Learning Structural Kernels for Natural Language Processing

Learning Structural Kernels for Natural Language Processing

Daniel Beck
Department of Computer Science
University of Sheffield, United Kingdom
debeck1@sheffield.ac.uk

Trevor Cohn
Computing and Information Systems
University of Melbourne, Australia
t.cohn@unimelb.edu.au

Christian Hardmeier
Department of Linguistics and Philology
Uppsala University, Sweden
christian.hardmeier@lingfil.uu.se

Lucia Specia
Department of Computer Science
University of Sheffield, United Kingdom
l.specia@sheffield.ac.uk

Abstract

Structural kernels are a flexible learning paradigm that has been widely used in Natural Language Processing. However, the problem of model selection in kernel-based methods is usually overlooked. Previous approaches mostly rely on setting default values for kernel hyperparameters or using grid search, which is slow and coarse-grained. In contrast, Bayesian methods allow efficient model selection by maximizing the evidence on the training data through gradient-based methods. In this paper we show how to perform this in the context of structural kernels by using Gaussian Processes. Experimental results on tree kernels show that this procedure results in better prediction performance compared to hyperparameter optimization via grid search. The framework proposed in this paper can be adapted to other structures besides trees, e.g., strings and graphs, thereby extending the utility of kernel-based methods.

1 Introduction

Kernel-based methods are a staple machine learning approach in Natural Language Processing (NLP). Frequentist kernel methods like the Support Vector Machine (SVM) pushed the state of the art in many NLP tasks, especially classification and regression. One interesting aspect of kernels is their ability to be defined directly on structured objects like strings, trees and graphs. This approach has the potential to move the modelling effort from feature engineering to kernel engineering. This is useful when we do not have much prior knowledge about how the data behaves, as we can more readily define a similarity metric between inputs instead of trying to characterize which features are the best for the task at hand.

Kernels are a very flexible framework: they can be combined and parameterized in many different ways. Complex kernels, however, lead to the problem of *model selection*, where the aim is to obtain the best kernel configuration in terms of hyperparameter values. The usual approach for model selection in frequentist methods is to employ grid search on some development data disjoint from the training data. This approach can rapidly become impractical when using complex kernels which increase the number of model hyperparameters. Grid search also requires the user to explicitly set the grid values, making it difficult to fine tune the hyperparameters. Recent advances in model selection tackle some of these issues, but have several limitations (see §6 for details).

Our proposed approach for model selection relies on Gaussian Processes (GPs) (Rasmussen and Williams, 2006), a widely used Bayesian kernel machine. GPs allow efficient and fine-grained model selection by maximizing the evidence on the training data using gradient-based methods, dropping the requirement for development data. As a Bayesian procedure, GPs also naturally balance between model capacity and generalization. GPs have been shown to achieve state of the art performance in various regression tasks (Hensman et al., 2013; Cohn and Specia, 2013). Therefore, we base our approach on this framework.

While prediction performance is important to consider (as we show in our experiments), we are

Transactions of the Association for Computational Linguistics, vol. 3, pp. 461–473, 2015. Action Editor: Stefan Riezler.
Submission batch: 2/2015; Revision batch 7/2015; Published 8/2015.

mainly interested in two other significant aspects that are enabled by our approach:

- Gradient-based methods are more efficient than grid search for high dimensional spaces. This allows us to easily propose new rich kernel extensions that rely on a large number of hyperparameters, which in turn can result in better modelling capacity.
- Since the model selection process is now fine-grained, we can interpret the resulting hyperparameter values, depending on how the kernel is defined.

In this work we focus on tree kernels, which have been successfully used in a number of NLP tasks (see §6). In most cases, these kernels are used as an SVM component and model selection is not considered an important issue. Hyperparameters are usually set to default values, which work reasonably well in terms of prediction performance. However, this is only possible due to the small number of hyperparameters these kernels contain.

We perform experiments comprising synthetic data (§4) and two real NLP regression tasks: Emotion Analysis (§5.1) and Translation Quality Estimation (§5.2). Our findings show that our approach outperforms SVMs using the same kernels.

2 Gaussian Process Regression

Our definition of GPs closely follows that of Rasmussen and Williams (2006). Consider a setting where we have a dataset $\mathcal{X} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$, where $\mathbf{x}_i$ is a d-dimensional input and y_i the corresponding output. Our goal is to infer an underlying function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ to explain this data, i.e. $f(\mathbf{x}_i) \approx y_i$. Formally, f is drawn from a GP prior,

$$f(\mathbf{x}) \sim \mathcal{GP}(\mu(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')),$$

where $\mu(\mathbf{x})$ is the *mean* function, which is usually the $\mathbf{0}$ constant, and $k(\mathbf{x}, \mathbf{x}')$ is the *kernel* function.

In a regression setting, we assume that the response variables are noisy latent function evaluations, i.e., $y_i = f(\mathbf{x}_i) + \eta$, where $\eta \sim \mathcal{N}(0, \sigma_n^2)$ is added white noise. We assume a Gaussian likelihood, which allows us to obtain a closed formula solution for the posterior, namely

$$y_* \sim \mathcal{N}(\mathbf{k}_*(\mathbf{K} + \sigma_n\mathbf{I})^{-1}\mathbf{y}^T, \\ k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^T(\mathbf{K} + \sigma_n\mathbf{I})^{-1}\mathbf{k}_*),$$

where $\mathbf{x}_*$ and y_* are respectively the test input and its response variable, $\mathbf{K}$ is the Gram matrix corresponding to the training inputs and $\mathbf{k}_* = [\langle \mathbf{x}_1, \mathbf{x}_* \rangle, \langle \mathbf{x}_2, \mathbf{x}_* \rangle, \ldots, \langle \mathbf{x}_n, \mathbf{x}_* \rangle]$ is the vector of kernel evaluations between the test input and each training input.

A key property of GP models is their ability to perform efficient model selection. This is achieved by employing gradient-based methods to maximize the marginal likelihood,

$$p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) = \int p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}, f)p(f)df,$$

where $\boldsymbol{\theta}$ represents the vector of model hyperparameters and $\mathbf{y}$ is the vector of response variables from the training data. For a Gaussian likelihood, we can take the log of the expression above to obtain in closed-form[1],

$$\log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) = \\ \underbrace{-\frac{1}{2}\mathbf{y}^T\mathbf{G}^{-1}\mathbf{y}}_{\text{data fit}} \; \underbrace{-\frac{1}{2}\log|\mathbf{G}|}_{\text{complexity penalty}} \; \underbrace{-\frac{n}{2}\log 2\pi}_{\text{constant}}$$

where $\mathbf{G} = \mathbf{K} + \sigma_n\mathbf{I}$. The *data fit* term is dependent on the training response variables, while the *complexity penalty* term relies only on the kernel and training inputs. Since the first two terms have conflicting objectives, optimizing the log marginal likelihood will naturally achieve a compromise and thus limit overfitting (without the need for any validation step or additional data).

To enable gradient-based optimization we need to derive the gradients w.r.t. the hyperparameters:

$$\frac{\partial}{\partial\theta_j}\log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) = \frac{1}{2}\mathbf{y}^T\mathbf{G}^{-1}\frac{\partial\mathbf{G}}{\partial\theta_j}\mathbf{G}^{-1}\mathbf{y} \\ - \frac{1}{2}\,\text{trace}\left(\mathbf{G}^{-1}\frac{\partial\mathbf{G}}{\partial\theta_j}\right).$$

[1] See Rasmussen and Williams (2006, pp. 113-114) for details on the derivation of this formula and also its correspondent gradient calculation.

The gradients of $\mathbf{G}$ depend on the underlying kernel. Therefore we can employ any kind of valid kernel in this procedure as long as its gradients can be computed. This not only allows for fine-tuning of hyperparameters but also allows for kernel extensions which are richly parameterized.

3 Tree Kernels

The seminal work on Convolution Kernels by Haussler (1999) defines a broad class of kernels on discrete structures by counting and weighting the number of substructures they share. Applying Haussler's formulation to trees we reach a general formula for a tree kernel between two trees t_1 and t_2, namely

$$k(t_1, t_2) = \sum_{f \in \mathcal{F}} w(f) c_1(f) c_2(f), \qquad (1)$$

where $\mathcal{F}$ is the set of all tree fragments, $c_1(f)$ and $c_2(f)$ return the counts for fragment f in trees t_1 and t_2, respectively, and $w(f)$ assigns a weight to fragment f. In other words, we can consider the kernel a weighted dot product over vectors of fragment counts. The actual fragment set $\mathcal{F}$ is deliberately left undefined: different concepts of tree fragments define different tree kernels.

In this paper, we will focus on Subset Tree Kernels (henceforth SSTK), first introduced by Collins and Duffy (2001). This kernel considers tree fragments that contains complete grammar rules (see Figure 1 for an example). Consider the set of nodes in the two trees as N_1 and N_2 respectively. We define $I_i(n)$ as an indicator function that returns 1 if fragment $f_i \in \mathcal{F}$ has root n and 0 otherwise. A SSTK can then be defined as:

$$k(t_1, t_2) = \sum_{n_1 \in N_1} \sum_{n_2 \in N_2} \Delta(n_1, n_2), \qquad (2)$$

$$\text{where} \quad \Delta(n_1, n_2) = \sum_{i=1}^{|\mathcal{F}|} \lambda^{\frac{s(i)}{2}} I_i(n_1) I_i(n_2)$$

and $s(i)$ is the number of fragments in i with at least one child[2].

The formulation in Equation 2 is the same as the one shown in Equation 1, except that we are now restricting the weights $w(f)$ to be a function of a hyperparameter λ. The original goal of λ is to act as a decay factor that penalizes contributions from larger fragments cf smaller ones (and therefore, it should be in the $[0, 1]$ interval). Without this factor, the resulting distribution over tree pairs is skewed, giving extremely large values when trees are equal and rapidly decreasing for small differences over fragment counts. The decay factor helps to spread this distribution, effectively giving smaller weights to larger fragments.

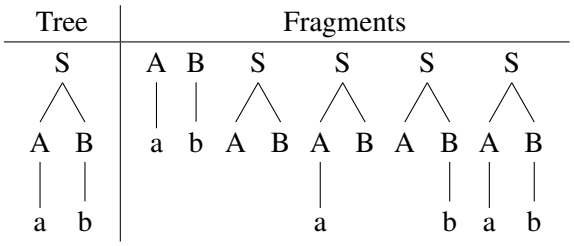

Figure 1: An example tree and the respective set of tree fragments defined by a SSTK.

The function Δ can be defined recursively,

$$\Delta(n_1, n_2) = \begin{cases} 0 & \text{pr}(n_1) \neq \text{pr}(n_2) \\ \lambda & \text{pr}(n_1) = \text{pr}(n_2) \wedge \\ & \text{preterm}(n_1) \\ \lambda g(n_1, n_2) & \text{otherwise,} \end{cases}$$

where pr(n) is the grammar production at node n and preterm(n) returns `true` if n is a pre-terminal node. The function g is defined as follows:

$$g(n_1, n_2) = \prod_{i=1}^{|n_1|} (\alpha + \Delta(c^i_{n_1}, c^i_{n_2})), \qquad (3)$$

where $|n|$ is the number of children of node n and c^i_n is the i^{th} child of node n. This recursive definition is calculated efficiently by employing dynamic programming to cache intermediate Δ results.

Equation 3 also adds another hyperparameter, α. This hyperparameter was introduced by Moschitti (2006b)[3] as a way to select between two different tree kernels. If $\alpha = 1$, we get the original SSTK, if $\alpha = 0$, then we obtain the Subtree Kernel, which only allows fragments with terminal symbols

[2] See Pighin and Moschitti (2010) for details and a proof on this derivation.

[3] In his original formulation, this hyperparameter was named σ but here we use α to not confuse it with the GP noise hyperparameter.

as leaves. We can also interpret the Subtree Kernel as a "sparse" version of the SSTK, where the "non-subtree" fragments have their weights equal to zero.

Even though fragment weights are affected by both kernel hyperparameters, previous work did not discuss their effects. The usual procedure fixes α to 1 (selecting the original SSTK) and sets λ to a default value (around 0.4). As explained in §2, the GP model selection procedure enables us to learn fine-grained values for these hyperparameters, which can lead to better performing models and aid interpretation. Furthermore, it also allows us to extend these kernels by adding new hyperparameters. We propose one such kernel in the next Section.

3.1 Symbol-aware Subset Tree Kernel

While varying the SSTK hyperparameters can lead to different weight schemes, they do that in a very coarse way. For some applications, it may be necessary to give more weight to specific fragments or set of fragments (e.g., NPs being more important than ADVP in an information extraction setting). The *Symbol-aware Subset Tree Kernel* (henceforth, SASSTK), which we introduce here, allows a more fine-grained control over the weights by employing one λ and one α hyperparameter *for each non-terminal symbol in the training data*. The calculation uses a similar recursive formula to the SSTK, namely:

$$\Delta(n_1, n_2) = \begin{cases} 0 & \text{pr}(n_1) \neq \text{pr}(n_2) \\ \lambda_x & \text{pr}(n_1) = \text{pr}(n_2) \wedge \\ & \quad \text{preterm}(n_1) \\ \lambda_x g_x(n_1, n_2) & \text{otherwise,} \end{cases}$$

where x is the symbol at node n_1 and

$$g_x(n_1, n_2) = \prod_{i=1}^{|n_1|} (\alpha_x + \Delta(c^i_{n_1}, c^i_{n_2})) \,. \qquad (4)$$

The SASSTK can be interpreted as a generalization of the SSTK: we can recover the latter by tying all λ and setting all $\alpha = 1$. By employing different hyperparameter values for each specific symbol, we can effectively modify the weights of all fragments where the symbol appears. Table 1 shows an example where we unrolled a kernel computation into its corresponding feature space, showing the resulting weighted counts for each feature.

	$\lambda_S = 1$ $\lambda_A = 1$ $\lambda_B = 1$	$\lambda_S = .5$ $\lambda_A = 1$ $\lambda_B = 1$	$\lambda_S = 2$ $\lambda_A = 1$ $\lambda_B = 1$
A → a	1	1	1
B → b	1	1	1
S → A B	1	0.5	2
S → (A a) B	1	0.5	2
S → A (B b)	1	0.5	2
S → (A a) (B b)	1	0.5	2
$k(t,t)$	6	3	18

Table 1: Resulting fragment weighted counts for the kernel evaluation $k(t,t)$, for different values of hyperparameters, where t is the tree in Figure 1.

3.2 Kernel Gradients

To enable hyperparameter optimization via gradient descent we must provide gradients for the kernels. In this Section we derive the gradients for SASSTK.

From Equation 2 we know that the kernel is a double summation over the Δ function. Therefore all gradients are also double summations, but over the gradients of Δ. We can obtain these in a vectorized way, by considering the gradients of the hyperparameter vectors $\boldsymbol{\lambda}$ and $\boldsymbol{\alpha}$ over Δ. Let k be the number of symbols considered in the model and $\boldsymbol{\lambda}$ and $\boldsymbol{\alpha}$ be k-dimensional vectors containing the respective hyperparameters.

In the following, we use the notation Δ_i as a shorthand for $\Delta(c^i_{n_1}, c^i_{n_2})$ and we also omit the parameters of g_x. We start with the $\boldsymbol{\lambda}$ gradient:

$$\frac{\partial \Delta}{\partial \boldsymbol{\lambda}} = \begin{cases} 0 & \text{pr}(n_1) \neq \text{pr}(n_2) \\ \boldsymbol{u} & \text{pr}(n_1) = \text{pr}(n_2) \wedge \\ & \quad \text{preterm}(n_1) \\ \dfrac{\partial(\lambda_x g_x)}{\partial \boldsymbol{\lambda}} & \text{otherwise,} \end{cases}$$

where x is the symbol at n_1, g_x is defined in Equation 4 and $\boldsymbol{u}$ is the k-dimensional unit vector with the element corresponding to symbol x equal to 1 and all others equal to 0. The gradient in the third case is defined recursively,

$$\begin{aligned} \frac{\partial(\lambda_x g_x)}{\partial \boldsymbol{\lambda}} &= \boldsymbol{u} g_x + \lambda_x \frac{\partial g_x}{\partial \boldsymbol{\lambda}} \\ &= \boldsymbol{u} g_x + \lambda_x \sum_{i=1}^{|n_1|} \frac{g_x}{\alpha_x + \Delta_i} \frac{\partial \Delta_i}{\partial \boldsymbol{\lambda}}. \end{aligned}$$

The $\boldsymbol{\alpha}$ gradient is derived in a similar way,

$$\frac{\partial \Delta}{\partial \boldsymbol{\alpha}} = \begin{cases} 0 & \text{pr}(n_1) \neq \text{pr}(n_2) \vee \\ & \text{preterm}(n_1) \\ \dfrac{\partial(\lambda_x g_x)}{\partial \boldsymbol{\alpha}} & \text{otherwise,} \end{cases}$$

and the gradient at the second case is also defined recursively,

$$\begin{aligned} \frac{\partial(\lambda_x g_x)}{\partial \boldsymbol{\alpha}} &= \lambda_x \frac{\partial g_x}{\partial \boldsymbol{\alpha}} \\ &= \lambda_x \sum_{i=1}^{|n_1|} \frac{g_x}{\alpha_x + \Delta_i} \left(\boldsymbol{u} + \frac{\partial \Delta_i}{\partial \boldsymbol{\alpha}} \right). \end{aligned}$$

Gradients can be efficiently obtained using dynamic programming. In fact, they can be calculated at the same time as Δ to improve performance since they all share many terms in their derivations. Finally, we can easily obtain the gradients for the original SSTK by letting $\boldsymbol{u} = 1$.

3.3 Kernel Normalization

It is common practice when using tree kernels to normalize the kernel. This helps reduce the random effect of tree size. Normalization can be achieved using the following, where $\hat{k}$ is the normalized kernel:

$$\hat{k}(t_1, t_2) = \frac{k(t_1, t_2)}{\sqrt{k(t_1, t_1) k(t_2, t_2)}}.$$

To apply this normalized version in the optimization procedure we must also derive gradients for the normalization function. In the following equation, we use k_{ij} and $\hat{k}_{ij}$ as a shorthand for $k(t_i, t_j)$ and $\hat{k}(t_i, t_j)$, respectively:

$$\frac{\partial \hat{k}_{12}}{\partial \boldsymbol{\theta}} = \frac{\dfrac{\partial k_{12}}{\partial \boldsymbol{\theta}}}{\sqrt{k_{11} k_{22}}} - \hat{k}_{12} \frac{\dfrac{\partial k_{11}}{\partial \boldsymbol{\theta}} k_{22} + k_{11} \dfrac{\partial k_{22}}{\partial \boldsymbol{\theta}}}{2 k_{11} k_{22}}.$$

3.4 Other Extensions

Many other structural kernels rely on recursive definitions and dynamic programming to perform their calculations. Examples include other tree kernels like the Partial Tree Kernel (Moschitti, 2006a) and string kernels like the ones defined on character n-grams (Lodhi et al., 2002) or word sequences (Cancedda et al., 2003). While in this paper we focus on the SSTK (and our proposed SASSTK), our approach can easily be extended to these other kernels, as long as all the corresponding recursive definitions are differentiable.

4 Synthetic Data Experiments

A natural question that arises in the proposed method is how much data is needed to accurately learn the kernel hyperparameters. To answer this question, we run a set of experiments using synthetic data. We generate this data by using a set of 1000 natural language syntactic trees, where we fix a random subset of 200 instances for testing and use the remaining 800 instances as training. For each training set size we define a GP over the full dataset, sample a function from it and use the function output as the response variable for each tree. We try two different GP priors, one using the SSTK and another one using the SASSTK.

The conditions above provide a controlled environment to check the modelling capacities of our approach since we know the exact distribution where the data comes from. The reasoning behind these experiments is that to be able to provide benefits in real tasks, where the data distribution is not known, our models have to be learnable in this controlled setting as well using a reasonable amount of data.

Finally, we also provide an empirical evaluation comparing the speed performance between our approach and grid search.

4.1 SSTK Prior

Our first experiments use a SSTK as the kernel with $\lambda = 0.001, \alpha = 1$ and $\sigma_n^2 = 0.01$. After obtaining the input trees and their sampled labels, we define a new GP model using only the training data plus the obtained response variables, this time using a SSTK with randomized hyperparameter values. Then we optimize the GP and check if the learned hyperparameters are close to the original ones, using 10 random restarts to limit the effect of local optima. We also use the optimized GP to predict response variables on the test set and measure Root Mean Squared Error (RMSE). Our hypothesis is that with a reasonable sample size we can retrieve the original hyperparameter values and obtain low RMSE. For each training set size, we repeat the experiment 20 times.

Figure 2 shows the results of these experiments. For small sizes the variance in the resulting hyperparameter values is large but as soon as we reach 200 instances we are able to retrieve the original values with high confidence. In other words, in an ideal setting 200 instances are enough to learn the kernel. It is also interesting to note that test RMSE after optimization steadily decreases as we increase training data size. This shows that if one is more interested in predictions themselves, it is still worth optimizing hyperparameters even if the training data is small.

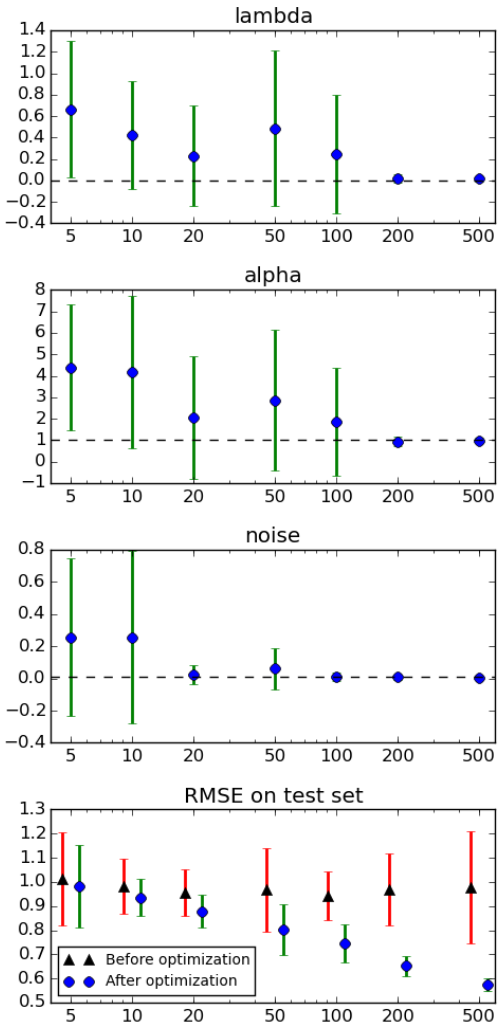

Figure 2: Results of synthetic experiments optimizing SSTK. The x axes correspond to different training set sizes and the the y axes are the obtained hyperparameter values in the first three plots and RMSE in the last plot. Dashed lines show the original hyperparameter values. Points are offset in RMSE chart for legibility.

4.2 SASSTK Prior

The large number of hyperparameters of the SASSTK makes it more prone to optimization and overfitting issues when compared to the SSTK. This raises the question of how much data is needed to justify its use. To address this question, we run similar experiments to those above for the SSTK, except that now we sample from a GP using a SASSTK as the kernel.

Instead of optimizing all hyperparameters freely we use a simpler version where we tie λ and α for each symbol to the same value, except for the symbol 'S'. Effectively this version has one extra λ and one extra α (henceforth λ_S and α_S) when compared to the SSTK. The GP prior hyperparameter values are set to $\lambda = 0.001, \lambda_S = 0.5, \alpha = 0.1, \alpha_S = 1$ and $\sigma_n^2 = 0.01$. For each training set size, we train two GPs, one using this SASSTK and one using the original SSTK, optimize them using 10 random restarts and measure RMSE on the test set.

Results are shown in Figure 3. For all training set sizes the SASSTK reaches lower RMSE than SSTK, with a substantial difference after reaching 100 instances. This shows that even for small datasets our proposed kernel manages to capture aspects which can not be explained by the original SSTK. Note that this is an ideal setting, and real datasets may need to be larger to realize gains from SASSTK. Nevertheless, these are promising results since they give evidence of a small lower bound on the dataset size for SASSTK to be effective.

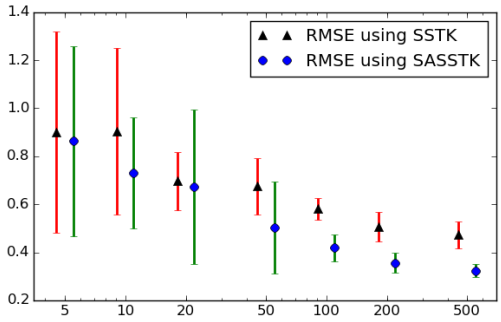

Figure 3: Results from synthetic experiments comparing SSTK and SASSTK. The x axis is training set size while the y axis corresponds to RMSE.

4.3 Performance Experiments

To provide an overview of how efficient is the gradient-based method compared to grid search we also run a set of experiments measuring wall clock training time vs. RMSE on a test set. For both GP and SVM models we employ the SSTK as the kernel and we use the same synthetic data from the previous experiments[4]. We perform 20 runs, keeping the test set as the same 200 instances for all runs and randomly sampling 200 instances from the remaining instances as training data.

Figure 4 shows the curves for both GP and SVM models. The GP curve is obtained by increasing the maximum number of iterations of the gradient-based method (in this case, L-BFGS) and the SVM curve is obtained by increasing the granularity of the grid size.

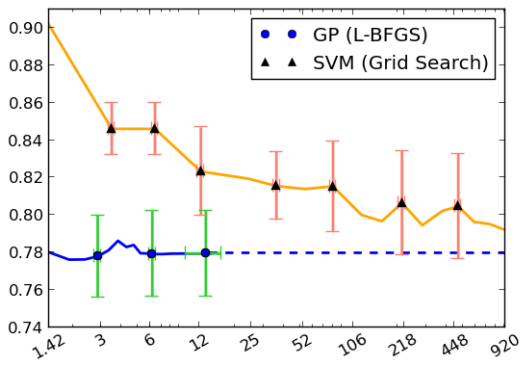

Figure 4: Results from performance experiments. The x axis corresponds to wall clock time in seconds and it is in log scale. The y axis shows RMSE on the test set. The blue dashed line corresponds to the RMSE value obtained after L-BFGS converged. Error bars are obtained by measuring one standard deviation over the 20 runs made in each experiment.

We can see that optimizing the GP model is consistently much faster than doing grid search on the SVM model (notice the logarithmic scale), even though it shows some variance when letting L-BFGS run for a larger number of iterations. The GP model also is able to better predictions in general. Even when taking the variances into account, grid search would still need around 10 times more computation time to achieve the same predictions obtained by the GP model. In real settings, SVMs predictions tend to be more on par with the ones provided by a GP (as shown in §5) but nevertheless these figures show that the GP can be much more time efficient when optimizing hyperparameters of a tree kernel.

An important performance aspect to take into account is parallelization. Grid search is embarassingly parallelizable since each grid point can run in a different core. However, the GP optimization can also benefit from multiple cores by running each kernel computation inside the Gram matrix in parallel. To keep the comparisons simpler, the results shown in this section use a single core but all experiments in §5 employ parallelization in the Gram matrix computation level (for both SVM and GP models).

5 NLP Experiments

Our experiments with NLP data address two regression tasks: Emotion Analysis and Quality Estimation. For both tasks, we use the Stanford parser (Manning et al., 2014) to obtain constituency trees for all sentences. Also, rather than using data official splits, we perform 5-fold cross-validation in order to obtain more reliable results.

5.1 Emotion Analysis

The goal of Emotion Analysis is to automatically detect emotions in a text (Strapparava and Mihalcea, 2008). This problem is closely related to Opinion Mining (Pang and Lee, 2008), with similar applications, but it is usually done at a more fine-grained level and involves the prediction of a set of labels for each text (one for each emotion) instead of a single label.

Beck et al. (2014a) used a multi-task GP for this task with a bag-of-words feature representation. In theory, it is possible to combine their multi-task kernel with our tree kernels, but to keep the focus of the experiments on testing tree kernel approaches, here we use independently trained models, one per emotion.

Dataset We use the dataset provided by the "Affective Text" shared task in SemEval2007 (Strapparava and Mihalcea, 2007), which is composed of 1000 news headlines annotated in terms of six emotions: *Anger, Disgust, Fear, Joy, Sadness* and *Sur-*

[4]For specific details on the SVM models used in all experiments performed in this paper we refer the reader to Appendix A.

prise. For each emotion, a score between 0 and 100 is given, 0 meaning total lack of emotion and 100, maximally emotional. Scores are mean-normalized before training the models.

Models We perform experiments using the following tree kernels:

- **SSTK:** the SSTK formulation introduced by Moschitti (2006b);
- **SASSTK$_{\text{full}}$:** our proposed Symbol-Aware SSTK;
- **SASSTK$_S$:** same as before, but using only two λ and two α hyperparameters: one for symbols corresponding to full sentences[5] and another for all other symbols. This configuration is similar to that in Section 4.2.

For all kernels, we also use a variation fixing the α hyperparameters to 1 to emulate the original SSTK.

Baselines and evaluation Our results are compared against three baselines:

- **SVM SSTK:** a SVM using an SSTK kernel.
- **SVM BOW:** same as before, but using an RBF kernel with a bag-of-words representation.
- **GP BOW:** same as SVM BOW but using a GP instead.

The SVM models are trained using a wrapper for LIBSVM[6] (Chang and Lin, 2001) provided by the scikit-learn toolkit[7] (Pedregosa et al., 2011) and optimized via grid search. Following previous work, we use Pearson's correlation coefficient as evaluation metric. Pearson's scores are obtained by concatenating all six emotions outputs together.

Table 2 shows the results. The best GP model with tree kernels outperforms the SVMs, showing that the fine-grained model selection procedure provided by the GP models is helpful when dealing with tree kernels. However, using the SASSTK models do not help in the case of free α and the SASSTK$_{\text{full}}$ actually performs worse than the original SSTK, even though the optimized marginal likelihood was higher. This is evidence that the SASSTK$_{\text{full}}$ model is overfitting the training data, probably due to its large number of hyperparameters.

	Pearson's
SVM BOW	0.5690
SVM SSTK	0.5254
GP BOW	0.5891
(free α)	
GP SSTK	0.5713
GP SASSTK$_{\text{full}}$	0.5118
GP SASSTK$_S$	0.5710
(fixed $\alpha = 1$)	
GP SSTK	0.5093
GP SASSTK$_{\text{full}}$	0.5435
GP SASSTK$_S$	0.5225

Table 2: Pearson's correlation scores for the Emotion Analysis task (higher is better).

Another interesting finding in Table 2 is that fixing the α values often harms performance. Inspecting the free α models showed that the values found by the optimizer were very close to zero. This indicates that the model selection procedure prefer towards giving smaller weights to incomplete tree fragments. We can interpret this as the model selecting a more lexicalized feature space, which also explains why the GP RBF model on bag-of-words performed the best in this task.

Finally, to understand how the optimized kernels could provide more interpretability, Table 3 shows the top 15 λ values obtained by the SASSTK$_{\text{full}}$ (fixed α variant) with their corresponding symbols. In this specific case the kernel does not give the best performance so there are limitations in doing a full linguistic analysis. Nevertheless, we believe this example shows the potential for developing more interpretable kernels. This is especially interesting because these models take into account a much richer feature space than what it is allowed by parametric models.

5.2 Quality Estimation

The goal of Quality Estimation is to provide a quality prediction for new, unseen machine translated texts (Blatz et al., 2004; Bojar et al., 2014). Exam-

[5] In this dataset, symbols are S, SQ, $SBARQ$ and $SINV$.

[6] http://www.csie.ntu.edu.tw/~cjlin/libsvm

[7] http://scikit-learn.org

JJR	0.8333	WHADVP	0.5004	VBP	0.4653
PRP$	0.6933	QP	0.5001	WHNP	0.4508
WDT	0.6578	JJS	0.4996	NN	0.4274
RBR	0.5445	NNS	0.4961	JJ	0.4021
VBG	0.5163	.	0.4777	SQ	0.4000

Table 3: Top 15 symbols sorted according to their obtained λ values in the $\text{SASSTK}_{\text{full}}$ model with fixed α. The numbers are the corresponding λ values, averaged over all six emotions.

ples of applications include filtering machine translated sentences that would require more post-editing effort than translation from scratch (Specia et al., 2009), selecting the best translation from different MT systems (Specia et al., 2010) or between an MT system and a translation memory (He et al., 2010), and highlighting segments that need revision (Bach et al., 2011). While various quality metrics exist, here we focus on *post-editing time* prediction.

Tree kernels have been used before in this task (with SVMs) by Hardmeier (2011) and Hardmeier et al. (2012). While their best models combine tree kernels with a set of explicit features, they also show good results using only the tree kernels. This makes Quality Estimation a good benchmark task to test our models.

Datasets We use two publicly available datasets containing post-edited machine translated sentences. Both are composed of a set of source sentences, their machine translated outputs and the corresponding post-editing time.

- **French-English (*fr-en*):** This dataset, described in (Specia, 2011), contains 2524 French sentences translated into English and post-edited by a novice translator.

- **English-Spanish (*en-es*):** This dataset was used in the WMT14 Quality Estimation shared task (Bojar et al., 2014), containing 858 sentences translated from English into Spanish and post-edited by an expert translator.

For each dataset, post-editing times are first divided by the translation output length (obtaining the *post-editing time per word*) and then mean normalized.

Models Since our data consists of pairs of trees, our models in this task use a pair of tree kernels. We combine these two kernels by either summing or multiplying them. As for underlying tree kernels, we try both SSTK and SASSTK_S. As in the Emotion Analysis task, we also experiment with a set of kernel configurations with the α hyperparameters fixed at 1. We also test models that combine our tree kernels with an RBF kernel on a set of 17 features extracted using the QuEst framework (Specia et al., 2013). These features are part of a strong baseline model used by the WMT14 shared task.

Baselines and evaluation We compare our results with a number of SVM models:

- **SVM SSTK:** same as in the Emotion Analysis task, using either a sum ($+$) or a product ($\times$) of SSTKs.

- **SVM RBF:** this is an SVM trained on the 17 features extracted by Quest.

- **SVM RBF SSTK:** a combination of the two models above.

For further comparison, we also show results obtained using a GP model and an RBF kernel on the QuEst-only features. Following previous work, we measure prediction performance using both Mean Absolute Error (MAE) and RMSE.

The prediction results are given in Table 4. They indicate a number of interesting findings:

- For the *fr-en* dataset, the GP models combining tree kernels with an RBF kernel outperform all other models. Results for the *en-es* dataset are less consistent, probably due to the small size of the dataset, but on average they are better than their SVM counterparts.

- The SVMs using a combination of kernels performs worse than using the RBF kernel alone. Inspecting the models, we found that grid search actually harms performance. For instance, for the *fr-en* dataset, MAE and RMSE for the RBF + SSTK $\times$ model before grid search are 0.4681 and 0.6016, respectively. On the other hand, for this dataset all GP models achieve better results after optimization.

- Unlike in the Emotion Analysis task, fixing α results in better performance, even though the resulting models have lower marginal likelihood than the ones with free α. The same effect happened when comparing the SASSTK models with the SSTK ones for the *en-es* dataset. Both cases are evidence of model overfitting.

	French-English		English-Spanish	
	MAE	RMSE	MAE	RMSE
(SVM)				
RBF	0.4610	0.5944	0.7831	1.0238
SSTK +	0.4710	0.6006	0.7777	1.0820
SSTK ×	0.4681	0.6016	0.7884	1.1044
RBF SSTK +	0.5146	0.6267	0.8077	1.0295
RBF SSTK ×	0.5186	0.6299	0.8367	1.0427
GP RBF	0.4555	0.5830	0.7842	1.0735
(GP free α)				
SSTK +	0.4789	0.5912	0.7551	1.0281
SSTK ×	0.4804	0.5843	0.7440	1.0008
SASSTK$_S$ +	0.4756	0.5889	0.8096	1.0754
SASSTK$_S$ ×	0.4797	0.5868	0.7484	1.0102
(GP fixed $\alpha = 1$)				
SSTK +	0.4694	0.5808	0.7614	1.0019
SSTK ×	0.4708	0.5733	**0.7205**	**0.9870**
SASSTK$_S$ +	0.4758	0.5888	0.8242	1.0912
SASSTK$_S$ ×	0.4699	0.5751	0.7469	1.0280
(GP fixed $\alpha = 1$)				
RBF SSTK +	0.4408	0.5651	0.7591	1.0469
RBF SSTK ×	0.4443	0.5659	0.7389	1.0302
RBF SASSTK$_S$ +	**0.4406**	**0.5648**	0.7692	1.0682
RBF SASSTK$_S$ ×	0.4440	0.5658	0.7682	1.0628

Table 4: Error scores for the Quality Estimation task (lower is better). Results are in terms of post-editing time per word. Bold scores are the best ones for each dataset.

We also inspect the resulting hyperparameters to obtain insights about the features used by the model. Table 5 shows the optimized λ values for the GP SSTK models with fixed α for the *fr-en* dataset. The λ values obtained are higher for the target sentence kernels than for the source sentence ones. We can interpret this as the model giving preference to features from the target trees instead of the source trees, which is what we would expect for this task.

5.3 Overfitting

Both NLP tasks show evidence that the GP models with large number of hyperparameters (SASSTK$_{\text{full}}$ in the case of Emotion Analysis and the free α models in Quality Estimation) are overfitting the corresponding datasets. While the Bayesian formulation for the marginal likelihood does help limiting overfitting, it does not prevent it completely. Small datasets or invalid assumptions about the Gaussian distribution of the data may still lead to poorly fitting models. Another means of reducing overfitting is by taking a fully Bayesian approach in which hyperparameters are considered as random variables and are marginalized out (Osborne, 2010); this is a research direction we plan to pursue in the future.[8]

	λ_{src}	λ_{tgt}
GP SSTK +	0.1394	0.3108
GP SSTK ×	0.1405	0.2641

Table 5: Learned hyperparameters for the GP SSTK models in the *fr-en* dataset, with α fixed at 1. λ_{src} and λ_{tgt} are the hyperparameters corresponding to the kernels on the source and target parse trees, respectively. The values shown are averaged over the cross-validation results.

5.4 Extensions to Other Tasks

The GP framework introduced in Section 2 can be extended to non-regression problems by changing the likelihood function. For instance, models for classification (Rasmussen and Williams, 2006, Chap. 3), ordinal regression (Chu and Ghahramani, 2005) and structured prediction (Altun et al., 2004; Bratières et al., 2013) were proposed in the literature. Since the likelihood is independent of the kernel, a natural future step is to apply the kernels and models introduced in this paper to different NLP tasks.

In light of that, we did initial experiments with constituency parsing reranking.[9] The first results were inconclusive but we do believe this is because we employed naive approaches using classification (1-best result vs. all) and regression (using PARSEVAL metrics as the response variable) models. A more appropriate way to tackle this task is by employing a reranking-based likelihood and this is a direction we plan to pursue in the future.

6 Related Work

Interest in model selection procedures for kernel-based methods has been growing in the last years.

[8] See also Rasmussen and Williams (2006, Chap. 5) for an in-depth discussion on this issue.

[9] We thank the anonymous reviewers for this suggestion.

One widely used approach for that is Multiple Kernel Learning (MKL) (Gönen and Alpaydın, 2011). MKL is based on the idea of using combinations of kernels to model the data and developing algorithms to tune the kernel coefficients. This is different from our method, where we focus on learning the hyperparameters of a single structural kernel. An approach similar to ours was proposed by Igel et al. (2007). They combine oligo kernels (a kind of n-gram kernel) with MKL, derive their gradients and optimize towards a kernel alignment metric. Compared to our approach, they restrict the length of the n-grams being considered, while we rely on dynamic programming to explore the whole substructure space. Also, their method does not take into account the underlying learning algorithm. Another recent approach proposed for model selection is random search (Bergstra and Bengio, 2012). Like grid search, it has the drawback of not employing gradient information, as it is designed for any kind of hyperparameters (including categorical ones).

Structural kernels have been successfully employed in a number of NLP tasks. The original SSTK proposed by Collins and Duffy (2001) was used to rerank the output of syntactic parsers. Recently, this reranking idea was also applied to discourse parsing (Joty and Moschitti, 2014). Other tree kernel applications include Semantic Role Labelling (Moschitti et al., 2008) and Relation Extraction (Plank and Moschitti, 2013). String kernels were mostly used in Text Classification (Lodhi et al., 2002; Cancedda et al., 2003), while graph kernels have been used for recognizing Textual Entailment (Zanzotto and Dell'Arciprete, 2009). However, these previous works focused on frequentist methods like SVM or voted perceptron while we employ a Bayesian approach.

Gaussian Processes are a major framework in machine learning nowadays: applications include Robotics (Ko et al., 2007), Geolocation (Schwaighofer et al., 2004) and Computer Vision (Sinz et al., 2004). Only very recently they have been successfully employed in a few NLP tasks such as translation quality estimation (Cohn and Specia, 2013; Beck et al., 2014b), detection of temporal patterns in text (Preoţiuc-Pietro and Cohn, 2013), semantic similarity (Rios and Specia, 2014) and emotion analysis (Beck et al., 2014a). In terms of feature representations, previous work focused on the vectorial inputs and applied well-known kernels for these inputs, e.g. the RBF kernel. As shown on §5.2, our approach is orthogonal to these previous ones, since kernels can be easily combined in different ways.

It is important to note that we are not the first ones to combine GPs with kernels on structured inputs. Driessens et al. (2006) employed a combination of GPs and graph kernels for reinforcement learning. However, unlike our approach, they did not attempt model selection, evaluating only a few hyperparameter values empirically.

7 Conclusions

This paper describes a Bayesian approach for structural kernel learning, based on Gaussian Processes for easy model selection. Experiments applying our models to synthetic data showed that it is possible to learn structural kernel hyperparameters using a fairly small amount of data. Furthermore we obtained promising results in two NLP tasks, including Quality Estimation, where we beat the state of the art. Finally, we showed how these rich parameterizations can lead to more interpretable kernels.

Beyond empirical improvements, an important goal of this paper is to present a method that enables new kernel developments through the extension of the number of hyperparameters. We focused on the Subset Tree Kernel, proposing an extension and then deriving its gradients. This approach can be applied to any structural kernel, as long as gradients are available. It is our hope that this work will serve as a starting point for future developments in these research directions.

Acknowledgements

Daniel Beck was supported by funding from CNPq/Brazil (No. 237999/2012-9). Dr. Cohn is the recipient of an Australian Research Council Future Fellowship (project number FT130101105). The authors would also like to thank the three anonymous reviewers for their helpful comments and suggestions.

References

Yasemin Altun, Thomas Hofmann, and Alexander J. Smola. 2004. Gaussian Process Classification for

Segmenting and Annotating Sequences. In *Proceedings of ICML*.

Nguyen Bach, Fei Huang, and Yaser Al-Onaizan. 2011. Goodness: A Method for Measuring Machine Translation Confidence. In *Proceedings of ACL*, pages 211–219.

Daniel Beck, Trevor Cohn, and Lucia Specia. 2014a. Joint Emotion Analysis via Multi-task Gaussian Processes. In *Proceedings of EMNLP*, pages 1798–1803.

Daniel Beck, Kashif Shah, and Lucia Specia. 2014b. SHEF-Lite 2.0 : Sparse Multi-task Gaussian Processes for Translation Quality Estimation. In *Proceedings of WMT14*, pages 307–312.

James Bergstra and Yoshua Bengio. 2012. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, 13:281–305.

John Blatz, Erin Fitzgerald, and George Foster. 2004. Confidence estimation for machine translation. In *Proceedings of the 20th Conference on Computational Linguistics*, pages 315–321.

Ondej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-amand, Radu Soricut, Lucia Specia, and Aleš Tamchyna. 2014. Findings of the 2014 Workshop on Statistical Machine Translation. In *Proceedings of WMT14*, pages 12–58.

Sébastien Bratières, Novi Quadrianto, and Zoubin Ghahramani. 2013. Bayesian Structured Prediction using Gaussian Processes. *arXiv:1307.3846*, pages 1–17.

Nicola Cancedda, Eric Gaussier, Cyril Goutte, and Jean-Michel Renders. 2003. Word-Sequence Kernels. *The Journal of Machine Learning Research*, 3:1059–1082.

Chih-Chung Chang and Chih-Jen Lin. 2001. LIBSVM : A Library for Support Vector Machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):1–39.

Wei Chu and Zoubin Ghahramani. 2005. Gaussian Processes for Ordinal Regression. *Journal of Machine Learning Research*, 6:1019–1041.

Trevor Cohn and Lucia Specia. 2013. Modelling Annotator Bias with Multi-task Gaussian Processes: An Application to Machine Translation Quality Estimation. In *Proceedings of ACL*, pages 32–42.

Michael Collins and Nigel Duffy. 2001. Convolution Kernels for Natural Language. In *Proceedings of NIPS*, pages 625–632.

Kurt Driessens, Jan Ramon, and Thomas Gärtner. 2006. Graph Kernels and Gaussian Processes for Relational Reinforcement Learning. *Machine Learning*, 64(1-3):91–119.

Mehmet Gönen and Ethem Alpaydın. 2011. Multiple kernel learning algorithms. *Journal of Machine Learning Research*, 12:2211–2268.

Christian Hardmeier, Joakim Nivre, and Jörg Tiedemann. 2012. Tree Kernels for Machine Translation Quality Estimation. In *Proceedings of WMT12*, pages 109–113.

Christian Hardmeier. 2011. Improving Machine Translation Quality Prediction with Syntactic Tree Kernels. In *Proceedings of EAMT*, pages 233–240.

David Haussler. 1999. Convolution Kernels on Discrete Structures. Technical report, University of California at Santa Cruz.

Yifan He, Yanjun Ma, Josef van Genabith, and Andy Way. 2010. Bridging SMT and TM with Translation Recommendation. In *Proceedings of ACL*, pages 622–630.

James Hensman, Nicolò Fusi, and Neil D. Lawrence. 2013. Gaussian Processes for Big Data. In *Proceedings of UAI*, pages 282–290.

Christian Igel, Tobias Glasmachers, Britta Mersch, and Nico Pfeifer. 2007. Gradient-based Optimization of Kernel-Target Alignment for Sequence Kernels Applied to Bacterial Gene Start Detection. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4(2):216–226.

Shafiq Joty and Alessandro Moschitti. 2014. Discriminative Reranking of Discourse Parses Using Tree Kernels. In *EMNLP*, pages 2049–2060.

Jonathan Ko, Daniel J. Klein, Dieter Fox, and Dirk Haehnel. 2007. Gaussian Processes and Reinforcement Learning for Identification and Control of an Autonomous Blimp. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 742–747.

Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. 2002. Text Classification using String Kernels. *The Journal of Machine Learning Research*, 2:419–444.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. In *Proceedings of ACL Demo Session*, pages 55–60.

Alessandro Moschitti, Daniele Pighin, and Roberto Basili. 2008. Tree Kernels for Semantic Role Labeling. *Computational Linguistics*, pages 1–32.

Alessandro Moschitti. 2006a. Efficient Convolution Kernels for Dependency and Constituent Syntactic Trees. In *Proceedings of ECML*, pages 318–329.

Alessandro Moschitti. 2006b. Making Tree Kernels practical for Natural Language Learning. In *EACL*, pages 113–120.

Michael Osborne. 2010. *Bayesian Gaussian Processes for Sequential Prediction, Optimisation and Quadrature*. Ph.D. thesis, University of Oxford.

Bo Pang and Lillian Lee. 2008. Opinion Mining and Sentiment Analysis. *Foundations and Trends in Information Retrieval*, 2(12):1–135.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Duborg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Daniele Pighin and Alessandro Moschitti. 2010. On Reverse Feature Engineering of Syntactic Tree Kernels. In *Proceedings of CONLL*, pages 223–233.

Barbara Plank and Alessandro Moschitti. 2013. Embedding Semantic Similarity in Tree Kernels for Domain Adaptation of Relation Extraction. In *Proceedings of ACL*, pages 1498–1507.

Daniel Preoţiuc-Pietro and Trevor Cohn. 2013. A temporal model of text periodicities using Gaussian Processes. In *Proceedings of EMNLP*, pages 977–988.

Carl Edward Rasmussen and Christopher K. I. Williams. 2006. *Gaussian processes for machine learning*, volume 1. MIT Press Cambridge.

Miguel Rios and Lucia Specia. 2014. UoW : Multi-task Learning Gaussian Process for Semantic Textual Similarity. In *Proceedings of SemEval*, pages 779–784.

Anton Schwaighofer, Marian Grigoras, Volker Tresp, and Clemens Hoffmann. 2004. GPPS: A Gaussian Process Positioning System for Cellular Networks. In *Proceedings of NIPS*, pages 579–586.

Fabian H. Sinz, Joaquin Quiñonero Candela, Gökhan H. Bakır, Carl E. Rasmussen, and Matthias O. Franz. 2004. Learning Depth from Stereo. *Pattern Recognition*, pages 1–8.

Lucia Specia, Nicola Cancedda, Marc Dymetman, Marco Turchi, and Nello Cristianini. 2009. Estimating the sentence-level quality of machine translation systems. In *Proceedings of EAMT*, pages 28–35.

Lucia Specia, Dhwaj Raj, and Marco Turchi. 2010. Machine translation evaluation versus quality estimation. *Machine Translation*, 24(1):39–50.

Lucia Specia, Kashif Shah, José G. C. De Souza, and Trevor Cohn. 2013. QuEst - A translation quality estimation framework. In *Proceedings of ACL Demo Session*, pages 79–84.

Lucia Specia. 2011. Exploiting Objective Annotations for Measuring Translation Post-editing Effort. In *Proceedings of EAMT*, pages 73–80.

Carlo Strapparava and Rada Mihalcea. 2007. SemEval-2007 Task 14 : Affective Text. In *Proceedings of SemEval*, pages 70–74.

Carlo Strapparava and Rada Mihalcea. 2008. Learning to identify emotions in text. In *Proceedings of the 2008 ACM Symposium on Applied Computing*, pages 1556–1560.

Fabio Massimo Zanzotto and Lorenzo Dell'Arciprete. 2009. Efficient kernels for sentence pair classification. In *Proceedings of EMNLP*, pages 91–100.

A Details on SVM Baselines

All SVM baselines employ the ϵ-insensitve loss function. Grid search optimization is done via 3-fold cross-validation on the respective training set and use RMSE as the metric to be minimized. After obtained the best hyperparameter values, the SVM is retrained using these values on the full respective training set. The specific intervals used in grid search depend on the task.

For the performance experiments on synthetic data, we employed an interval of $[10^{-2}, 10]$ for C (regularization coefficient) and ϵ, $[10^{-8}, 1]$ for λ and $[10^{-4}, 2]$ for α. In each run we incrementally increase the size of the grid by adding intermediate values on each interval. We keep a linear scale for the SSTK hyperparameters and a logarithmic scale for C and ϵ. As an example, Table 6 shows the resulting grids when the grid value is 4 for each hyperparameter. For all NLP experiments the grid is fixed for all hyperparameters (including γ, the lengthscale value in the RBF kernel), with its corresponding values shown on Table 7.

C / ϵ	$[10^{-2}, 10^{-1}, 1, 10]$
λ	$[10^{-8}, 0.33, 0.67, 1]$
α	$[10^{-4}, 0.67, 1.33, 2]$

Table 6: Resulting grids for the performance experiments when grid size is set to 4 for each hyperparameter.

C	$[10^{-2}, 1, 100]$
ϵ	$[10^{-2}, 10^{-1}, 1, 10]$
λ	$[10^{-16}, 0.25, 0.5]$
α	$[1]$ (fixed)
γ	$[10^{-3}, 0.0178, 0.316, 5.62, 100]$

Table 7: Grid values for the NLP experiments.