Learning Kernels over Strings using Gaussian Processes - Supplementary Material

A Derivations for the Vectorised Kernel and its Gradients

First, let's restate the original kernel equations from [Lodhi et al., 2002, Cancedda et al., 2003]. For this we use the following notation:

- A *string* $s = s_1, \ldots, s_{|s|}$ is a sequence of symbols, where $|s|$ is the length of the sequence.
- A concatenation of two strings s and t is denoted by st.
- A substring $s[i:j]$ is the substring $s_i \ldots s_j$ of s. For notation simplicity, we also define $s[:j] = s[1:j]$, the prefix of s up to symbol s_j, and $s[:-1] = s[1:|s|-1]$, the substring corresponding to all symbols in s except for the last one.

We also going to assume different decay hyperparameters for gaps and symbol matches and the existence of a similarity function sim between individual symbols. Given two strings s and t, the string kernel k_n of order n between these two strings is defined as follows:

$$
\begin{aligned}
k'_0(s,t) &= 1, \text{for all } s, t, \\
k'_i(s,t) &= \begin{cases} 0, & \min(|s|,|t|) < i \\ \lambda_g k'_i(s[:-1],t) + k''_i(s,t) & \text{otherwise} \end{cases} \\
k''_i(s,t) &= \begin{cases} 0, & \min(|s|,|t|) < i \\ \lambda_g k''_i(s,t[:-1]) + \lambda_m^2 \mathrm{sim}(s_{|s|},t_{|t|}) k'_{i-1}(s[:-1],t[:-1]) & \text{otherwise} \end{cases} \\
k_i(s,t) &= \begin{cases} 0, & \min(|s|,|t|) < i \\ k_i(s[:-1],t) + \sum_{j}^{|t|} \lambda_m^2 \mathrm{sim}(s_{|s|},t_j) k'_{i-1}(s[:-1],t[:j-1]) & \text{otherwise} \end{cases} \\
k_n(s,t) &= \sum_{i=1}^{n} \mu_i k_i(s,t)
\end{aligned}
$$

where λ_g and λ_m are the gap and match decay hyperparameters and μ_i is the weight for the intermediate gappy n-gram kernel for subsequences of length i. The original string kernel from [Lodhi et al., 2002] can be recovered by tying λ_g and λ_m, and by assuming the similarity function returns 1 if symbols are equal and 0 otherwise (hard match).

In the algorithm above, the match decay is only applied in conjunction with the similarity function, while the gap decay is part of the recursions between k'' and k'. Our vectorised version relies on two components:

- We precompute similarities over the symbols of the two strings, giving raise to a matrix $\mathbf{S}$ with dimensionality $|s| \times |t|$.

- We unroll the recursion between k'' and k'. This is done by first vectorising k' in the form of a matrix $\mathbf{K}'$, which contains the values for k' for each symbol pair. Then the recursion is unrolled by explicitly encoding the gap decay weights into a matrix $\mathbf{D}_\ell$ of dimensionality $\ell \times \ell$:

$$\mathbf{D}_\ell = \begin{bmatrix} 0 & \lambda_g^0 & \lambda_g^1 & \lambda_g^2 & \dots & \lambda_g^{|\ell|-2} \\ 0 & 0 & \lambda_g^0 & \lambda_g^1 & \dots & \lambda_g^{|\ell|-3} \\ 0 & 0 & 0 & \lambda_g^0 & \dots & \lambda_g^{|\ell|-4} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & \lambda_g^0 \\ 0 & 0 & 0 & 0 & \dots & 0 \end{bmatrix}$$

  This effectively adds redundant calculations to the algorithm but allows us to redefine it in terms of vector and matrix multiplications.

Having defined these matrices, the remainder of the algorithm can readily be vectorised and results in the version presented in the main paper, which we restate here:

$$\begin{aligned} \mathbf{S} &= \mathbf{E}_s \mathbf{E}_t^T, \\ \mathbf{K}'_0 &= \mathbf{1}, \\ \mathbf{K}'_i &= \mathbf{D}_{|s|} \mathbf{K}''_i \mathbf{D}_{|t|}, \\ \mathbf{K}''_i &= \lambda_m^2 (\mathbf{S} \odot \mathbf{K}'_{i-1}), \\ k_i &= \sum_{j,k} \lambda_m^2 (\mathbf{S} \odot \mathbf{K}'_{ijk}), \\ k(s,t) &= \boldsymbol{\mu}^T \mathbf{k}, \end{aligned}$$

where $\mathbf{E}_s$ and $\mathbf{E}_t$ are matrices of symbol embeddings for each string and $\odot$ is the Hadamard (element-wise) product.

From the vectorised equations we can see that the gradient with respect to $\boldsymbol{\mu}$ is simply $\mathbf{k}$, i.e., the vector of kernel evaluations for each subsequence length. For λ_m we obtain the equations

$$\begin{aligned} \frac{\partial \mathbf{K}'_0}{\partial \lambda_m} &= \mathbf{0}, \\ \frac{\partial \mathbf{K}'_i}{\partial \lambda_m} &= \mathbf{D}_{|s|} \frac{\partial \mathbf{K}''_i}{\partial \lambda_m} \mathbf{D}_{|t|}, \\ \frac{\partial \mathbf{K}''_i}{\partial \lambda_m} &= 2\lambda_m (\mathbf{S} \odot \mathbf{K}'_{i-1}) + \lambda_m^2 \left(\mathbf{S} \odot \frac{\partial \mathbf{K}_{i-1}}{\partial \lambda_m} \right), \\ \frac{\partial k_i}{\partial \lambda_m} &= 2\lambda_m \sum_{j,k} (\mathbf{S} \odot \mathbf{K}'_{ijk}) + \lambda_m^2 \left(\mathbf{S} \odot \frac{\partial \mathbf{K}_{ijk}}{\partial \lambda_m} \right), \\ \frac{\partial k}{\partial \lambda_m} &= \boldsymbol{\mu}^T \frac{\partial \mathbf{k}}{\partial \lambda_m}. \end{aligned}$$

And for λ_g we first define

$$\frac{\partial \mathbf{D}_\ell}{\partial \lambda_g} = \begin{bmatrix} 0 & 0 & \lambda_g^0 & 2\lambda_g^1 & 3\lambda_g^2 & \dots & (|\ell|-2)\lambda_g^{|\ell|-3} \\ 0 & 0 & 0 & \lambda_g^0 & 2\lambda_g^1 & \dots & (|\ell|-3)\lambda_g^{|\ell|-4} \\ 0 & 0 & 0 & 0 & \lambda_g^0 & \dots & (|\ell|-4)\lambda_g^{|\ell|-5} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \dots & \lambda_g^0 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 \end{bmatrix}.$$

These matrices are then plugged into the equations

$$\begin{aligned}
\frac{\partial \mathbf{K}'_0}{\partial \lambda_g} &= \mathbf{0}, \\
\frac{\partial \mathbf{K}'_i}{\partial \lambda_g} &= \frac{\partial \mathbf{D}_{|s|}}{\partial \lambda_g} \mathbf{K}''_i \mathbf{D}_{|t|} + \mathbf{D}_{|s|} \frac{\partial \mathbf{K}''_i}{\partial \lambda_g} \mathbf{D}_{|t|} + \mathbf{D}_{|s|} \mathbf{K}''_i \frac{\partial \mathbf{D}_{|t|}}{\partial \lambda_g}, \\
\frac{\partial \mathbf{K}''_i}{\partial \lambda_g} &= \lambda_m^2 \left(\mathbf{S} \odot \frac{\partial \mathbf{K}'_{i-1}}{\partial \lambda_g} \right), \\
\frac{\partial k_i}{\partial \lambda_g} &= \lambda_m^2 \sum_{j,k} \left(\mathbf{S} \odot \frac{\partial \mathbf{K}'_{ijk}}{\partial \lambda_g} \right), \\
\frac{\partial k}{\partial \lambda_g} &= \boldsymbol{\mu}^T \frac{\partial \mathbf{k}}{\partial \lambda_g}.
\end{aligned}$$

Notice that many terms in these equations are shared with the main kernel calculation. Our implementation use this fact to improve performance by calculating the kernel and its gradients in a single pass.

References

[Cancedda et al., 2003] Cancedda, N., Gaussier, E., Goutte, C., and Renders, J.-M. (2003). Word-Sequence Kernels. *The Journal of Machine Learning Research*, 3:1059–1082.

[Lodhi et al., 2002] Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., and Watkins, C. (2002). Text Classification using String Kernels. *The Journal of Machine Learning Research*, 2:419–444.