

Kernelized Sorting for Natural Language Processing

Jagadeesh Jagarlamudi and Seth Juarez and Hal Daumé III

School of Computing, University of Utah, USA
{jags, seth, hal}@cs.utah.edu

Abstract

Kernelized sorting is an approach for matching objects from two sources (or domains) that does not require any prior notion of similarity between objects across the two sources. Unfortunately, this technique is highly sensitive to initialization and high dimensional data. We present variants of kernelized sorting to increase its robustness and performance on several Natural Language Processing (NLP) tasks: document matching from parallel and comparable corpora, machine transliteration and even image processing. Empirically we show that, on these tasks, a semi-supervised variant of kernelized sorting outperforms matching canonical correlation analysis.

Introduction

Object matching, or alignment, is an underlying problem for many natural language processing tasks, including document alignment (Vu, Aw, and Zhang 2009), sentence alignment (Gale and Church 1991; Rapp 1999) and transliteration mining (Hermjakob, Knight, and Daumé III 2008; Udupa et al. 2009). For example, in document alignment, we have English documents (objects) and French documents (objects) and our goal is to discover a matching between them. Standard approaches require users to define a measure of similarity between objects on one side and objects on the other (“How similar is this English document to that French document?”). Doing so typically requires external resources, such as bilingual dictionaries, which are not available for most language pairs. Kernelized sorting (Quadrianto, Song, and Smola 2009) provides an alternative: one only has to define *monolingual* notions of similarity to perform a matching. Unfortunately, a direct application of kernelized sorting lacks robustness on NLP problems due to high dimensionality and noisy data. We describe several modifications to kernelized sorting that renders it applicable to several NLP applications, attaining significant gains over a naive application.

Kernelized sorting exploits inter-domain similarities between objects (eg., English-to-English and French-to-French) to discover an intra-domain matching (eg., English-to-French). It works under the assumption that if object A is similar to objects B and C in English, then the French coun-

terpart of A should be equally similar to the French counterparts of B and C . As suggested by the name, it involves computation of two kernel matrices, one for each language. It then tries to find a permutation of one of these matrices to maximize an appropriately-defined measure of similarity (discussed in the next section).

However, several problems plague a direct application of kernelized sorting to NLP problems. First, because NLP applications have high dimensionality, the kernel matrices typically end up being highly noisy and diagonally dominant. Moreover, the objective function maximized in kernelized sorting is convex, subject to linear constraints, rendering solutions highly sensitive to initialization¹. To address these issues, we explore sub-polynomial smoothing as a method for avoiding diagonal dominance. We further develop a semi-supervised “bootstrapping” variant of kernelized sorting that addresses the problem of noise. We compare kernelized sorting with matching canonical correlation analysis (MCCA) (Haghighi et al. 2008) on a wide variety of tasks and data sets and show that these strategies are sufficient to turn kernelized sorting from an approach with highly unpredictable performance into a viable approach for NLP problems.

Kernelized Sorting

Given two sets of observations from different domains $X \in \mathcal{X}$ and $Y \in \mathcal{Y}$ with an unknown underlying alignment, kernelized sorting aims to recover the alignment *without* using any cross-domain clues. Using only the similarities between objects in the same domain, it maximizes Hilbert Schmidt Independence Criterion (HSIC) (Gretton et al. 2005) to find the alignment.

Assuming the observations X and Y are drawn from a joint probability distribution \Pr_{xy} , HSIC can be used to verify if the joint probability distribution factorizes into $\Pr_x \times \Pr_y$. At a broader level, it measures the distance between the probability distributions \Pr_{xy} and $\Pr_x \times \Pr_y$ (Smola et al. 2007). This distance is zero if and only if X and Y are independent. Moreover, given the kernel matrices K and L (on X and Y respectively) and a projection matrix $H = I - 1/n$, this distance can be empirically estimated using $n^{-2}\text{tr}(HKL)$ where n is the total number of

¹It is easy to *minimize* convex functions, but hard to *maximize*.

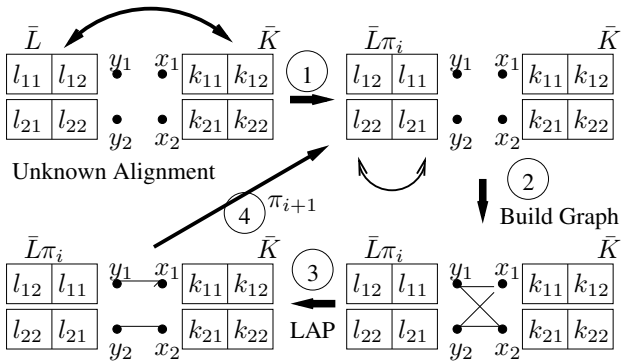


Figure 1: The process of kernelized sorting to match observations in Y & X

observations in each domain.

Since kernelized sorting aims to find a best correspondence between the observations, it maximizes the dependency (as measured by HSIC) of the kernel matrices. By representing an alignment in terms of a permutation matrix $\pi \in \{0, 1\}^{n \times n}$, where $\pi_{ij} = 1$ indicates the alignment between observations X_j and Y_i , finding the best alignment is formulated as an optimization problem:

$$\pi^* = \arg \max_{\pi \in \Pi_n} \text{tr}(\bar{K} \pi^T \bar{L} \pi) \quad (1)$$

$$\text{s.t. } \pi 1_n = 1_n \ \& \ \pi^T 1_n = 1_n \quad (2)$$

where $\bar{K} := HKH$ and $\bar{L} := HLH$ are data centered versions of K and L respectively, Π_n is the set of all permutation matrices of size $n \times n$ and 1_n is a column vector of ones of size n . The constraints (Eqn. 2) together with $\pi_{ij} \in \{0, 1\}$ indicate that an observation in one domain is allowed to be aligned to only one observation in another domain². Intuitively, in the operation $\pi^T \bar{L} \pi$, applying π from the right interchanges the columns of matrix \bar{L} so that the i^{th} dimension of resulting Y vectors is aligned to the i^{th} dimension of X vectors. For example, the effect of applying $\pi = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ on \bar{L} interchanges the first and second column (shown in the step 1 of Fig. 1). Similarly applying from the left interchanges the rows so that the new i^{th} observation corresponds to observation X_i . The optimization problem (Eqn. 1) is solved iteratively, i.e. given a permutation matrix π_i at i^{th} iteration, we fix π_i and solve for the permutation matrix π_{i+1} given by $\arg \max_{\pi} \text{tr}(\bar{K} \pi^T \bar{L} \pi_i)$. This optimization problem is converted into a Linear Assignment Problem (LAP) (Jonker and Volgenant 1987) and solved for optimal π_{i+1} .

The entire process is depicted in Figure 1. In the beginning there are no alignments between the sets of observations and hence the algorithm picks some initial permutation π_0 . At any iteration, the columns of \bar{L} are permuted according to the current permutation π_i so that the dimensions of both the sets of observations match (step 1). As discussed previously this aligns the dimensions of observation

²We will relax this constraint in the following sections.

vectors, i.e. we have aligned the relationships with other objects, and now we will try to find a best match between observations according to current π_i . This is done by first building a weighted bipartite graph between X and Y , with weights equal to $\langle X_i, Y_j \rangle$ (step 2)³. And then (step 3), a LAP solver finds an optimal assignment of observations in set Y to observations in set X which becomes our new permutation matrix π_{i+1} for step 4. This process continues until the convergence in the objective function value.

Problems in Adapting to NLP

Since kernelized sorting involves a *maximization* of a convex function, $\text{tr}(\bar{K} \pi^T \bar{L} \pi)$, it is sensitive to the initialization of π . Quadrianto, Song and Smola (2009) suggest using the projection on first principal components (referred as principal component based initialization) as the initial permutation matrix. This initialization increases data dependency. They report that kernelized sorting is able to recover 140 pairs of images out of 320 (data set is described in ‘‘Experimental Setup’’ Section). In our preliminary experiments (using their code), we varied the number of image pairs and plotted the number of recovered alignments (Fig. 2). It can be clearly seen that its performance is very sensitive to the data and is unpredictable. Adding an additional 10 pairs of images to a data set with 190 pairs decreases the number of recovered alignments from 78 to zero. Kernel matrices computed over textual data tend to be very noisy and thus pose a further threat to the robustness of its direct application.

Apart from the issue of robustness, the diagonal dominance of kernel matrices used in NLP applications also creates a major problem. Since kernelized sorting relies on maximizing an empirical estimate of HSIC, the bias involved in this estimate becomes important when kernel matrices are diagonally dominant. To understand this, consider the objective function being maximized in the iterative step $\text{tr}(\bar{K} \hat{L})$, where $\hat{L} = \bar{L} \pi_i$. This objective function can be rewritten as:

$$\sum_{ij} \bar{K}_{ij} \hat{L}_{ij} = \sum_{i,j \text{ s.t. } i \neq j} \bar{K}_{ij} \hat{L}_{ij} + \sum_i \bar{K}_{ii} \hat{L}_{ii}$$

If the kernel matrices are diagonal dominant, then the similarity of an object with itself dominates the similarity with the rest of the objects in the same domain. Thus while finding an optimal π_{i+1} , we end up aligning this object to an equally self-similar object in the other domain. As a result the similarity of an observation with the rest of the observations from the same domain becomes moot. NLP applications are known for their huge dimensionality and near-orthogonal feature vectors and hence this issue becomes crucial.

In the next couple of sections we suggest two simple and practical improvements to kernelized sorting to handle both diagonal dominance and noisy kernels.

Sub-polynomial Kernel

Diagonal dominance also creates problem in other scenarios like Support Vector Machine learning and document Clus-

³In this particular case Y_j is the permuted vector in step 1.

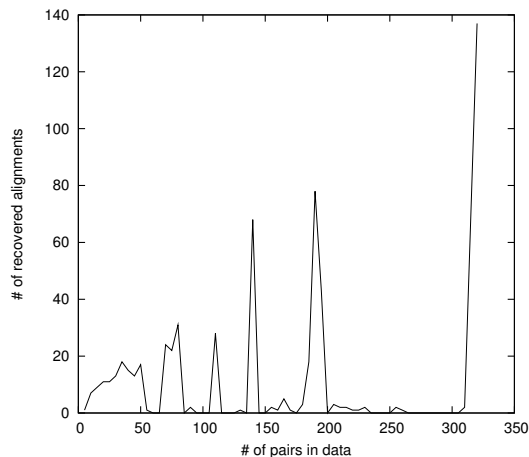


Figure 2: Number of recovered alignments on image data set using naive KS technique.

tering (Greene and Cunningham 2006). Sub-polynomial kernels (Schölkopf et al. 2002) have been proposed to address the issue of large diagonal elements in the SVM literature. Given a kernel function corresponding to a feature map ϕ and a scalar p s.t. $0 < p < 1$, the sub-polynomial kernel function is defined as:

$$k_{SP}(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle^p$$

Let K be the kernel matrix corresponding to the unmodified kernel function, then the sub-polynomial kernel matrix K_{SP} can be easily computed by raising each element to the power p . Since $p < 1$, elements that are less than one will increase while elements that are greater than one will decrease. This operation decreases the range in which the modified elements lie and as a result the ratio of diagonal elements to the off-diagonal elements decrease.

Unfortunately K_{SP} may no longer be a positive definite matrix. To convert into a valid kernel matrix, first the rows are normalized to unit length and then a gram matrix is computed as $K_{SP} \leftarrow K_{SP} \times K_{SP}^T$. If we consider the observations in set X , this transformation has the effect of first representing each observation in terms of its similarity with all observations and then a new similarity is defined between two objects based on the closeness of their similarity vectors.

In general, the problem with using sub-polynomial kernel is the need to choose the value of the parameter p . This however turns out to be an advantage of using sub-polynomial kernels for the problem at hand. Recall that kernelized sorting is sensitive to initialization, each value of p give us two new kernel matrices and each such pair will result in a new initialization matrix (the projection onto first principal components). Since each of these principal component based initializations is likely to find a better solution than a random initialization, the process of searching for the value of p gives us a systematic way to explore different possible candidate initializations. From the resulting solutions we take the permutation matrix that achieves highest objective function value.

$ \begin{array}{cc} Y_1 & \bullet & X_1 \\ & \diagdown & \diagup \\ Y_2 & \bullet & X_2 \\ & \diagup & \diagdown \\ Y_3 & \bullet & X_3 \\ & & \\ Y_4 & \bullet & X_4 \end{array} $	$ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0.5 \\ 0 & 0 & 0.5 & 0.5 \end{bmatrix} $	$ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} $
	Uniform wts.	Zero wts.

Table 1: Permutation matrices corresponding to the relaxed strategies in the presence of seed alignments.

Seed Alignments

As discussed in earlier sections, the performance of kernelized sorting significantly drops when the kernel matrices are noisy. In this section we propose another improvement to kernelized sorting which helps in recovering alignment in such situations.

Since kernelized sorting tries to find a globally consistent alignments, even low confidence, and quite possibly incorrect alignments, will have equal effect as any confident (probably correct) alignments in determining the next permutation matrix. The key idea is to first rank the alignments at any iteration and give preference to top ranked alignments than the low confident ones. As a result, the incorrect alignments will have minimal effect in determining the future permutation matrix and hopefully the alignments become better and better with each iteration. In step 3 of the algorithm (Fig. 1), the new permutation matrix is determined by solving a linear assignment problem. The output of this stage is a set of weighted correspondences between observations in different domains. We can use this weight to identify the (probably) correct alignments from incorrect ones. But, since the algorithm runs in an unsupervised setting an incorrect permutation matrix at initial iterations can cause bad correspondences to receive high weight. Hence this weight may not be a good indicator of accuracy of an alignment. To overcome this, we provide an initial set of seed alignments to the algorithm and force the algorithm to respect these alignments in every iteration.

Given an initial set of seed alignments⁴, the algorithm starts with a permutation matrix π_0 s.t. $\pi_{0_{ij}} = 1$ if observation Y_i is aligned to X_j in the initial seed. We relaxed the constraints on the permutation matrix (Eqn. 2) and explored two strategies for the remaining observations: uniform weights (w_u) and zero weights (w_z). Table 1 shows an example seed alignments and the corresponding permutation matrix for both the strategies⁵. After the columns of \bar{L} are permuted (step 1 of Fig 1), a weighted bipartite graph is constructed. In this particular example, the edge weights are computed as $\langle X_j, (Y_i^T \pi)^T \rangle$:

$$\begin{aligned}
 w_u(Y_i, X_j) &= Y_{i2}X_{j1} + Y_{i1}X_{j2} \\
 &\quad + 0.5 * (Y_{i3} + Y_{i4}) * (X_{j3} + X_{j4}) \\
 w_z(Y_i, X_j) &= Y_{i2}X_{j1} + Y_{i1}X_{j2}
 \end{aligned}$$

Notice that in both these cases the alignments of Y_3 and Y_4 do not have any effect on the edge weights and hence the

⁴used only 10 seed alignments in our experiments.

⁵Recall that $\pi_{ij} = 1$ if observations Y_i and X_j are aligned.

assignments of the LAP solver will not be effected by the alignments of these observations. However with uniform weighting, even though the alignments of Y_3 and Y_4 do not matter, the similarities of Y_3 and Y_4 with other objects from the same domain (captured in terms of Y_{i3} and Y_{i4} respectively) will still effect the edge weights. Where as they won't have any effect on edge weights in zero weighting case.

Since applying LAP solver on this weighted graph may produce an inconsistent assignment for observations Y_1 and Y_2 , with respect to seed alignments, we perturb the edge weights by assigning high weight to an edge in the seed alignments and thus the LAP solver is forced to respect seed alignments. The new permutation matrix is formed by considering the most highly weighted $k_i + \delta$ assignments of the LAP solver, where k_i is the number of confident alignments at current iteration and δ (set to 2 in our experiments) is the number of new alignments added at each iteration. Thus at any iteration, we can reduce the effect of low confidence alignments in future permutation matrix while being consistent with the initial seed alignments. This process is repeated until the desired criterion is met.

Experimental setup

Since our main suggestions are ways to deal with diagonal dominance and noisy kernels, we focus mostly on evaluating on NLP tasks which are known for both these problems due to high dimensionality. We compare our system with MCCA (Haghighi et al. 2008) on the task of aligning documents written in multiple languages and the task of finding transliteration equivalents between names in different languages. To evaluate the robustness of our suggestions we also evaluate on the image alignment task as introduced by (Quadrianto, Song, and Smola 2009).

We use the following three data sets for the task of aligning documents:

- **EP-P**: 250 pairs of parallel documents extracted from EuroParl English and Spanish corpus (Koehn 2005).
- **EP-C**: 226 pairs of comparable documents artificially constructed out of EuroParl corpus by considering only the first half of English document and second half of the aligned Spanish document.
- **WP**: 115 pairs of aligned comparable articles from Wikipedia.

For all the above data sets, documents are first preprocessed to remove stop words and words are stemmed using snowball stemmer <http://snowball.tartarus.org/>. The preprocessed documents are converted into TFIDF vectors and then a linear dot product kernel is used to build the required kernel matrices.

We use the following data sets for the remaining two tasks:

- **TL**: 300 pairs of multi-word name transliterations between English and Arabic for the task of finding transliteration equivalents, derived from (Hermjakob, Knight, and Daumé III 2008).
- **IMG**: 320 pairs of images of size 20×40 pixels.

The image data set is same as the one used in (Quadrianto, Song, and Smola 2009) and is preprocessed in the same manner. For the transliteration data set, the Arabic names are romanized first and then each name in both the languages is converted into a vector by considering all possible unigram, bigram and trigram character sequences. Again a simple linear dot product kernel is used to build the kernel matrices in both English and Arabic.

We compare our system with a direct application of kernelized sorting (KS) technique and MCCA (Haghighi et al. 2008). For KS, we use the same experimental setup used by Quadrianto, Song, and Smola (2009): zero out the diagonal elements, use principal component based initialization and a convex combination of previous permutation matrix and the current solution of LAP solver. MCCA is a semi-supervised technique that uses a set of seed alignments. In each iteration it uses Canonical Correlation Analysis (CCA) (Hotelling 1936) to compute the directions along which already aligned observations are maximally correlated. Then it finds the alignment between new observations by mapping them into the sub-space found by CCA and then selectively updates the seed list. We use same set of seed alignments for both MCCA and our approach.

Results

We evaluate our suggestions incrementally:

- **p-smooth**: only sub-polynomial kernels with the principal component based initialization.
- **S-U**: p -smooth and ten random seed alignments with *uniform* weights for low confident alignments.
- **S-Z**: p -smooth and ten random seed alignments with *zero* weights for low confident alignments.

When using sub-polynomial kernel, we try all values of p from 0.01 to 1 with an increment of 0.01. At each value of p , we consider both the principal component based initialization and the solution corresponding to the previous value of p and keep the permutation that gives the maximum objective function value. In general we found that the best value of p lies in the range of 0.05 and 0.6. Table 2 reports the number of recovered alignments on all the tasks.

Aligning Images

First we report the results on the task of aligning images. The baseline system (KS) recovered 137 pairs of alignments while p -smooth also retrieved 136 pairs (fourth column of table 2). Though there is not much of difference between the number of recovered alignments with all the 320 pairs of images, p -smooth is more robust to changes in the data set as shown in Fig. 3. We start with five pairs of images and add five pairs in each step and let the systems determine the alignment. From the figure it is clear that, the performance of the baseline system crucially depends on the data set, while the dependence is not vital when the kernels are smoothed using sub-polynomial kernels.

Aligning Documents and Transliterations

The results on both document alignment task and the task of matching transliterations are also reported in Table 2.

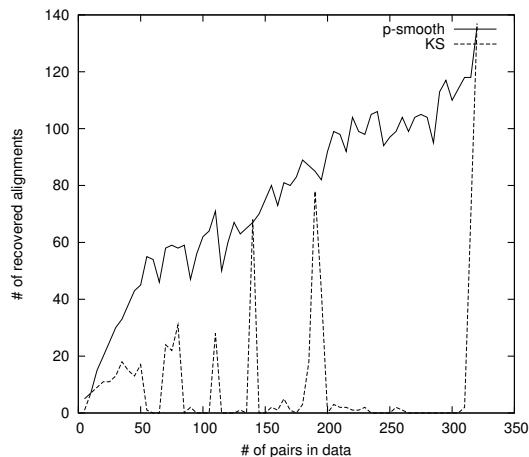


Figure 3: Number of recovered alignments on image data set. Both the naive KS and p -smooth are shown.

	KS	MCCA	p -smooth	S-U	S-Z
IMG (320)	137	35	136	144	144
EP-P (250)	6	250	250	250	250
EP-C (226)	2	34	41	88	88
WP (115)	5	48	11	76	81
TL (300)	7	209	30	250	262

Table 2: Number of Aligned pairs on all data sets. p -smooth and KS do not require seed alignments

Though KS performed competitively in aligning images, it recovers almost none of the alignments in case documents and transliterations⁶.

On the task of aligning parallel documents (EP-P), though both MCCA and our runs are able to recover all the alignments, our approach didn't require any seed alignments (p -smooth). This is because when the data is parallel, both source and target kernel matrices look almost identical. That is, the similarity vector of an observation Y_i (with other observations from \mathcal{Y}) is nearly the same as the similarity vector of its aligned observation X_{a_i} (with observations from \mathcal{X}). In this case, removing the diagonal dominance is enough to recover all alignments.

As the data becomes less parallel, the kernel matrices start becoming noisy and hence p -smooth by itself is not able to retrieve many document pairs. In this case, providing a small set of ten random seed alignments turned out to be very helpful in improving the accuracies, especially for WP and TL datasets. Though both the weighting strategies performed competitively, assigning zero weights, thus ignoring the low confident alignments completely, is marginally better in TL and WP data sets. Both our runs (S-U and S-Z) outperformed MCCA especially when the kernels are noisy, and we believe it is because while searching for different

⁶For baseline system, we use the same code available from the http://users.rsise.anu.edu.au/~nquadrianto/kernelized_sorting.html

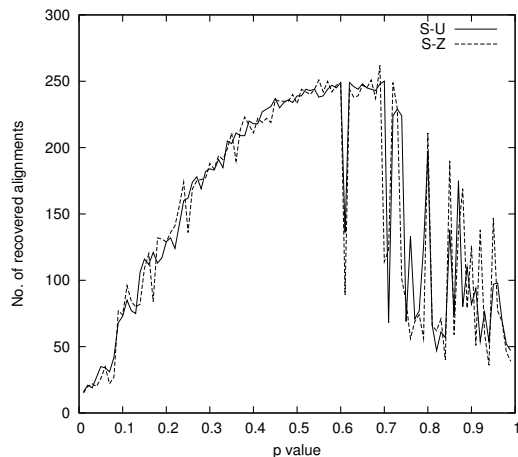


Figure 4: Number of recovered alignments (y-axis) is plotted against the value of p (x-axis) on TL data set.

values of p we perturb the data slightly and subsequently the objective function being maximized helps us in identifying the correct perturbation and hence a better alignment.

We also study the effect of the parameter p which controls the smoothing of kernel matrices. In Fig. 4 we plot the number of recovered alignments on the task of finding named entity transliterations (TL dataset) with the value of p . For both S-U and S-Z runs, the seed alignments constitute the initial permutation matrix. In these runs, accuracies increase till certain value of p and then start decreasing. This is very intuitive because, for very small values of p the ratio of highest element to the lowest element in the matrix becomes almost one and hence it doesn't distinguish the vectors. As the value of p becomes high, close to one, the performance again decreases because of the diagonal dominance. The curves for other data sets are similar and in general, for the datasets we experimented, the best value of p was found in the range from 0.05 to 0.6.

Discussion and Future Work

At first glance, it may appear that the assumptions of kernelized sorting: requirement of equal number of observations from both domains and the one-to-one alignment for every observation are too restrictive for its application to NLP. For, these assumptions are often not true in the case of NLP applications due to the inherent ambiguity of natural language. In this section, we argue that the technique can be modified to handle these cases. First, we empirically show that having noisy observations, i.e. observations without an aligned object in other domain, doesn't severely effect the performance of the technique. And then we suggest modifications to handle situations where the other assumptions could potentially break.

To study the effect of noise, we have added some random examples (with out any aligned object in other domain) in each domain to EP-C and TL data sets. Table 3 shows the number of alignments recovered by both MCCA and S-Z strategies. The presence of noisy examples equally ef-

	#Aligned	#Noisy	MCCA	S-Z
EP-C	200	26	31 (34)	69 (88)
TL	300	50	185 (209)	236 (262)

Table 3: Second column indicates the number of aligned pairs and the third column indicates the number of noisy examples added. New noisy pairs were added for TL data set while existing pairs were made noisy in EP-C data set.

ected both the algorithms (compared to the number of recovered alignments with out noisy pairs, shown in parenthesis). Though we didn’t experiment this, another possible technique to handle the noisy examples could be to stop the algorithm at intermediate iterations instead running it until it finds an alignment for every observation. The extension of this algorithm to situation with different number of observations on each side is also straight forward. Instead of π being a square matrix of size n , it will now become a rectangular matrix of appropriate size and the input to the LAP solver (step 3 of Fig. 1) is modified by adding dummy nodes on the side with fewer observations and assigning a very low weight to edges connecting these nodes so as to discourage aligning with these dummy nodes. Similarly, by relaxing the permutation matrix (like it was done for the uniform weighting case, S-U run) and by considering the weights output by the LAP solver, we believe it is possible to extend it to allow an observation to align with multiple observations from the other domain.

Now we compare the computational complexity of our approach with the MCCA technique. MCCA requires the actual representation of the observations while kernelized sorting needs only kernel matrices. Since the number of observations is usually much smaller than the number of dimensions, the memory requirement is less compared to that of MCCA. Moreover, MCCA involves computation of CCA projections in each iteration which is not required for our technique so this saves computational time per iteration. On the other hand, kernelized sorting involves searching for different values of p which is not required for MCCA technique. By restricting the search to a small region this can be efficiently implemented. In future, we would like to explore the possibility to learn the value of p using the seed sample.

In datasets with high number of observations, it is probably safer to assume that an observation will have significant similarity with only few observations. This indicates that, for each observation only a small number of observations decide its alignment and considerable fraction of the entire observations can be discarded. In future, we would like to exploit this local neighbourhood property to make this approach scalable to bigger data sets.

Conclusions

Due to the high dimensionality of NLP data, kernel matrices computed with out careful feature design typically end up being diagonally dominant and noisy. In this paper we have presented simple yet effective strategies to address both these issues for application kernelized sorting to NLP

tasks. Empirical results on various tasks and data sets show not only the improved robustness but also better accuracies compared to MCCA (a semi-supervised technique based on CCA).

Acknowledgements

We thank the anonymous reviewers for their helpful comments. This work was partially supported by NSF grant IIS-0916372.

References

- Gale, W. A., and Church, K. W. 1991. A program for aligning sentences in bilingual corpora. In *Proceedings ACL-91*, 177–184.
- Greene, D., and Cunningham, P. 2006. Practical solutions to the problem of diagonal dominance in kernel document clustering. In *ICML ’06*, 377–384. New York, NY, USA: ACM.
- Gretton, A.; ; Bousquet, O.; Smola, E.; and Schölkopf, B. 2005. Measuring statistical dependence with hilbert-schmidt norms. In *Proceedings Algorithmic Learning Theory*, 63–77. Springer-Verlag.
- Haghighi, A.; Liang, P.; Kirkpatrick, T. B.; and Klein, D. 2008. Learning bilingual lexicons from monolingual corpora. In *Proceedings of ACL-08: HLT*, 771–779.
- Hermjakob, U.; Knight, K.; and Daumé III, H. 2008. Name translation in statistical machine translation - learning when to transliterate. In *Proceedings of ACL-08: HLT*, 389–397.
- Hotelling, H. 1936. Relation between two sets of variables. *Biometrika* 28:322–377.
- Jonker, R., and Volgenant, A. 1987. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing* 38(4):325–340.
- Koehn, P. 2005. Europarl: A parallel corpus for statistical machine translation. In *MT Summit*.
- Quadrianto, N.; Song, L.; and Smola, A. J. 2009. Kernelized sorting. In Koller, D.; Schuurmans, D.; Bengio, Y.; and Bottou, L., eds., *Advances in NIPS-09*, 1289–1296.
- Rapp, R. 1999. Automatic identification of word translations from unrelated english and german corpora. In *Proceedings of ACL-99*, 519–526.
- Schölkopf, B.; Weston, J.; Eskin, E.; Leslie, C.; and Noble, W. S. 2002. A kernel approach for learning from almost orthogonal patterns. In *ECML ’02*, 511–528. London, UK: Springer-Verlag.
- Smola, A.; Gretton, A.; Song, L.; and Schölkopf, B. 2007. A hilbert space embedding for distributions. In *ALT ’07: Proceedings of Algorithmic Learning Theory*, 13–31. Berlin, Heidelberg: Springer-Verlag.
- Udupa, R.; Saravanan, K.; Kumaran, A.; and Jagarlamudi, J. 2009. Mint: A method for effective and scalable mining of named entity transliterations from large comparable corpora. In *EACL*, 799–807.
- Vu, T.; Aw, A.; and Zhang, M. 2009. Feature-based method for document alignment in comparable news corpora. In *EACL*, 843–851.