HASKER: An efficient algorithm for string kernels. Application to polarity classification in various languages

Marius Popescu\textsuperscript{1}, Cristian Grozea\textsuperscript{2}, Radu Tudor Ionescu\textsuperscript{1,○}

\begin{itemize}
  \item \textsuperscript{○}raducu.ionescu@gmail.com
  \item \textsuperscript{1}Department of Computer Science, University of Bucharest
  Bucharest, Romania
  \item \textsuperscript{2}VISCOM, Fraunhofer FOKUS
  Berlin, Germany
\end{itemize}

Marseille, KES 2017
September 7th, 2017
Table of Contents

• 1. Introduction
Table of Contents

• 1. Introduction
• 2. Motivation
Table of Contents

• 1. Introduction
• 2. Motivation
• 3. Basic Principles of Kernel Methods
## Table of Contents

- 1. Introduction
- 2. Motivation
- 3. Basic Principles of Kernel Methods
- 4. String Kernels
- 5. HASKER
- 6. Experiments
  - 6.1. Time Evaluation
  - 6.2. English Polarity Classification
  - 6.3. Arabic Polarity Classification
  - 6.4. Chinese Polarity Classification
- 7. Conclusion
Table of Contents

• 1. Introduction
• 2. Motivation
• 3. Basic Principles of Kernel Methods
• 4. String Kernels
• 5. HASKER
• 6. Experiments
Table of Contents

1. Introduction
2. Motivation
3. Basic Principles of Kernel Methods
4. String Kernels
5. HASKER
6. Experiments
   • 6.1. Time Evaluation
   • 6.2. English Polarity Classification
   • 6.3. Arabic Polarity Classification
   • 6.4. Chinese Polarity Classification

KES 2017  M. Popescu, C. Grozea, R.T. Ionescu  HASKER: An efficient algorithm for string kernels  2/31
Table of Contents

1. Introduction
2. Motivation
3. Basic Principles of Kernel Methods
4. String Kernels
5. HASKER
6. Experiments
   6.1. Time Evaluation
   6.2. English Polarity Classification
   6.3. Arabic Polarity Classification
   6.4. Chinese Polarity Classification
7. Conclusion
Outline

• 1. Introduction
• 2. Motivation
• 3. Basic Principles of Kernel Methods
• 4. String Kernels
• 5. HASKER
• 6. Experiments
  • 6.1. Time Evaluation
  • 6.2. English Polarity Classification
  • 6.3. Arabic Polarity Classification
  • 6.4. Chinese Polarity Classification
• 7. Conclusion
1. Introduction

- We present a simple and efficient algorithm for computing various spectrum string kernels
- We show that our algorithm is faster than the state-of-the-art Harry tool [Rieck et al., JMLR 2016] which is based on suffix trees
- We demonstrate that our approach can reach state-of-the-art performance for polarity classification in various languages
- We make our tool freely available online: http://string-kernels.herokuapp.com
1. Introduction

- We present a simple and efficient algorithm for computing various spectrum string kernels
- We show that our algorithm is faster than the state-of-the-art Harry tool [Rieck et al., JMLR 2016] which is based on suffix trees

We demonstrate that our approach can reach state-of-the-art performance for polarity classification in various languages.

We make our tool freely available online: http://string-kernels.herokuapp.com
1. Introduction

• We present a simple and efficient algorithm for computing various spectrum string kernels
• We show that our algorithm is faster than the state-of-the-art Harry tool [Rieck et al., JMLR 2016] which is based on suffix trees
• We demonstrate that our approach can reach state-of-the-art performance for polarity classification in various languages
• We make our tool freely available online: http://string-kernels.herokuapp.com
1. Introduction

• We present a simple and efficient algorithm for computing various spectrum string kernels
• We show that our algorithm is faster than the state-of-the-art Harry tool [Rieck et al., JMLR 2016] which is based on suffix trees
• We demonstrate that our approach can reach state-of-the-art performance for polarity classification in various languages
• We make our tool freely available online: http://string-kernels.herokuapp.com
Outline

1. Introduction
2. Motivation
3. Basic Principles of Kernel Methods
4. String Kernels
5. HASKER
6. Experiments
   - 6.1. Time Evaluation
   - 6.2. English Polarity Classification
   - 6.3. Arabic Polarity Classification
   - 6.4. Chinese Polarity Classification
7. Conclusion
2. Motivation

- Advantages of using string kernels:
  - Language independent [Ionescu et al., COLI 2016]
  - Linguistic theory neutral [Ionescu et al., COLI 2016]
  - Robust to topic variation [Ionescu et al., EMNLP 2014]
  - Improved state-of-the-art by 32% in a cross-corpus experiment (absolute value!)
2. Motivation

- Advantages of using string kernels:
  - language independent [Ionescu et al., COLI 2016] (only a set of labeled training samples is required)
  - linguistic theory neutral [Ionescu et al., EMNLP 2014] (improved state-of-the-art by 32\% in a cross-corpus experiment)
2. Motivation

Advantages of using string kernels:
- language independent [Ionescu et al., COLI 2016] (only a set of labeled training samples is required)
- linguistic theory neutral [Ionescu et al., COLI 2016] (we don’t even need to tokenize the text)
2. Motivation

- Advantages of using string kernels:
  - language independent [Ionescu et al., COLI 2016] (only a set of labeled training samples is required)
  - linguistic theory neutral [Ionescu et al., COLI 2016] (we don’t even need to tokenize the text)
  - robust to topic variation [Ionescu et al., EMNLP 2014] (improved state-of-the-art by 32% in a cross-corpus experiment)
2. Motivation

- Advantages of using string kernels:
  - language independent [Ionescu et al., COLI 2016] (only a set of labeled training samples is required)
  - linguistic theory neutral [Ionescu et al., COLI 2016] (we don’t even need to tokenize the text)
  - robust to topic variation [Ionescu et al., EMNLP 2014] (improved state-of-the-art by 32% in a cross-corpus experiment) – that is absolute value!
2. Motivation

- String kernels have demonstrated impressive performance levels in various competitions:
2. Motivation

• String kernels have demonstrated impressive performance levels in various competitions:
  • 1st place in PAN 2012 Traditional Authorship Attribution Tasks [Popescu & Grozea, CLEF 2012]
2. Motivation

- String kernels have demonstrated impressive performance levels in various competitions:
  - 1st place in PAN 2012 Traditional Authorship Attribution Tasks [Popescu & Grozea, CLEF 2012]
  - 3rd place in BEA8 2013 Native Language Identification Shared Task [Popescu & Ionescu, BEA8 2013]
  - 2nd place in VarDial 2016 Arabic Dialect Identification Shared Task [Ionescu & Popescu, VarDial 2016]
  - 1st place in VarDial 2017 Arabic Dialect Identification Shared Task [Ionescu & Butnaru, VarDial 2017]
  - 1st place in all three tracks of the BEA12 2017 Native Language Identification Shared Task [Ionescu & Popescu, BEA12 2017]
2. Motivation

• String kernels have demonstrated impressive performance levels in various competitions:
  • 1st place in PAN 2012 Traditional Authorship Attribution Tasks [Popescu & Grozea, CLEF 2012]
  • 3rd place in BEA8 2013 Native Language Identification Shared Task [Popescu & Ionescu, BEA8 2013]
  • 2nd place in VarDial 2016 Arabic Dialect Identification Shared Task [Ionescu & Popescu, VarDial 2016]
2. Motivation

- String kernels have demonstrated impressive performance levels in various competitions:
  - 1st place in PAN 2012 Traditional Authorship Attribution Tasks [Popescu & Grozea, CLEF 2012]
  - 3rd place in BEA8 2013 Native Language Identification Shared Task [Popescu & Ionescu, BEA8 2013]
  - 2nd place in VarDial 2016 Arabic Dialect Identification Shared Task [Ionescu & Popescu, VarDial 2016]
  - 1st place in VarDial 2017 Arabic Dialect Identification Shared Task [Ionescu & Butnaru, VarDial 2017]
2. Motivation

- String kernels have demonstrated impressive performance levels in various competitions:
  - 1st place in PAN 2012 Traditional Authorship Attribution Tasks [Popescu & Grozea, CLEF 2012]
  - 3rd place in BEA8 2013 Native Language Identification Shared Task [Popescu & Ionescu, BEA8 2013]
  - 2nd place in VarDial 2016 Arabic Dialect Identification Shared Task [Ionescu & Popescu, VarDial 2016]
  - 1st place in VarDial 2017 Arabic Dialect Identification Shared Task [Ionescu & Butnaru, VarDial 2017]
  - 1st place in all three tracks of the BEA12 2017 Native Language Identification Shared Task [Ionescu & Popescu, BEA12 2017]
Outline

1. Introduction
2. Motivation
3. Basic Principles of Kernel Methods
4. String Kernels
5. HASKER
6. Experiments
   - 6.1. Time Evaluation
   - 6.2. English Polarity Classification
   - 6.3. Arabic Polarity Classification
   - 6.4. Chinese Polarity Classification
7. Conclusion
3. Linear Classification in Primal Form

Features: \( f_1, f_2, f_3, f_4, f_5, f_6, f_7 \)

<table>
<thead>
<tr>
<th></th>
<th>( f_1 )</th>
<th>( f_2 )</th>
<th>( f_3 )</th>
<th>( f_4 )</th>
<th>( f_5 )</th>
<th>( f_6 )</th>
<th>( f_7 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_1 )</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( x_2 )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>( x_3 )</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>( x_4 )</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

\( X = L \)

Train samples: \( x_1, x_2, x_3, x_4 \)

Linear classifier: \( C = (w_1, w_2, w_3, w_4, w_5, w_6, w_7, b) \) such that \( \text{sign}(X * W' + b) = L \)

Test samples: \( y_1, y_2, y_3 \)

\( Y = P \)

Apply \( C \) to obtain predictions: \( P = \text{sign}(Y * W' + b) \)
3. Linear Classification in Primal Form

Features: \( f_1, f_2, f_3, f_4, f_5, f_6, f_7 \)

<table>
<thead>
<tr>
<th></th>
<th>( f_1 )</th>
<th>( f_2 )</th>
<th>( f_3 )</th>
<th>( f_4 )</th>
<th>( f_5 )</th>
<th>( f_6 )</th>
<th>( f_7 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_1 )</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( x_2 )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>( x_3 )</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>( x_4 )</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Train samples: \( x_1, x_2, x_3, x_4 \)

\[
\begin{align*}
X & = \begin{bmatrix}
4 & 0 & 2 & 5 & 3 & 0 & 1 \\
0 & 0 & 1 & 3 & 4 & 0 & 2 \\
2 & 1 & 0 & 0 & 1 & 2 & 5 \\
1 & 3 & 0 & 1 & 0 & 1 & 2 \\
\end{bmatrix} \\
L & = \begin{bmatrix}
1 \\
1 \\
-1 \\
-1 \\
\end{bmatrix}
\end{align*}
\]

Linear classifier: \( C = (w_1, w_2, w_3, w_4, w_5, w_6, w_7, b) \) such that \( \text{sign}(X \ast W' + b) = L \)

For example, the frequency of some p-gram in some sample

<table>
<thead>
<tr>
<th></th>
<th>( f_1 )</th>
<th>( f_2 )</th>
<th>( f_3 )</th>
<th>( f_4 )</th>
<th>( f_5 )</th>
<th>( f_6 )</th>
<th>( f_7 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y_1 )</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>( y_2 )</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>( y_3 )</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Test samples: \( y_1, y_2, y_3 \)

\[
\begin{align*}
Y & = \begin{bmatrix}
1 & 0 & 2 & 4 & 2 & 0 & 2 \\
1 & 2 & 0 & 1 & 2 & 2 & 1 \\
3 & 1 & 0 & 0 & 4 & 1 & 1 \\
\end{bmatrix} \\
P & = \begin{bmatrix}
? \\
? \\
? \\
\end{bmatrix}
\end{align*}
\]

Apply \( C \) to obtain predictions: \( P = \text{sign}(Y \ast W' + b) \)
3. Linear Classification in Dual Form

Kernel type: linear

Train samples: \( x_1, x_2, x_3, x_4 \)

\[
\begin{array}{cccc}
  x_1 & x_2 & x_3 & x_4 \\
  55 & 31 & 16 & 11 \\
  31 & 30 & 14 & 7 \\
  16 & 14 & 35 & 17 \\
  11 & 7 & 17 & 16 \\
\end{array}
\]

\[ x_1 = X \cdot X' = K_x \]

\[
\begin{array}{c}
  l_1 \\
  1 \\
  l_2 \\
  1 \\
  l_3 \\
  -1 \\
  l_4 \\
  -1 \\
\end{array}
\]

\[ = L \]

Linear classifier: \( C = (\alpha_1, \alpha_2, \alpha_3, \alpha_4, b) \) such that \( \text{sign}(K_x \cdot \alpha + b) = L \)

Test samples: \( y_1, y_2, y_3 \)

\[
\begin{array}{cccc}
  y_1 & y_2 & y_3 \\
  36 & 26 & 14 & 9 \\
  16 & 13 & 15 & 12 \\
  25 & 18 & 18 & 9 \\
\end{array}
\]

\[ y_1 = Y \cdot Y' = K_y \]

\[
\begin{array}{c}
  p_1 \\
  ? \\
  p_2 \\
  ? \\
  p_3 \\
  ? \\
\end{array}
\]

\[ = P \]

Apply \( C \) to obtain predictions: \( P = \text{sign}(K_y \cdot \alpha' + b) \)
3. Kernel Function

**Definition**
A kernel is a function $k$ that for all $x, z \in X$ satisfies

$$k(x, z) = \langle \phi(x), \phi(z) \rangle,$$

where $\phi$ is an embedding map from $X$ to an inner product feature space $F$

$$\phi : x \mapsto \phi(x) \in F$$
3. Embedding Map

- For the linear kernel, $\phi(x) = x$
3. Embedding Map

- For the linear kernel, $\phi(x) = x$
- We can replace the inner product with any similarity measure $s$, as long as the resulted matrix is symmetric and positive semi-definite:

$$k(x, z) = s(x, z)$$
3. Embedding Map

- For the linear kernel, $\phi(x) = x$
- We can replace the inner product with any similarity measure $s$, as long as the resulted matrix is symmetric and positive semi-definite:
  \[ k(x, z) = s(x, z) \]
- Hence, we no longer have to explicitly use the embedding map
3. Embedding Map

The function $\phi$ embeds the data into a feature space where the nonlinear relations now appear as linear.

- The function $\phi$ embeds the data into a feature space where the nonlinear relations now appear as linear.
3. Embedding Map

- The function $\phi$ embeds the data into a feature space where the nonlinear relations now appear as linear.
- Then, we can use a simple linear classifier to separate the samples.
3. Kernel Ridge Regression

- The Ridge Regression optimization criterion is given by the mean squared error (MSE) with a regularization term:

\[
\min_w \mathcal{L}_\lambda(w, S) = \min_w (\lambda \|w\|^2 + \sum_{i=1}^{n} (l_i - g(x_i))^2),
\]
3. Kernel Ridge Regression

- The Ridge Regression optimization criterion is given by the mean squared error (MSE) with a regularization term:

$$\min_w \mathcal{L}_\lambda(w, S) = \min_w (\lambda \|w\|^2 + \sum_{i=1}^{n} (l_i - g(x_i))^2),$$

where $S = \{(x_i, l_i) \mid x \in \mathbb{R}^m, l \in \mathbb{R}\}$ is the training set.
3. Kernel Ridge Regression

- The Ridge Regression optimization criterion is given by the mean squared error (MSE) with a regularization term:

\[
\min_w \mathcal{L}_\lambda(w, S) = \min_w (\lambda \|w\|^2 + \sum_{i=1}^{n} (l_i - g(x_i))^2),
\]

where \( S = \{(x_i, l_i) \mid x \in \mathbb{R}^m, l \in \mathbb{R}\} \) is the training set and \( g(x) = w'x = \sum_{i=1}^{m} w_i x_i \) is the prediction function.
3. Kernel Ridge Regression

- The Ridge Regression optimization criterion is given by the mean squared error (MSE) with a regularization term:

\[
\min_w \mathcal{L}_\lambda(w, S) = \min_w (\lambda \|w\|^2 + \sum_{i=1}^{n} (l_i - g(x_i))^2),
\]

where \( S = \{(x_i, l_i) \mid x \in \mathbb{R}^m, l \in \mathbb{R}\} \) is the training set and \( g(x) = w^\prime x = \sum_{i=1}^{m} w_i x_i \) is the prediction function.

- The optimal solution for \( w \) is given by:

\[
\frac{\partial \mathcal{L}_\lambda(w, S)}{\partial w} = \frac{\partial (\lambda \|w\|^2 + \sum_{i=1}^{n} (l_i - g(x_i))^2)}{\partial w} = 0
\]
3. Kernel Ridge Regression

• The optimal solution is given by:

\[
\frac{\partial L_\lambda(w, S)}{\partial w} = \frac{\partial (\lambda \|w\|^2 + (l - Xw)'(l - Xw))}{\partial w} = 2\lambda w - 2X'l + 2X'Xw = 0
\]
3. Kernel Ridge Regression

- The optimal solution is given by:

\[
\frac{\partial L_{\lambda}(w, S)}{\partial w} = \frac{\partial (\lambda \|w\|^2 + (l - Xw)'(l - Xw))}{\partial w} = 2\lambda w - 2X'l + 2X'Xw = 0
\]

- The optimal \(w\) is:

\[
X'Xw + \lambda w = X'l
\]
\[
(X'X + \lambda)w = X'l
\]
\[
w = (X'X + \lambda)^{-1}X'l
\]
3. Kernel Ridge Regression

- The optimal solution is given by:

\[
\frac{\partial L_\lambda(w, S)}{\partial w} = \frac{\partial (\lambda \|w\|^2 + (l - Xw)'(l - Xw))}{\partial w} = 2\lambda w - 2X'l + 2X'Xw = 0
\]

- The optimal \( w \) is:

\[
X'Xw + \lambda w = X'l \\
(X'X + \lambda)w = X'l \\
w = (X'X + \lambda)^{-1}X'l
\]

- Given that \( w = X'\alpha \) and \( K = XX' \), we obtain the dual solution:

\[
\alpha = (K + \lambda I_n)^{-1}l
\]
3. Kernel Ridge Regression

- The optimal solution is given by:

\[
\frac{\partial L_\lambda(w, S)}{\partial w} = \frac{\partial (\lambda ||w||^2 + (l - Xw)'(l - Xw))}{\partial w} \]

\[
= 2\lambda w - 2X' l + 2X'Xw = 0
\]

- The optimal \( w \) is:

\[
X'Xw + \lambda w = X' l \\
(X'X + \lambda)w = X' l \\
w = (X'X + \lambda)^{-1}X' l
\]

- Given that \( w = X'\alpha \) and \( K = XX' \), we obtain the dual solution:

\[
\alpha = (K + \lambda I_n)^{-1}l
\]

(only one line of code in Matlab)
1. Introduction
2. Motivation
3. Basic Principles of Kernel Methods
4. String Kernels
5. HASKER
6. Experiments
   • 6.1. Time Evaluation
   • 6.2. English Polarity Classification
   • 6.3. Arabic Polarity Classification
   • 6.4. Chinese Polarity Classification
7. Conclusion
4. String Kernels

Example

Given \( s = \text{“pineapple”} \) and \( t = \text{“apple pie”} \) over an alphabet \( \Sigma \), and the substring length \( p = 2 \), the sets of 2-grams that appear in \( s \) and \( t \) are denoted by \( S \) and \( T \), respectively:

\[
S = \{ \text{"pi"}, \text{"in"}, \text{"ne"}, \text{"ea"}, \text{"ap"}, \text{"pp"}, \text{"pl"}, \text{"le"} \}, \\
T = \{ \text{"ap"}, \text{"pp"}, \text{"pl"}, \text{"le"}, \text{"_"}, \text{"_p"}, \text{"pi"}, \text{"ie"} \}.
\]

The \( p \)-spectrum kernel between \( s \) and \( t \) can be computed as follows:

\[
k_2(s, t) = \sum_{v \in \Sigma^2} \text{num}_v(s) \cdot \text{num}_v(t),
\]

where \( \Sigma^2 = S \cup T \).

As the frequency of each 2-gram in \( s \) or \( t \) is not greater than one, the \( p \)-spectrum kernel is equal to \( |S \cap T| \), namely the number of common 2-grams among the two strings. Thus, \( k_2(s, t) = 5 \).
4. String Kernels

Example

Given \( s = \text{“pineapple”} \) and \( t = \text{“apple pie”} \) over an alphabet \( \Sigma \), and the substring length \( p = 2 \), the sets of 2-grams that appear in \( s \) and \( t \) are denoted by \( S \) and \( T \), respectively:

\[
S = \{ \text{pi}, \text{in}, \text{ne}, \text{ea}, \text{ap}, \text{pp}, \text{pl}, \text{le} \},
\]

\[
T = \{ \text{ap}, \text{pp}, \text{pl}, \text{le}, \text{e}_\text{p}, \text{pi}, \text{ie} \}.
\]

The \( p \)-spectrum kernel between \( s \) and \( t \) can be computed as follows:

\[
k_2(s, t) = \sum_{v \in \Sigma^2} \text{num}_v(s) \cdot \text{num}_v(t),
\]

where \( \Sigma^2 = S \cup T \).

As the frequency of each 2-gram in \( s \) or \( t \) is not greater than one, the \( p \)-spectrum kernel is equal to \( |S \cap T| \), namely the number of common 2-grams among the two strings:

\[
k_2(s, t) = 5.
\]
4. String Kernels

Example

Given $s = \text{“pineapple”}$ and $t = \text{“apple pie”}$ over an alphabet $\Sigma$, and the substring length $p = 2$, the sets of 2-grams that appear in $s$ and $t$ are denoted by $S$ and $T$, respectively:

$$S = \{ \text{“pi”}, \text{“in”}, \text{“ne”}, \text{“ea”}, \text{“ap”}, \text{“pp”}, \text{“pl”}, \text{“le”} \},$$

$$T = \{ \text{“ap”}, \text{“pp”}, \text{“pl”}, \text{“le”}, \text{“e_”}, \text{“_p”}, \text{“pi”}, \text{“ie”} \}.$$
4. String Kernels

Example

Given $s = \text{“pineapple”}$ and $t = \text{“apple pie”}$ over an alphabet $\Sigma$, and the substring length $p = 2$, the sets of 2-grams that appear in $s$ and $t$ are denoted by $S$ and $T$, respectively:

$S = \{ \text{“pi”}, \text{“in”}, \text{“ne”}, \text{“ea”}, \text{“ap”}, \text{“pp”}, \text{“pl”}, \text{“le”} \}$,

$T = \{ \text{“ap”}, \text{“pp”}, \text{“pl”}, \text{“le”}, \text{“e_”}, \text{“_p”}, \text{“pi”}, \text{“ie”} \}$

The $p$-spectrum kernel between $s$ and $t$ can be computed as follows:

$$k_2(s, t) = \sum_{v \in \Sigma^2} \text{num}_v(s) \cdot \text{num}_v(t),$$

where $\Sigma^2 = S \cup T$. 
4. String Kernels

Example

Given $s = \text{“pineapple”}$ and $t = \text{“apple pie”}$ over an alphabet $\Sigma$, and the substring length $p = 2$, the sets of 2-grams that appear in $s$ and $t$ are denoted by $S$ and $T$, respectively:

- $S = \{ \text{“pi”}, \text{“in”}, \text{“ne”}, \text{“ea”}, \text{“ap”}, \text{“pp”}, \text{“pl”}, \text{“le”} \}$,
- $T = \{ \text{“ap”}, \text{“pp”}, \text{“pl”}, \text{“le”}, \text{“e_”}, \text{“_p”}, \text{“pi”}, \text{“ie”} \}$

The $p$-spectrum kernel between $s$ and $t$ can be computed as follows:

$$k_2(s, t) = \sum_{v \in \Sigma^2} \text{num}_v(s) \cdot \text{num}_v(t),$$

where $\Sigma^2 = S \cup T$. As the frequency of each 2-gram in $s$ or $t$ is not greater than one, the $p$-spectrum kernel is equal to $|S \cap T|$, namely the number of common 2-grams among the two strings. Thus, $k_2(s, t) = 5$. 
4. String Kernels

- The character $p$-grams presence bits kernel is given by:

$$k_0^p(s, t) = \sum_{v \in \Sigma^p} \text{in}_v(s) \cdot \text{in}_v(t),$$

where $\text{in}_v(s)$ is 1 if string $v$ occurs as a substring in $s$, and 0 otherwise.
4. String Kernels

• The character $p$-grams presence bits kernel is given by:

$$k_{p}^{0/1}(s, t) = \sum_{v \in \Sigma^p} \text{in}_v(s) \cdot \text{in}_v(t),$$

where $\text{in}_v(s)$ is 1 if string $v$ occurs as a substring in $s$, and 0 otherwise.

• The intersection string kernel is defined as follows:

$$k_{p}^{\cap}(s, t) = \sum_{v \in \Sigma^p} \min\{\text{num}_v(s), \text{num}_v(t)\},$$

where $\text{num}_v(s)$ is the number of occurrences of string $v$ as a substring in $s$. 

HASKER: An efficient algorithm for string kernels
Outline

1. Introduction
2. Motivation
3. Basic Principles of Kernel Methods
4. String Kernels
5. HASKER
6. Experiments
   6.1. Time Evaluation
   6.2. English Polarity Classification
   6.3. Arabic Polarity Classification
   6.4. Chinese Polarity Classification
7. Conclusion
5. HASKER: HAsh map algorithm for String KERnels

• Phase 1: given two strings $s$ and $t$ as input, we first build a hash map $h$ that retains the occurrence counts of each $p$-gram in $s$:

$$h = \{ \text{"pi"} : 1, \text{"in"} : 1, \text{"ne"} : 1, \text{"ea"} : 1, \text{"ap"} : 1, \text{"pp"} : 1, \text{"pl"} : 1, \text{"le"} : 1 \}$$
5. HASKER: HAsh map algorithm for String KERnels

• Phase 1: given two strings $s$ and $t$ as input, we first build a hash map $h$ that retains the occurrence counts of each $p$-gram in $s$:

$$h = \{ \text{“pi”} : 1, \text{“in”} : 1, \text{“ne”} : 1, \text{“ea”} : 1,$$
$$\text{“ap”} : 1, \text{“pp”} : 1, \text{“pl”} : 1, \text{“le”} : 1 \}$$

• Phase 2: for each $p$-gram in $t$ that appears in $h$, we add the occurrence counts stored in $h$ to the similarity value $k$:

$$k \leftarrow k + h(t[i : i + p]), \forall i \in \{1, \ldots, |t| − p + 1\}$$
5. HASKER: HAsh map algorithm for String KERnels

- Phase 1: given two strings $s$ and $t$ as input, we first build a hash map $h$ that retains the occurrence counts of each $p$-gram in $s$:

$$h = \{ \text{“pi”} : 1, \text{“in”} : 1, \text{“ne”} : 1, \text{“ea”} : 1, \text{“ap”} : 1, \text{“pp”} : 1, \text{“pl”} : 1, \text{“le”} : 1 \}$$

- Phase 2: for each $p$-gram in $t$ that appears in $h$, we add the occurrence counts stored in $h$ to the similarity value $k$:

$$k \leftarrow k + h(t[i : i + p]), \forall i \in \{1, \ldots, |t| - p + 1\}$$

- Return: $k$ – the $p$-spectrum kernel between $x$ and $y$
5. HASKER: HAs$h$ map algorithm for String KERnels

• Time complexity: if hash table lookups are $O(1)$, our algorithm works in linear time with respect to the length of the strings
5. HASKER: Hash map algorithm for String KERnels

- Time complexity: if hash table lookups are $O(1)$, our algorithm works in linear time with respect to the length of the strings
- HASKER can easily be adapted for the presence bits string kernel or the intersection string kernel
5. HASKER: HAsh map algorithm for String KERnels

- Time complexity: if hash table lookups are $O(1)$, our algorithm works in linear time with respect to the length of the strings.
- HASKER can easily be adapted for the presence bits string kernel or the intersection string kernel.
- Our tool provides implementation for all these kernels: http://string-kernels.herokuapp.com
Outline

• 1. Introduction
• 2. Motivation
• 3. Basic Principles of Kernel Methods
• 4. String Kernels
• 5. HASKER
• 6. Experiments
  • 6.1. Time Evaluation
  • 6.2. English Polarity Classification
  • 6.3. Arabic Polarity Classification
  • 6.4. Chinese Polarity Classification
• 7. Conclusion
### 6.1. Time Evaluation

<table>
<thead>
<tr>
<th>Method</th>
<th>#strings</th>
<th>$p$-gram</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Harry</td>
<td>1000</td>
<td>3</td>
<td>132.6</td>
</tr>
<tr>
<td>HASKER (ours)</td>
<td>1000</td>
<td>3</td>
<td>35.5</td>
</tr>
<tr>
<td>Harry</td>
<td>1000</td>
<td>5</td>
<td>141.8</td>
</tr>
<tr>
<td>HASKER (ours)</td>
<td>1000</td>
<td>5</td>
<td>38.8</td>
</tr>
<tr>
<td>Harry</td>
<td>5000</td>
<td>3</td>
<td>3109.4</td>
</tr>
<tr>
<td>HASKER (ours)</td>
<td>5000</td>
<td>3</td>
<td>867.1</td>
</tr>
<tr>
<td>Harry</td>
<td>5000</td>
<td>5</td>
<td>3426.9</td>
</tr>
<tr>
<td>HASKER (ours)</td>
<td>5000</td>
<td>5</td>
<td>969.0</td>
</tr>
</tbody>
</table>

Table: Running times (seconds) of Harry [Rieck et al., JMLR 2016] versus HASKER

- Reported times are measured on a computer with Intel Core i7 2.3 GHz processor and 8 GB of RAM using one Core.
6.2. English Polarity Classification

Figure: Accuracy rates obtained by three types of string kernels on the IMDB Review official test set. Results are reported for $p$-grams in the range 5-10.
6.3. Arabic Polarity Classification

Figure: Accuracy rates obtained by three types of string kernels on the LABR test set. Results are reported for $p$-grams in the range 2-6.
6.4. Chinese Polarity Classification

Figure: Accuracy rates obtained by three types of string kernels on the PKU data set using a 10-fold cross-validation procedure. Results are reported for $p$-grams in the range $1-4$. 
6. Results Summary

<table>
<thead>
<tr>
<th>Corpus</th>
<th>Language</th>
<th>Method</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMDB</td>
<td>English</td>
<td>[Maas et al., ACL 2011](\Diamond)</td>
<td>88.9%</td>
</tr>
<tr>
<td>Review</td>
<td></td>
<td>[Le et al., ICML 2014](^*)</td>
<td>92.6%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KRR and (\hat{k}_{8+9+10}) (\Diamond)</td>
<td>90.7%</td>
</tr>
<tr>
<td>LABR</td>
<td>Arabic</td>
<td>[Nabil et al., ACL 2013](\Diamond,(^*)</td>
<td>82.7%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KRR and (\hat{k}_{3+4+5}) (\Diamond)</td>
<td>86.5%</td>
</tr>
<tr>
<td>PKU</td>
<td>Chinese</td>
<td>[Wan, EMNLP 2008](\Diamond)</td>
<td>86.1%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[Zhai et al., ESA 2011](^*)</td>
<td>94.1%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KRR and (\hat{k}_{1+2})</td>
<td>94.2%</td>
</tr>
</tbody>
</table>

Table: Results on all corpora. \(\Diamond\) – release best; \(^*\) – state-of-the-art.
Outline

• 1. Introduction
• 2. Motivation
• 3. Basic Principles of Kernel Methods
• 4. String Kernels
• 5. HASKER
• 6. Experiments
  • 6.1. Time Evaluation
  • 6.2. English Polarity Classification
  • 6.3. Arabic Polarity Classification
  • 6.4. Chinese Polarity Classification
• 7. Conclusion
7. Conclusions

• We presented an efficient algorithm for spectrum string kernels which is $4 \times$ faster than Harry [Rieck et al., JMLR 2016]
7. Conclusions

- We presented an efficient algorithm for spectrum string kernels which is 4× faster than Harry [Rieck et al., JMLR 2016]
- String kernels allowed automatic and implicit extraction of the linguistic knowledge needed to solve a difficult semantic task
7. Conclusions

- We presented an efficient algorithm for spectrum string kernels which is $4\times$ faster than Harry [Rieck et al., JMLR 2016]
- String kernels allowed automatic and implicit extraction of the linguistic knowledge needed to solve a difficult semantic task
- The requirements of our method are reduced to the bare minimum: a set of labeled text documents
Finally

- Thank you
- Questions