cat/0
       ↓
     0   yellow:dog/0.6

C.fst
0   bird:red/0.5
   ↓
  1  pig:green/0.3
      ↓
  2   cat:blue/0
       ↓
     0   dog:yellow/0.6
```
**Projection - Illustration**

- **Program**: `fsmproject -l T.fsm > A.fsm`
- **Graphical representation:**

```plaintext
A.fsm
0
red/0.5
1
green/0.3
2 /0.8
blue/0
yellow/0.6

C.fst
0
bird/red/0.5
1 pig/green/0.3
2 /0.8
cat/blue/0
dog/yellow/0.6
```

```plaintext
T.fst
0
red/bird/0.5
1 green/pig/0.3
2 /0.8
blue/cat/0
yellow/dog/0.6
```

Corinna Cortes and Mehryar Mohri - WFSTs in Computational Biology - Part I
Some Fundamental Binary Operations

(Pereira and Riley, 1997; Mohri et al. 1996)

- **Composition** \(((K, \oplus, \otimes, 0, 1)\) commutative)\n
\[
[T_1 \circ T_2](x, y) = \bigoplus_z [T_1](x, z) \otimes [T_2](z, y)
\]

- **Intersection** \(((K, \oplus, \otimes, 0, 1)\) commutative)\n
\[
[A_1 \cap A_2](x) = [A_1](x) \otimes [A_2](x)
\]

- **Difference** \((A_2\) unweighted and deterministic)\n
\[
[A_1 - A_2](x) = [A_1 \cap \overline{A_2}](x)
\]
• **Complexity and implementation:**

  • quadratic complexity:

    \[ O((|E_1| + |Q_1|) (|E_2| + |Q_2|)) \]

  • path multiplicity in presence of \( \epsilon \)-transitions: \( \epsilon \)-filter;

  • lazy implementation.
Composition - Illustration

- **Program**: `fsmcompose A.fsm B.fsm > C.fsm`

- **Graphical representation**: 

```plaintext
Program: fsmcompose A.fsm B.fsm > C.fsm

Graphical representation:
```

Diagram of the composition process with transitions labeled with probabilities and states marked with labels.
Multiplicity and $\epsilon$-Transitions - Problem

Redundant $\epsilon$-paths.
Solution - Filter $F$ for Composition

Replace $T_1 \circ T_2$ with $\tilde{T}_1 \circ F \circ \tilde{T}_2$. 
Intersection - Illustration

- **Program**: `fsmintersect A.fsm B.fsm > C.fsm`
- **Graphical representation**:

```
A.fsa
0 1
red/0.5 0.3
1 2
blue/0 0.8
2
red/0.7
3 /0.8
yellow/1.9
4 /1.3
blue/0.6
```
Difference - Illustration

- **Program**: fsmdifference A.fsm B.fsm > C.fsm

- **Graphical representation**:

  A.fsa
  0
  red/0.5
  1 green/0.3
  2 blue/0
  3 yellow/0.6

  B.fsa
  0 red/0.2
  1 blue/0.6
  2 green/0.4
  3 yellow/1.3

  C.fsa
  0 red/0.7
  1 green/0.7
  2 blue/0.6
  3 yellow/1.9

  Graphical representation of the difference between A.fsm and B.fsm, with colors and probabilities indicated.
Optimization Algorithms

- **Connection**: removes non-accessible/non-coaccessible states
- **ε-Removal**: removes ε-transitions
- **Determinization**: creates equivalent deterministic machine
- **Pushing**: creates equivalent pushed/stochastic machine
- **Minimization**: creates equivalent minimal deterministic machine
• **Conditions**: there are specific semiring conditions for the use of these algorithms, e.g., not all weighted automata or transducers can be determinized using the determinization algorithm.
Connection - Illustration

- **Program**: `fsmconnect A.fsm > C.fsm`

- **Graphical representation**:

```
[Diagram of a connection with states and transitions]
```
Connection - Algorithm

• **Definition:**
  - Input: weighted transducer $T_1$
  - Output: equivalent weighted transducer $T_2$ with all states connected

• **Description:**
  1. Depth-first search of $T_1$ from $I_1$.
  2. Mark accessible and coaccessible states.
  3. Keep marked states and corresponding transitions.

• **Complexity:** linear $O(|Q_1| + |E_1|)$. 
**ε-Removal - Illustration**

- **Program**: `fsmrmepsilon T.fsm > TP.fsm`
- **Graphical representation:**

---

**Diagram**

- Nodes: 0, 1, 2, 3, 4/0
- Edges:
  - 0: a:b/0.1, ε/ε/0.2, b:a/0.4
  - 1: a:b/0.1, ε/ε/0.2, b:a/0.9
  - 2: b:a/0.5, ε/ε/0.6, a:a/0.8
  - 3: b:b/0.5, b:a/0.9, a:a/1.6
  - 4/0: a:a/1.6, b:a/1

---

**Weights**

- ε/ε/0.2
- a:b/0.1
- b:a/0.4
- b:a/0.9
- ε/ε/0.6
- b:a/0.5
- a:a/0.8
- b:b/0.5
- b:a/1.6
- a:a/1.6
- b:a/1

---

**Weights**

- 0.2
- 0.8
- 0.6
- 1.6
- 3
- 0
- 1
- 2
- 3
- 1
- 2
**ε-Removal - Algorithm**

(Mohri, 2001)

- **Definition:**
  - Input: weighted transducer $T_1$
  - Output: equivalent WFST $T_2$ with no ε-transition

- **Description:**
  2. Removal of εs.

- **Complexity:**
  - Acyclic $T_\epsilon : O(|Q|^2 + |Q||E|(T_\oplus + T_\otimes ))$.
  - General case (tropical semiring):
    $$O(|Q||E| + |Q|^2 \log |Q|)$$
Computation of $\varepsilon$-closures

- **Definition**: for $p$ in $Q$,

$$C[p] = \{(q, w) : q \in \varepsilon[p], d[p, q] = w \neq \bar{0}\},$$

where $d[p, q] = \bigoplus_{\pi \in P(p, \varepsilon, q)} w[\pi].$

- **Problem formulation**: all-pairs shortest-distance problem in $T_\varepsilon$ ($T$ reduced to its $\varepsilon$-transitions).
  - closed semirings: generalization of Floyd-Warshall algorithm.
  - $k$-closed semirings: generic sparse shortest-distance algorithm.
Determinization - Algorithm

(Mohri, 1997)

- **Definition:**
  - Input: weighted automaton or transducer $T_1$
  - Output: equivalent *subsequential* or *deterministic* machine $T_2$: has a unique initial state and no two transitions leaving the same state share the same input label.

- **Description:**
  1. Generalization of subset construction: *weighted subsets* $\{ (q_1, w_1), \ldots, (q_n, w_n) \}$, where $w_i$'s are remainder weights.
  2. Computation of the weight of resulting transitions.
Determinization - Conditions

- **Semiring**: weakly left divisible semirings.

- **Definition**: $T$ is **determinizable** when the determinization algorithm applies to $T$.

- All unweighted automata are determinizable.

- All acyclic machines are determinizable.

- Not all weighted automata or transducers are determinizable.

- Characterization based on the **twins property**.

- **Complexity**: exponential, but lazy implementation.
Determinization of Weighted Automata - Illustration

- **Program**: fsmdeterminize A.fsm > D.fsm
- **Graphical representation**: 

![Graphical Representation of Determinization](image)

```plaintext
Determinization of Weighted Automata - Illustration

- **Program**: fsmdeterminize A.fsm > D.fsm
- **Graphical representation**:

![Graphical Representation of Determinization](image)
```
Determinization of Weighted Transducers - Illustration

- **Program:** `fsmdeterminize T.fsm > D.fsm`

- **Graphical representation:**

```
Determine ination of Weighted Transducers - Illustration
• Program: fsmdeterminize T.fsm > D.fsm
• Graphical representation:
```

![Graphical representation of determinization of weighted transducers](image_url)
Pushing - Algorithm

(Mohri, 1997; 2004)

- **Definition:**
  - **Input:** weighted automaton or transducer $T_1$
  - **Output:** equivalent automaton or transducer $T_2$
    such that the longest common prefix of all outgoing paths be $\epsilon$ or such that the sum of the weights of all outgoing transitions be $\overline{1}$ modulo the string or weight at the initial state.
• **Description:**

1. **Single-source shortest distance computation:**
   for each state $q$,
   \[
   d[q] = \bigoplus_{\pi \in P(q, F)} w[\pi].
   \]

2. **Reweighting:** for each transition $e$ such that $d[p[e]] \neq 0$,
   \[
   w[e] \leftarrow (d[p[e]])^{-1}(w[e] \otimes d[n[e]])
   \]
• **Conditions** (automata case): weakly divisible semiring, zero-sum free semiring or automaton.

• **Complexity:**
  
  • automata case
    
    • acyclic case: linear $O(|Q| + |E|(T_{\oplus} + T_{\otimes}))$.
    
    • general case (tropical semiring):
      
      $$O(|Q| \log |Q| + |E|).$$
    
    • transducer case:
      
      $$O((|P_{\text{max}}| + 1) |E|).$$
Weight Pushing - Illustration

- **Program**: fsmpush -ic A.fsm >P.fsm
- **Graphical representation**:
- **Tropical semiring**:

![Graphical representation of tropical semiring](image)

```plaintext
Program: fsmpush -ic A.fsm >P.fsm
```
• **Log semiring:**

![Diagram of Log semiring]
Label Pushing - Illustration

- **Program**: `fsmpush -il T.fsm >P.fsm`

- **Graphical representation:**

```
0 --a:a-- 1 --c:ε-- 2 --a:ε-- 3 --b:ε-- 4 --c:d-- 5 --a:ε-- 6
    |                |               |               |
0 --a:a-- 1 --c:d-- 2 --a:d-- 3 --b:ε-- 4 --c:ε-- 5 --a:ε-- 6
```
Minimization - Algorithm

(Mohri, 1997)

• **Definition:**
  - Input: deterministic weighted automaton or transducer $T_1$.
  - Output: equivalent deterministic automaton or transducer $T_2$ with the minimal number of states and transitions.

• **Description:**
  - Canonical representation: use pushing or other algorithm to standardize input automata.
  - Automata minimization: encode pairs (label, weight) as labels and use classical unweighted minimization algorithm.
• **Complexity:**

• **Automata case**
  
  • acyclic case: linear, $O(|Q| + |E|(T_\oplus + T_\otimes))$.

  • general case (tropical semiring): $O(|E| \log |Q|)$.

• **Transducer case**

  • acyclic case: $O(S + |Q| + |E| (|P_{max}| + 1))$.

  • general case (tropical semiring):
    
    $O(S + |Q| + |E| (\log |Q| + |P_{max}|))$. 
Minimization - Illustration

- **Program**: `fsmminimize D.fsm > M.fsm`

- **Graphical representation:**

![Graphical Representation](image)
Equivalence - Algorithm

• **Definition:**
  • Input: deterministic weighted automata $A$ and $B$.
  • Output: TRUE iff $A$ and $B$ equivalent.

• **Description** (Mohri, 1997):
  • Canonical representation: use pushing or other algorithm to standardize input automata.
  • Automata minimization: encode pairs (label, weight) as labels and use classical algorithm for testing the equivalence of unweighted automata.

• **Complexity:** (second stage is quasi-linear)

\[
O(|E_1| + |E_2| + |Q_1| \log |Q_1| + |Q_2| \log |Q_2|).
\]
Equivalence - Illustration

- **Program**: `fsmequiv [-v] D.fsm M.fsm`
- **Graphical representation**:

```
0 red/0.3
  1 yellow/0.9
  2 blue/0.7
  3 yellow/0.9
```

```
0 red/0
  1 blue/0
  2 yellow/0.9
```

`==?`
Single-Source Shortest-Distance Algorithms - Illustration

- **Program**: `fsmbestpath [-n N] A.fsm > C.fsm`
- **Graphical representation**:
Pruning - Illustration

- **Program**: fsmprune -c 1.0 A.fsm > C.fsm
- **Graphical representation**:

![Graphical representation of pruned FSMs](image-url)
Summary

- **FSM Library**:
  - weighted finite-state transducers (semirings);
  - elementary unary operations (e.g., reversal);
  - rational operations (sum, product, closure);
  - fundamental binary operations (e.g., composition);
  - optimization algorithms (e.g., $\varepsilon$-removal, determinization, minimization);
  - search algorithms (e.g., shortest-distance algorithms, $n$-best paths algorithms, pruning).
Weighted Finite-State Transducers in Computational Biology

Corinna Cortes  
Google Research  
corinna@google.com

Mehryar Mohri  
Courant Institute  
mohri@cs.nyu.edu
This Tutorial

- Weighted finite-state transducers and software library (FSM Library)
- Pairwise alignments for bioinformatics
- Kernels for computational biology
Pairwise Alignments for Bioinformatics
Biological Motivation

• **General idea**: use sequence similarity and known properties of some proteins to predict functional and structural information for other proteins.

• **Applications**:
  • Reconstruction of long DNA sequences,
  • Searching and mining DNA databases,
  • Finding frequent nucleotide patterns,
  • Determining informative sequences.
Why Weighted Finite-State Transducers?

(Mohri, Pereira, and Riley, 2000)

• **Generality**

  • Unified representation framework
  
  • Single general algorithm
  
  • No need to design and implement a novel algorithm for each new sequence alignment problem
  
  • Just define a new alignment transducer
  
  • Alignments between sets of strings, finite automata, or weighted automata (Mohri, 2003).
Why Weighted Finite-State Transducers?

- **Efficiency**
  - Compact representation of alignment transducer
  - Benefit of general transducer optimization algorithms
  - General quadratic-time alignment algorithm
  - Exploits graph representation and sparseness
  - Most ‘pruning’ ideas used in dynamic programming can be used with WFSTs as well.
Why Weighted Finite-State Transducers?

- **Flexibility**
  - Can use any weighted finite-state transducer
  - Can create more complex alignment transducers using general transducer algorithms, e.g., rational operations
  - Easy to modify, e.g., tailor alignment transducer to a specific problem
  - Graphical representation helps design and understanding
## Terminology

<table>
<thead>
<tr>
<th>Biology</th>
<th>Computer Science</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequence</td>
<td>string</td>
</tr>
<tr>
<td>subsequence</td>
<td>factor</td>
</tr>
<tr>
<td>-</td>
<td>substring</td>
</tr>
</tbody>
</table>

- **Computer science:**
  - Factor: contiguous symbols.
  - Substring or subsequence: non-necessarily contiguous symbols.
Local Edits

- **Insertion**: $\epsilon \rightarrow a$
- **Deletion**: $a \rightarrow \epsilon$
- **Substitution**: $a \rightarrow b$ (a $\neq$ b)
- **Example**: 2 insertions, 3 deletions, 1 substitution

```
c t t g $\epsilon$ $\epsilon$ a c
$$\epsilon$$ t a $\epsilon$ g t $\epsilon$ c
```

- This is called an *alignment*. 
Edit-Distance - Global Alignment

• Let $c(a, b)$ be the cost associated to the alignment of $a$ with $b$, where $a$ and $b$ are in $\Sigma \cup \{\epsilon\}$.

• **Problem**: Find the best alignment (minimal cost) of two sequences $x$ and $y$.

• **Solution**: general algorithm;
  
  • Construct edit transducer $T_e$ using $c(a, b)$ costs.
  
  • Represent $x$ and $y$ by automata $X$ and $Y$.

  • Compute best path of $X \circ T_e \circ Y$. 
Global Alignment - Example

• **Example:** \(c(A, G) = 1, c(A, T) = c(G, C) = .5\), no cost for matching symbols.

• **Representation:**

\[
\begin{align*}
&0 \quad A \xrightarrow{0} \quad G \xrightarrow{0} \quad C \xrightarrow{0} \quad T \xrightarrow{0} \quad 0 \\
&0 \quad A \xrightarrow{0} \quad C \xrightarrow{0} \quad C \xrightarrow{0} \quad T \xrightarrow{0} \quad G \xrightarrow{0} \quad 0
\end{align*}
\]

```
echo "A G C T" | farcompilestrings >X.fsm
```
Global Alignment - Example

- **Program:**
  
  ```
  fsmcompose X.fsm Te.fsm Y.fsm | fsmbestpath -n 1 >A.fsm
  ```

- **Graphical representation:**

![Graphical representation of global alignment](image_url)
Local Alignment

• **Problem**: Find the best local alignment (alignment between factors) of two sequences $x$ and $y$.

• **Motivation**: Ignore non-coding regions (introns).

• **Solution**: General algorithm;
  - Construct edit transducer $T_e$ using $c(a, b)$ costs.
  - Construct factor transducer $T_f$.
  - Represent $x$ and $y$ by automata $X$ and $Y$.
  - Compute $X' = \text{Project}_2(X \circ T_f)$ and $Y' = \text{Project}_1(T_f^{-1} \circ Y)$.
  - Compute best path of $X' \circ T_e \circ Y'$.
Factor Transducer

- **Definition**: transducer mapping a string to the set of factors of that string.
Factor Automaton of a String

- **Program:**
  ```
  fsmcompose X.fsm Tf.fsm | fsmproject -2 | fsmrmepsilon | fsmdeterminize | fsmminimize >Xp.fsm
  ```

- **Representation:** size linear in that of $X$. 

![Factor Automaton Diagram]
Local Alignment - Example

- **Program**: `fsmcompose Xp.fsm Te.fsm Yp.fsm | fsmbestpath -n1 >B.fsm`

- **Graphical representation**:

```
0\[C:C/-1.5]\rightarrow 1\[T:e/1]\rightarrow 2\[G:G/-1.5]\rightarrow 3\[A:A/-1.5]\rightarrow 4/0
```

- **Sequences**:
  - \(X = \text{T G T C T G A G}\)
  - \(Y = \text{A C A C G A}\)

  - Matching cost: \(-1.5\)
  - Insertion/deletion/substitution costs: \(+1\)
Alignment with Gaps

- **Problem**: Find the best local or global alignment with gaps (consecutive sequence of deletions/insertions) two sequences $x$ and $y$.

- **Motivation**: Ignore long gaps due to introns.

- **Example**: 
  \[
  \begin{align*}
  \text{A T G A } & \epsilon \epsilon G A C \\
  \text{A } & \epsilon \epsilon G A T G \epsilon C
  \end{align*}
  \]

- **Gap penalties**:
  - **Constant**: constant penalty $w_0$ per gap.
  - **Affine**: $w_0 + w_1 \times |\text{gap}|$
  - **Convex**: $w_0 \log |\text{gap}|$. 


Gap Alignment Transducer

- **Representation**: affine gap penalties.
More Complex Alignments

• Complex alignment transducers
  • Simpler alignment transducers can be combined using rational operations to create more complex ones.
  • Any weighted transducer can help define an alignment.
  • Alignment transducers can be learned from examples.
Learning Alignment Transducers

- **Problem**: given a fix topology for the transducer $T_e$, learn the transition weights from examples.

- **Example**: (Ristad and Yianilos, 1997)
  - learn edit costs of one-state edit-distance transducer;
  - data: large corpus of pairs of related sequences;
  - algorithm: use the Expectation-Maximization (EM) algorithm.
Alignment of Sets of Sequences

• **Problem:** Find the best alignment (minimal cost) of two sets of sequences $X$ and $Y$.

• **Note:** $X$ and $Y$ may be finite sets of biological sequences or infinite sets, e.g., represented or modeled by HMMs.

• **Solution:**
  • Construct edit transducer $T_e$.
  • Represent $X$ and $Y$ by automata $A_X$ and $A_Y$.
  • Compute best path of $A_X \circ T_e \circ A_Y$. 
Longest Common Subsequence of Automata

- **Problem:** Find the longest common subsequence of two (finite) sets of sequences $X$ and $Y$ each given by a finite automaton.

$$|\text{lcs}(X, Y)| = \max \{ |\text{lcs}(x, y)| : x \in X, y \in Y \}.$$ 

- **Solution:**
  
  - Construct LCS transducer $T_{\text{lcs}}$.
  
  - Represent $X$ and $Y$ by automata $A_X$ and $A_Y$.
  
  - Compute best path of $A_X \circ T_{\text{lcs}} \circ A_Y$. 
Longest Common Subsequence Transducer

- **Representation**: example.
Summary

• An alignment scoring function can be defined by a WFST.

• A general algorithm can be used to efficiently compute the best alignments between sequences, or sets of sequences represented by finite automata.

• Complex alignments can be learned from examples.

• The FSM library can be used for the implementation and computation of alignments.
Weighted Finite-State Transducers in Computational Biology

Corinna Cortes
Google Research
corinna@google.com

Mehryar Mohri
Courant Institute
mohri@cs.nyu.edu
This Tutorial

- Weighted finite-state transducers and software library (FSM Library)
- Pairwise alignments for bioinformatics
- Kernels for computational biology
Kernels for Computational Biology
Outline

• Introduction to machine learning
• Support vector machines
• Kernel methods
• Rational kernels
• Applications to computational biology
• Theory

http://www.cs.nyu.edu/~mohri/rational.html
Learning from Examples

Given data, learn mapping:

- Classification
  - Two-group classification
  - Multi-class classification
- Regression
  - one- or multi-dimensional
- Clustering
Remote Homology Problem

- **Problem**: assign protein sequences to families or superfamilies of proteins for which structural properties are known.

- **Data**: SCOP (Structural Classification of Proteins into superfamilies and families).

- **Classification**: 33 superfamilies.
Two-Group Classification

Protein Classification: Remote Homology Detection
(SCOP Data, http://scop.berkeley.edu/)

- **Allophycocyanin** ([Spirulina platensis])
  SIVTSKVNADAEARYLSPGELDRIKSFTSGERRVIAETMTGARERIIKQAGDQLFGKRPDVSPGGNAYGA
  DMTATCLRDLDDYLLRLITYGIVAGDVTPIEEIGVGVREMYKSLGTPIEAIAEGVRAKMSVATSLLSGADAAEAGSYFDYLIGAMS

- **Allophycocyanin** ([Spirulina platensis])
  MQDAITSVINSSDVQGKYLDASAIQKLKAYFATGELRVRAATTISANAAANIVKEAVAKSLLYSDVTRPGGNMYTT
  RRYAACIRDLDDYLLRLITYYAMLADPSILDERLVNLKETYNSLGVPIGATVQAIAQAMKEVTAGLVGGAGKEAGEM
  GIYFYDYICSGLS

- **Snake phospholipase A2** [Eastern cottonmouth snake (Agkistrodon piscivorus piscivorus)]
  SVLELGKMIQLETGKNAITSYGYSYGNCGWHRGQPKDATDRCCFVHKCCYKKLTDCHNHTDYSYSWKNK
  AIICEEKNPLCEMKCECDEKAVAILRENLTYNKKYKAYFKLCKCKKPDTCC

- **Snake phospholipase A2** [Agkistrodon piscivorus piscivorus]
  NLFQFEKLIKKMTGKSGMLWYSAYGCVGCGGQGRPKDATDRCCFVHDCCYGVHTGCNPKMDITYSVDDN
  GNIVCQQGNTNPCKQQICECDRAAAICFRDNKTYDSKTYWKYPKNCKESEEP

- **Bacterial chitobiase, n-terminal domain** [(Serratia marcescens)]
  DQQLVDQLSQLLKNVKMLDNRAENGVDCAILAGDWARSSCNRLVFLTSNDQQAIDGGKDWVIYFHSPRTQLRVD
  DNDQFKIHALTGDLYKLEPTAKFSFGPAKVAIEPVMAYWQLFRNDFLRWYATSGDACKPMLANTDENLDDQVFVAPFTGDQWHKRTKDKNLMTPASRFV

- **Tumor suppressor, DNA-binding domain** [Human (Homo sapiens)]
  SSSVPSQKTYQGSYFRLGFLHSGTAKSVTCTYSPALNMFCQLAKTCVPQWLVWDSTPPPGRTRAMAYKQ
  SQHMTEVRRCPHHERCSDDSLAPPQHLRVEGLRLVEYLDNRNTFRHHSVVYPYEPEVGSDCTTIHYNY
  MCNSSCMGGMNRRPILTIITLEDSSGNLLGRNSFEVRVCACPGRDRRTEEENL
Learning from Examples

Given labeled data. Divide into training and test data:

1. Chose learning machine and cost function;
2. Minimize error on training data;
3. Measure error on test data.
Overfitting

It is always possible to fit the training data, but the goal of learning is good generalization ability.
Overfitting

Capacity control:

• **Trees**: stopping criteria, pruning (pruning data set).

• **Neural networks**: number of hidden units, early stopping, weight decay.

• **Support vector machines**: built-in capacity control, no parameter tuning.
Outline

• Introduction to machine learning
• **Support vector machines**
• Kernel methods
• Rational kernels
• Applications to computational biology
• Theory

http://www.cs.nyu.edu/~mohri/rational.html
Support Vector Machines

• **Training data**, generated according to distribution $D$: $(x_1, y_1) \ldots (x_m, y_m) \in \mathbb{R}^N \times \{-1, +1\}$

• **Problem**: find function (classifier) $h : \mathbb{R}^N \rightarrow \{-1, +1\}$ with small generalization error over test data generated according to $D$.

• **Linear classification**:
  - Simple hypotheses: hyperplanes
  - Linear separation in high-dimensional feature space (Vapnik 1995; Boser et al., 1992)
  - Kernel methods
Linear Separation

- Functions: $h(x) = w \cdot x + b$, $w \in \mathbb{R}^N$, $b \in \mathbb{R}$
- Classifier: $\text{sign } h(x) \in \{-1, +1\}$
Optimal Hyperplane

- Canonical hyperplane: \( w \cdot x + b \in \{-1, +1\} \) for closest points.
- Margin: \( \rho = \frac{1}{\|w\|} \). For points on opposite sides of the margin: \(|w \cdot (x_2 - x_1)| = 2\).
Optimization Problem

- **Constrained optimization:**

\[
\min_x \frac{1}{2} \|w\|^2
\]

subject to \(y_i(x_i \cdot w + b) \geq 1, \quad i = 1, \ldots, m.\)

- **Solution:**

  - Quadratic optimization algorithms e.g., SMO algorithm (Platt, 1999)
Soft-Margin Hyperplane

- **Soft margin optimization problem** (Cortes & Vapnik, 1995):

\[
\min_{w, \xi} \frac{1}{2}||w||^2 + \frac{C}{m} \sum_{i=1}^{m} \xi_i
\]

subject to \( \xi_i \geq 0 \), and \( y_i(w \cdot x_i + b) \geq 1 - \xi_i, i = 1, \ldots, m \).
Non-Linear Separation

- Linear separation impossible in most problems
- **Non-linear mapping** from input space to high-dimensional feature space: $\Phi : X \rightarrow F$
- **Generalization ability**: independent of the dimension of $F$, depends only on margin and the number of examples
Outline

• Introduction to machine learning
• Support vector machines
• Kernel methods
• Rational kernels
• Applications to computational biology
• Theory

http://www.cs.nyu.edu/~mohri/rational.html
Kernel Methods

• **Idea:**
  • Define $K$ (called *kernel*) such that:
  
  $$\Phi(x) \cdot \Phi(y) = K(x, y)$$
  
  • $K$ often interpreted as a ‘similarity measure’

• **Benefits:**
  • **Efficiency:** $K$ may be much more efficient to compute than $\Phi$ and the dot product
  
  • **Flexibility:** $K$ can be chosen arbitrarily so long as the existence of $\Phi$ is guaranteed (Mercer’s condition).
Mercer’s Condition

- **Theorem:** Let $X \times X$ be a compact subset of $\mathbb{R}^N$ and let $K : X \times X \rightarrow \mathbb{R}$ be in $L_\infty(X \times X)$ and symmetric. Then, $K$ admits a uniformly convergent expansion:

$$K(x, y) = \sum_{n=0}^{\infty} a_n \phi_n(x) \phi_n(y), \text{ with } a_n > 0,$$

iff for any function $c$ in $L_2(X)$,

$$\int \int_{X \times X} c(x)c(y)K(x, y)dx\,dy \geq 0.$$
Positive Definite Symmetric Kernels

• Condition equivalent to Mercer’s condition in the discrete case: for all \( \{x_1, \ldots, x_n\} \subseteq X \), matrix \( K(x_i, x_j)_{i,j\leq n} \) is symmetric and:
  
  • is semi-definite positive: for all \( \{c_1, \ldots, c_n\} \subseteq \mathbb{R} \),
  
  \[
  \sum_{i,j=1}^{n} c_i c_j K(x_i, x_j) \geq 0
  \]

  • or, equivalently, has non-negative eigenvalues.
Example - Polynomial Kernels

• Definition:

\[ \forall x, y \in \mathbb{R}^N, K(x, y) = (x \cdot y + c)^d, \quad c > 0 \]

• Example: for \( N = 2 \) and \( d = 2 \),

\[ K(x, y) = (x_1 y_1 + x_2 y_2 + c)^2 \]

\[
\begin{pmatrix}
  x_1^2 \\
  x_2^2 \\
  \sqrt{2} x_1 x_2 \\
  \sqrt{2c} x_1 \\
  \sqrt{2c} x_2 \\
  c
\end{pmatrix}
\cdot
\begin{pmatrix}
  y_1^2 \\
  y_2^2 \\
  \sqrt{2} y_1 y_2 \\
  \sqrt{2c} y_1 \\
  \sqrt{2c} y_2 \\
  c
\end{pmatrix}
\]
XOR Problem

• Use second-degree polynomial kernel with $c = 1$:

\[
\begin{align*}
X_1 & = x_1 \times x_2 \\
\sqrt{2} X_1 X_2 & = x_1 x_2 \\
\sqrt{2} X_1 & = \sqrt{2} x_1 \\
\sqrt{2} X_2 & = \sqrt{2} x_2
\end{align*}
\]

- (1, 1, $\sqrt{2}$, $\sqrt{2}$, $\sqrt{2}$, 1)
- (1, 1, $\sqrt{2}$, $\sqrt{2}$, $\sqrt{2}$, 1)
- (1, 1, $\sqrt{2}$, $\sqrt{2}$, $\sqrt{2}$, 1)
- (1, 1, $\sqrt{2}$, $\sqrt{2}$, $\sqrt{2}$, 1)
- (1, 1, $\sqrt{2}$, $\sqrt{2}$, $\sqrt{2}$, 1)

- (1, 1, $\sqrt{2}$, $\sqrt{2}$, $\sqrt{2}$, 1)
- (1, 1, $\sqrt{2}$, $\sqrt{2}$, $\sqrt{2}$, 1)
- (1, 1, $\sqrt{2}$, $\sqrt{2}$, $\sqrt{2}$, 1)
- (1, 1, $\sqrt{2}$, $\sqrt{2}$, $\sqrt{2}$, 1)
- (1, 1, $\sqrt{2}$, $\sqrt{2}$, $\sqrt{2}$, 1)

Linearly non-separable

Linearly separable by

\[ x_1 x_2 = 0 \]
Other Standard PDS Kernels

- **Gaussian kernels:**

\[ K(x, y) = \exp\left( -\frac{||x - y||^2}{2\sigma^2} \right), \quad \sigma \neq 0 \]

- **Sigmoid Kernels:**

\[ K(x, y) = \tanh(a(x \cdot y) + b), \quad a \geq 0, b \geq 0 \]
Closure Properties of PDS Kernels

- **Theorem**: Positive definite symmetric (PDS) kernels are closed under:
  - sum,
  - product,
  - tensor product,
  - pointwise limit,
  - composition with a power series.
Kernel Applications -
Fixed-Size Feature Vectors

- **Tissue classification from microarray data:**


- 2-group classification problem, 2,000 to 97,802-dimensional input vectors, <50 patterns.

Kernel Applications - Fixed-Size Feature Vectors

- **Protein secondary structure prediction:**
    - 3-group classification problem. 120 dimensional input vectors.

- **Protein fold prediction:**
    - 27-group classification problem. 125 dimensional input vectors.

http://www.support-vector.net/bioinformatics.html
Outline

• Introduction to machine learning
• Support vector machines
• Kernel methods
• Rational kernels
• Applications to computational biology
• Theory

http://www.cs.nyu.edu/~mohri/rational.html
String Kernels

- Kernels defined over pairs of sequences

- **Motivation**: computational biology, text and speech classification

- **Idea**: two sequences are related when they share some common factors/subsequences

- **Example**: sum of the product of the counts of common factors
Example: Mismatch Kernel

- **Definition** (Leslie et al., 2002): Let $k$ and $m$ be non-negative integers with $m \leq k$. The $(k, m)$-mismatch kernel is the kernel defined over protein sequences $x$ and $y$ by:

$$K_{(k,m)}(x, y) = \sum_{z_1 \in F_k(x), z_2 \in F_k(y), z \in \Sigma^k} d_m(z_1, z) \cdot d_m(z, z_2)$$

where $F_k(x)$ is the set of all factors of $x$ of length $k$, and

$$d_m(z_1, z_2) = \begin{cases} 1 & |\text{mismatches}| \leq m \\ 0 & \text{otherwise.} \end{cases}$$
Weighted Transducers

\[ [T](x, y) = \text{Sum of the weights of all successful paths with input } x \text{ and output } y \]

\[ [T](abb, baa) = .1 \times .2 \times .3 \times .1 + .5 \times .3 \times .6 \times .1 \]
Unified Framework for String Kernels

- **Definition**: a kernel $K$ is *rational* if there exists a weighted transducer $T$ such that for all strings $x$ and $y$:

$$K(x, y) = [[T]](x, y)$$
Rational Kernels Over Weighted Automata

• **Definition:** a kernel $K$ is *rational* if there exists a weighted transducer $T$ such that for all weighted automata $A$ and $B$:

$$K(A, B) = \sum_{x,y} [[A]](x) \cdot [[T]](x, y) \cdot [[B]](y)$$

This definition can be generalized to the case of an arbitrary *semiring* (general operations).
Computation Algorithm

• Observation:

\[
\sum_{x,y} [[A]](x) \cdot [[T]](x, y) \cdot [[B]](y) = \sum_{x,y} [[A \circ T \circ B]](x, y)
\]

• Compute composed weighted transducer:

\[
U = A \circ T \circ B
\]

• Use single-source shortest-distance algorithm to compute the sum of the weights of all successful paths of \(U\).
Algorithm: Complexity

- **Automata case**: $O(|T||A||B|)$
- **String case**:
  - **General**: $O(|T||x||y|)$
  - **Specific, using failure functions**: $O(|x| + |y|)$
Transducer for Computing Counts

- $X$ may be a string or an automaton representing a regular expression.

- Alphabet $\Sigma = \{a, b\}$.
Transducer for Bigram Counts

Weighted Transducer $T, \Sigma = \{a, b\}$

$(A \circ T)$ computes the expected count of each bigram ($aa, ab, ba, bb$) in $A$. 
Transducer for Gappy Bigram Counts

Weighted Transducer $T', \Sigma = \{a, b\}$

$(A \circ T)$ computes the expected count of each ‘gappy bigram’ ($aa$, $ab$, $ba$, $bb$) in $A$, with gap penalty factor $\lambda$. 
Count-Based Similarity Measures

• Counts of sequences in $A$: $A \circ T$.

• Counts of sequences in $B$: $T^{-1} \circ B$.

• Sum of the product of counts of matching sequences:

$$A \circ (T \circ T^{-1}) \circ B$$

$\rightarrow$ Rational kernel $\equiv T \circ T^{-1}$
Representation by WFSTs - Example

Rational kernel \((T' \circ T'^{-1})\)

Representation of the kernel of (Lodhi et al., 2002).
Outline

• Introduction to machine learning
• Support vector machines
• Kernel methods
• Rational kernels
• Applications to computational biology
• Theory

http://www.cs.nyu.edu/~mohri/rational.html
String Kernel Applications (I): Protein Classification, Remote Homology Detection
Remote Homology Problem

- **Problem**: assign protein sequences to families or superfamilies of proteins for which structural properties are known.

- **Data**: SCOP (Structural Classification of Proteins into superfamilies and families).

- **Classification**: 33 superfamilies.
Two-Group Classification

Protein Classification: Remote Homology Detection (SCOP Data, http://scop.berkeley.edu/)

- **Allophycocyanin** ([Spirulina platensis])
  
  SIVTKSIVNADAEARYLSPGELDRIKSFVTSGERRVRIAETMTGARERIIKQAGDQLFGKRPDVSPPGGNAYGA
  
  DMTATCLRLDYYLRLITYGIVAGDVTPIEIEIGVGVREMYKSLGTPIEIAIEGVRAMKSATSLLSGADAAEAGSYFDYLIGAMS

- **Allophycocyanin** ([Spirulina platensis])
  
  MQDAITSVINSSDVQKYLDASAIQKLKAYFTGELRVRRAATTISANAAANIVKEAVAKSLLYSDVTRPGGNMYYTTR
  
  RRYAACIRDLDDYLLRATYAMLAGDPSILDERTVNLKETYNSLGVPIGATQAIQAMKEVTAGLNVGGGAGKEMGIYFDYICGSGLS

- **Snake phospholipase A2** [Eastern cottonmouth snake (Agkistrodon piscivorus piscivorus)]
  
  SVLELGKIQLTGKNAITSVYGCGWCWGRQPKDATDRCCFVHKCYYKKLTDNCNHKTDRYSWSWKH
  
  AIICEEKPNCLKEMCECDKAKAVAIKRENLDYNTKYYKAYFKLKDCKKPDTC

- **Snake phospholipase A2** [Eastern cottonmouth snake (Agkistrodon piscivorus piscivorus)]
  
  NLFQFEKLKMTGKSMALWASYGCGWGGQGRPKDATDRCCFVHDCGYKTVGCNPMDIYTYSVDNGNIVCGTTNPCKKQICECDRAAIICFRDNLKTDSKTWYKPPKNCKESEP

- **Bacterial chitobiase, n-terminal domain** ([Serratia marcescens])
  
  DQQLVDQLSQLKLNVKMLDNRAENGVDCAALGADWASCNVRLFTLSNDQQAIDGKDWVIYFHSPRQTLLRN
  
  DNDQFKIAHTGDLKLEPTAKFSFQAPQKAVEIPVVAEYWQLRFRNDFLPRWYATSGDAKPKMLANTTDENLDFQVFAPFTGDWKRKTDDKNILMTSPASRFV

- **Tumor supressor, DNA-binding domain** [Human (Homo sapiens)]
  
  SSSVPSQKTYQGSGYGFRLGFLHSGTAKSVTCTYSPALNMFCQLAKTCPVQLWVDSTPPGTRVRAMAIYKQ
  
  SQHMTEVRRCHCRCDSGDLAPPQHLIRVEGNLRVEYLDDRNTFRHSVVVPEPVEVGSDCTTIHYNYMCNSSCMGMGMRPRPILTITLDEDSSGNLLGRNSFEVRCACPGDRDRTEEENL
Example: Mismatch Kernel

- **Definition** (Leslie et al., 2002): Let $k$ and $m$ be non-negative integers with $m \leq k$. The $(k, m)$-mismatch kernel is the kernel defined over protein sequences $x$ and $y$ by:

$$K_{(k,m)}(x, y) = \sum_{z_1 \in F_k(x), z_2 \in F_k(y), z \in \Sigma^k} d_m(z_1, z) \cdot d_m(z, z_2)$$

where $F_k(x)$ is the set of all factors of $x$ of length $k$, and

$$d_m(z_1, z_2) = \begin{cases} 1 & |\text{mismatches}| \leq m \\ 0 & \text{otherwise.} \end{cases}$$
Rational Kernel Representation

- **Definition** *(Leslie et al., 2002)*: for sequences \(x\) and \(y\),

\[
K_{(k,m)}(x, y) = \sum_{z_1 \in F_k(x), z_2 \in F_k(y), z \in \Sigma^k} d_m(z_1, z) \cdot d_m(z, z_2)
\]

- **Representation**:

\[
K(3,2)
\]
Results
String Kernel Applications (2): Recognition of Translation Initiation Sites
Recognition of Translation Initiation Sites

- **Problem**: determine whether a start codon position in a DNA sequence is a translation initiation site (TIS).

- **Data**: (Pedersen and Nielsen, 1997), about 13,500 sequences; 3,300 true TIS.

- **Classification** problem.
Locality-Improved Kernel

• **Definition** (Zien et al, 2000): based on matching scores over windows of length $2l + 1$. The locality-improved kernel for two sequences $x$ and $y$ is defined by (can be combined with polyn. kernel):

$$K(x, y) = \sum_{p=1}^{m} \text{win}_p(x, y),$$

where $\text{win}_p$ is defined by:

$$\text{win}_p(x, y) = \left( \sum_{j=-l}^{+l} w_j \text{match}_{p+j}(x, y) \right)^{d_1}.$$
Rational Kernel Representation

- **Representation** \((l = 1)\):

\[
T \circ T^{-1}
\]
String Kernel Applications (3): Convolution Kernels
Convolution Kernels for Strings

• **Definition** (Haussler, 1999): $K_1$ a PDS rational transduction modeling substitutions, $K_2$ modeling insertions.

\[
K_H = (1 - \gamma)[K_2(\gamma K_1 K_2)^*]
\]

• **Representation:**

\[
\begin{align*}
0 & \xrightarrow{(1 - \gamma)K_2} 1 \\
1 & \xrightarrow{\gamma K_1} 2 \\
1 & \xrightarrow{K_2} 1
\end{align*}
\]
String Kernel Applications (4):
Edit-Distance,
Smith-Waterman
Edit-Distance

• **Theorem**: Let $\Sigma$ be a non-empty finite alphabet. $d$ is a symmetric rational kernel. But,

• $d$ is not a PDS kernel.

• $d$ is not *negative definite* if $|\Sigma| > 1$
Negative Definite Kernels

- **Definition:** Let $X$ be a non-empty set. A function $K: X \times X \rightarrow \mathbb{R}$ is said to be a *negative definite symmetric (NDS) kernel* if it is symmetric ($K(x, y) = K(y, x)$) and for all $\{x_1, \ldots, x_n\} \subseteq X, \{c_1, \ldots, c_n\} \subseteq \mathbb{R}$, with $\sum_{i=1}^{n} c_i = 0$,

$$
\sum_{i=1}^{n} c_i c_j K(x_i, x_j) \leq 0.
$$

- Clearly, if $K$ is a PDS kernel, then $-K$ is a NDS kernel but the converse is not true in general.
PDS and NDS Kernels

• **Theorem:** let $X$ be a non-empty set and let $K : X \times X \rightarrow \mathbb{R}$ be a symmetric kernel, then:

  • $K$ is a NDS kernel iff $\exp(-tK)$ is a PDS kernel for all $t > 0$;
Edit-Distance

Smallest eigenvalue of $\exp(-d)$ for $|\Sigma| = 2$
Relationship to Other Kernels in Bioinformatics

- Mismatch kernels (Leslie et al., 2003)
- Convolution kernels (Haussler, 1999)
- Path kernels (Takimoto and Warmuth, 2004)
- Edit-distance (Smith-Waterman)
- Many other string kernels
Outline

- Introduction to machine learning
- Support vector machines
- Kernel methods
- Rational kernels
- Applications to computational biology
- Theory

http://www.cs.nyu.edu/~mohri/rational.html
Positive Definite Symmetric (PDS) Rational Kernels: Theory

• How to construct a PDS rational kernel?
• Is there a characterization of PDS rational kernels?
• Can we combine PDS rational kernels?
PDS Rational Kernels: General Construction

- \( T \) arbitrary weighted transducer
- **Theorem:** \( T \circ T^{-1} \) defines a PDS rational kernel
- **Proof ideas:**
  - **Kernel:**
    \[
    K(x, y) = \sum_{z \in \Delta^*} [[T]](x, z) \cdot [[T]](y, z)
    \]
  - **Pointwise limit of:**
    \[
    K_n(x, y) = \sum_{|z| \leq n} [[T]](x, z) \cdot [[T]](y, z)
    \]
  - **Matrix**
    \[
    M_n = (K_n(x_i, x_j))_{i \leq l, j \leq l} = AA^t
    \]
  - **With:**
    \[
    A = [[T]](x_i, z_j)_{i \leq l, j \leq m}
    \]
**PDS Rational Kernels: Characterization?**

- **Theorem:** in the acyclic case, PDS rational kernels are of the form $T \circ T^{-1}$.

- **Theorem:** PDS rational kernels $S = T \circ T^{-1}$ are closed under sum, product, and Kleene-closure.

- **Conjecture:** all PDS rational kernels are of the form $T \circ T^{-1}$. 
PDS Rational Kernels: Closure Properties

- **Theorem:** PDS rational kernels are closed under sum, product, and Kleene-Closure.

- **Proof ideas** (product case):

\[
[[T_1 \cdot T_2]](x, y) = \sum_{x_1 x_2 = x, y_1 y_2 = y} [[T_1]](x_1, y_1) \cdot [[T_2]](x_2, y_2)
\]

\[
= \sum_{x_1 x_2 = x, y_1 y_2 = y} (T_1 \odot T_2)((x_1, x_2), (y_1, y_2))
\]

- \((T_1 \odot T_2)\) : tensor product is PDS. Thus, there exists a Hilbert Space \(H\) and a mapping \(u \rightarrow \phi_u\) such that:

\[
K_{T_1} \odot K_{T_2}(u, v) = \langle \phi_u, \phi_v \rangle
\]

- Thus,

\[
[[T_1 \cdot T_2]](x, y) = \left\langle \sum_{x_1 x_2 = x} \phi(x_1, x_2), \sum_{y_1 y_2 = y} \phi(y_1, y_2) \right\rangle
\]
Conclusion

• **Rational kernels:**
  - unifying framework for the design of kernels in computational biology;
  - theoretical foundation;
  - general and efficient algorithms;
  - can reproduce previous string kernels;
  - easy to visualize and understand;
  - can exploit weighted alternatives.