Model	**Templates**	**Accuracy**	**Speed**
Baseline	46	97.22	1x
Lasso	6	90.53	10.97x
Lasso	10	94.33	7.67x
Lasso	15	94.84	5.23x
Lasso	23	96.19	3.31x
GOMP	6	92.18	6.83x
GOMP	10	94.15	4.29x
GOMP	15	96.46	2.83x
GOMP	23	96.81	1.96x
Wrapper	6	96.45	6.13x
Wrapper	10	96.88	3.82x
Wrapper	15	97.01	2.62x
Wrapper	23	97.15	1.70x

Table 4: Comparison of the wrapper method for learning template orderings with group lasso and group orthogonal matching pursuit. Speeds are measured relative to the baseline, which is about 23,000 tokens/second on a 2.1GHz AMD Opteron machine.

Supplementary Material

A Experiments: Learning Template Ordering

As described in Section 4.5, we experimented with 3 different algorithms for learning an ordering of feature templates: Group lasso (**Lasso**), which prunes feature templates based on their norm after ℓ_1 regularization; Group orthogonal matching pursuit (**GOMP**), which selects features by iteratively training a model using the existing features then adds the template that maximizes Eqn. 4; and the wrapper method (**wrapper**). We use the methods of setting regularization parameters for Lasso and GOMP discussed in Section 4.5. Note that for the case of Group Lasso, this corresponds to the experimental setup used when evaluating Group Lasso for NLP in Martins et al. (2011).

We compare the different methods for picking template orderings in Table 4, training and testing models for WSJ POS tagging. To keep concerns separate, we use standard sequential prediction using all templates rather than our dynamic prediction method. We found the wrapper method to be the most successful towards our goal of achieving high accuracy while using as few templates as possible, which is important when using dynamic prediction. While the wrapper method comes within 0.15% of state-of-the-art accuracy using the first 23 out of 46 total templates, neither GOMP nor Lasso are able to exceed 97% accuracy with so few templates.

In terms of speed, Lasso outperforms both GOMP and the wrapper method, predicting 3 times as fast with 23 templates as the baseline, whereas GOMP predicts twice as fast, and the wrapper method 1.7 times baseline speed. It is initially puzzling that for a given number of templates, each model does not achieve the same speed increase. Since each template has only a single active feature for a given test instance, we are computing the same number of multiplications and additions for each model. However, the different models are selecting very different feature templates, whose features can take a widely different amount of time to compute. For example, creating and hashing the string representing a trigram conjunction into a sparse vector takes much longer than creating a feature whose template has only two possible values, *true* and *false*.

This behavior is a reflection of some interesting properties of the template selection methods, which help to explain the superior performance of the wrapper method.

Since the Group Lasso penalizes the norms of the templates as a surrogate for the sparsity (a 0-1 loss), it is naturally biased against large templates and will always include very small templates – the early stages of the regularization path will contain these small templates whose features can be quickly computed. Another likely source of speedup arises from the impact on cache locality of using much smaller cardinality weight vectors. This also explains the poor performance of the induced template ordering. Including small templates early on runs at cross-purposes to our goal of placing highly predictive templates up front for our dynamic prediction algorithm.

In contrast, the wrapper method and GOMP both pick larger, more predictive templates early on since they attempt to minimize loss functions that are unrelated to template size. While GOMP produces better orderings than lasso, in this case the wrapper method works better because the template ordering produced by GOMP is unable to incorporate signal from a validation set and overfits the training set.

We find that the wrapper method works well with our dynamic training objective, which benefits from predictive, though expensive, features early on in the ordering. When using dynamic prediction, the speed per template used is offset by using significantly fewer templates on examples that are easier to classify.

B Sparse Regularized Group Orthogonal Matching Pursuit

Group Orthogonal Matching Pursuit (GOMP) picks a stagewise ordering of feature templates to add to a generalized linear model. At each stage, GOMP effectively uses each feature template to perform a linear regression to fit the gradient of the loss function. This attempts to find the correlation of each feature subset with the residual of the model. It then adds the feature template that best fits this gradient, and retrains the model. We adapt this algorithm to the setting of high-dimensional NLP problems by efficiently inverting the covariance matrices of the feature templates, and regularizing the computation of the residual correlation. This results in a scalable feature selection technique for our problem setting, detailed below.

For purposes of exposition, we will break from the notation of Section 3 that combines features of x and y since the algorithm is designed from a linear-algebraic, regression standpoint that considers the design matrix X and the label matrix Y as separate entities. We will call each group of features G_i, its associated design matrix X_i (the feature matrix for that template).

At each step k, we use our selected set of feature templates/groups and compute the gradient of our loss function on the training data set, call it $r^{(k)}$, and then we select a new feature template G_i with corresponding design matrix X_i to add to our selected groups, by finding the index that maximizes:

$$\arg\max_i \mathbf{tr}\big((r^{(k)})^\top X_i (X_i^\top X_i)^{-1} X_i^\top r^{(k)}\big) \tag{3}$$

After adding this template, we retrain the model on the selected set of templates and repeat. Note that this appears to be a difficult optimization problem to solve: some of our templates have hundreds of thousands or even millions of features and computing the inverse $(X_i^\top X_i)^{-1}$ could be expensive – however, due to the special structure of our NLP problems, where each feature template contains one-hot features, this covariance matrix is diagonal and hence trivially invertible.

However, we find in practice that due to the large number of features and relatively small number of examples in our NLP models, this picks very-high cardinality feature templates early on that generalize poorly. The reason becomes apparent when we notice that the correlation-finding subroutine, Equation (3), is essentially an un-regularized least squares problem, attempting to regress the loss function gradient onto the data matrices for each template. This suggests we should try a form of regularization, by using some template-dependent constant α_i to regularize the inversion of the covariance matrix:

$$\arg\max_i \mathbf{tr}\big((r^{(k)})^\top X_i (X_i^\top X_i + \lambda_i I)^{-1} X_i^\top r^{(k)}\big) \tag{4}$$

Heuristically, we pick this regularization parameter to be a low fractional power of the dimension size for each feature template $\lambda_i = \mathbf{col}(X_i)^{\alpha_i}$, where α_i is picked to be in the regime of $[0.25, 0.5]$.