Carnegie Mellon University
Research Showcase @ CMU

Language Technologies Institute | School of Computer Science

2014

Locally Non-Linear Learning for Statistical Machine Translation via Discretization and Structured Regularization

Jonathan H. Clark
Carnegie Mellon University

Chris Dyer
Carnegie Mellon University, cdyer@cs.cmu.edu

Alon Lavie
Carnegie Mellon University, alon@cmu.edu

Follow this and additional works at: http://repository.cmu.edu/lti

 Part of the Computer Sciences Commons

Published In

Transactions of the Association for Computational Linguistics, 2, 393-404.

This Article is brought to you for free and open access by the School of Computer Science at Research Showcase @ CMU. It has been accepted for inclusion in Language Technologies Institute by an authorized administrator of Research Showcase @ CMU. For more information, please contact research-showcase@andrew.cmu.edu.

Locally Non-Linear Learning for Statistical Machine Translation via Discretization and Structured Regularization

Jonathan H. Clark* **Chris Dyer†** **Alon Lavie†**

*Microsoft Research
Redmond, WA 98052, USA
jonathan.clark@microsoft.com

†Carnegie Mellon University
Pittsburgh, PA 15213, USA
{cdyer,alavie}@cs.cmu.edu

Abstract

Linear models, which support efficient learning and inference, are the workhorses of statistical machine translation; however, linear decision rules are less attractive from a modeling perspective. In this work, we introduce a technique for learning arbitrary, rule-local, non-linear feature transforms that improve model expressivity, but do not sacrifice the efficient inference and learning associated with linear models. To demonstrate the value of our technique, we discard the customary log transform of lexical probabilities and drop the phrasal translation probability in favor of raw counts. We observe that our algorithm learns a variation of a log transform that leads to better translation quality compared to the explicit log transform. We conclude that non-linear responses play an important role in SMT, an observation that we hope will inform the efforts of feature engineers.

1 Introduction

Linear models using `log`-transformed probabilities as features have emerged as the dominant model in MT systems. This practice can be traced back to the IBM noisy channel models (Brown et al., 1993), which decompose decoding into the product of a translation model (TM) and a language model (LM), motivated by Bayes' Rule. When Och and Ney (2002) introduced a `log`-linear model for translation (a linear sum of `log`-space features), they noted that the noisy channel model was a special case of their model using `log` probabilities. This same formulation persisted even after the introduction of MERT (Och, 2003), which optimizes a linear model; again, using two `log` probability features (TM and LM) with equal weight recovered the noisy channel model. Yet systems now use many more features, some of which are not even probabilities. We no longer believe that equal weights between the TM and LM provides optimal translation quality; the probabilities in the TM do not obey the chain rule nor Bayes' rule, nullifying several theoretical mathematical justifications for multiplying probabilities. The story of multiplying probabilities may just amount to heavily penalizing small values.

The community has abandoned the original motivations for a linear interpolation of two `log`-transformed features. Is there empirical evidence that we should continue using this particular transformation? Do we have any reason to believe it is better than other non-linear transformations? To answer these, we explore the issue of non-linearity in models for MT. In the process, we will discuss the impact of linearity on feature engineering and develop a general mechanism for learning a class of non-linear transformations of real-valued features.

Applying a non-linear transformation such as `log` to features is one way of achieving a non-linear response function, even if those features are aggregated in a linear model. Alternatively, we could achieve a non-linear response using a natively non-linear model such as a SVM (Wang et al., 2007) or RankBoost (Sokolov et al., 2012). However, MT is a structured prediction problem, in which a full hypothesis is composed of partial hypotheses. MT decoders take advantage of the fact that the model

*This work was conducted as part of the first author's Ph.D. work at Carnegie Mellon University.

Transactions of the Association for Computational Linguistics, 2 (2014) 393–404. Action Editor: Robert C. Moore.
Submitted 2/2014; Revised 6/2014; Published 10/2014. ©2014 Association for Computational Linguistics.

score decomposes as a linear sum over both local features and partial hypotheses to efficiently perform inference in these structured spaces (§2) – currently, there are no scalable solutions to integrating the hypothesis-level non-linear feature transforms typically associated with kernel methods while still maintaining polynomial time search. Another alternative is incorporating a recurrent neural network (Schwenk, 2012; Auli et al., 2013; Kalchbrenner and Blunsom, 2013) or an additive neural network (Liu et al., 2013a). While these models have shown promise as methods of augmenting existing models, they have not yet offered a path for replacing or transforming existing real-valued features.

In this article, we discuss background (§2), describe local discretization, our approach to learning non-linear transformations of individual features, compare it with globally non-linear models (§3), present our experimental setup (§5), empirically verify the importance of non-linear feature transformations in MT and demonstrate that discretization can be used to recover non-linear transformations (§6), discuss related work (§7), and conclude (§8).

2 Background and Definitions

2.1 Feature Locality & Structured Hypotheses

Decoding a given source sentence $\mathbf{f}$ can be expressed as search over target hypotheses $\mathbf{e}$, each with an associated complete derivation D. To find the best-scoring hypothesis $\hat{\mathbf{e}}(\mathbf{f})$, a linear model applies a set of weights $\mathbf{w}$ to a complete hypothesis' feature vector $\mathbf{H}$:

$$\hat{\mathbf{e}}(\mathbf{f}) = \arg\max_{\mathbf{e},D} \sum_{i=0}^{|\mathbf{H}|} w_i H_i(\mathbf{f}, \mathbf{e}, D) \qquad (1)$$

However, this hides many of the realities of performing inference in modern decoders. Traditional inference would be intractable if every feature were allowed access to the entire derivation D and its associated target hypothesis $\mathbf{e}$. Decoders take advantage of the fact that features decompose over partial derivations d. For a complete derivation D, the global features $H(D)$ are an efficient summation over local features $h(d)$:

$$\hat{\mathbf{e}}(\mathbf{f}) = \arg\max_{\mathbf{e},D} \sum_{i=0}^{|\mathbf{H}|} w_i \underbrace{\sum_{d \in D} h_i(d)}_{H_i(D)} \qquad (2)$$

This contrasts with non-local features such as the language model (LM), which cannot be exactly calculated given an arbitrary partial hypothesis, which may lack both left and right context.[1] Such features require special handling including future cost estimation. In this study, we limit ourselves to local features, leaving the traditional non-local LM feature unchanged. In general, feature locality is relative to a particular **structured hypothesis space**, and is unrelated to the structured features described in Section 4.2.

2.2 Feature Non-Linearity and Separability

Unlike models that rely primarily on a large number of sparse indicator features, state-of-the-art machine translation systems rely heavily on a small number of dense real-valued features. However, unlike indicator features, real-valued features may benefit from non-linear transformations to allow a linear model to better fit the data.

Decoders use a linear model to rank hypotheses, selecting the highest-*ranked* derivation. Since the absolute score of the model is irrelevant, non-linear responses are useful only in cases where they elicit novel rankings. In this section, we will discuss these cases in terms of separability. Here, we are separating the correctly ranked pairs of hypotheses from the incorrect in the implicit pairwise rankings defined by the total ordering on hypotheses provided by our model.

When the local feature vectors h of each oracle-best[2] hypothesis (or hypotheses) are distinct from those of all other competing hypotheses, we say that the inputs are **oracle separable** given the feature set. If there exists a weight vector that distinguishes the oracle-best ranking from all other rankings under a linear model, we say that the inputs are **linearly separable** given the feature set. If the inputs are oracle separable but not linearly separable, we say that there are **non-linearities** that are unexplained by the feature set. For example, this can happen if a feature is positively related to quality in some regions but negatively related in other regions.

As we add more sentences to our corpus, separability becomes increasingly difficult. For a given

[1] This is especially problematic for chart-based decoders.

[2] We define the oracle-best hypothesis in terms of some external quality measure such as BLEU

corpus, if all hypotheses are oracle separable, we can always produce the oracle translation – assuming an optimal (and potentially very complex) model and weight vector. If our hypothesis space also contains all reference translations, we can always recover the reference. In practice, both of these conditions are typically violated to a certain degree. However, if we modify our feature set such that some lower-ranked higher-quality hypothesis can be separated from all higher-ranked lower-quality hypotheses, then we can improve translation quality. For this reason, we believe that separability remains an informative tool for thinking about modeling in MT.

Currently, non-linearities in novel real-valued features are typically addressed via manual feature engineering involving a good deal of trial and error (Gimpel and Smith, 2009)[3] or by manually discretizing features (e.g. indicator features for count=N). We will explore one technique for automatically avoiding non-linearities in Section 3.

2.3 Learning with Large Feature Sets

While MERT has proven to be a strong baseline, it does not scale to larger feature sets in terms of both inefficiency and overfitting. While MIRA (Chiang et al., 2008), Rampion (Gimpel and Smith, 2012), and HOLS (Flanigan et al., 2013) have been shown to be effective over larger feature sets, they are difficult to explicitly regularize – this will become important in Section 4.2. Therefore, we use the PRO optimizer (Hopkins and May, 2011) as our baseline learner since it has been shown to perform comparably to MERT for a small number of features, and to significantly outperform MERT for a large number of features (Hopkins and May, 2011; Ganitkevitch et al., 2012). Other very recent MT optimizers such as the linear structured SVM (Cherry and Foster, 2012), AROW (Chiang, 2012) and regularized MERT (Galley et al., 2013) are also compatible with the discretization and structured regularization techniques described in this article.[4]

[3]Gimpel et al. eventually used raw probabilities in their model rather than log-probabilities.

[4]Since we dispense with nearly all of the original dense features and our structured regularizer is scale sensitive, one would need to use the ℓ_1-renormalized variant of regularized MERT.

3 Discretization and Feature Induction

In this section, we propose a feature induction technique based on discretization that produces a feature set that is less prone to non-linearities (see §2.2).

We define feature induction as a function $\Phi(y)$ that takes the result of the feature function $y = h(x) \in \mathbb{R}$ and returns a tuple $\langle y', j \rangle$ where $y' \in \mathbb{R}$ is a transformed feature value and j is the transformed feature index.[5] Building on equation 2, we can apply feature induction as follows:

$$\hat{\mathbf{e}}(\mathbf{f}) = \arg\max_{\mathbf{e},\mathbf{D}} \sum_{\mathbf{d}\in\mathbf{D}} \underbrace{\sum_{i=0}^{|\mathbf{H}|} w'_j y'}_{\langle y',j\rangle=\Phi_i(h_i(d))} \qquad (3)$$
$$\mathbf{H}'(\mathbf{f},\mathbf{e},\mathbf{D})$$

At first glance, one might be tempted to simply choose some non-linear function for Φ (e.g. $\log(x), \exp(x), \sin(x), x^n$). However, even if we were to restrict ourselves to some "standard" set of non-linear functions, many of these functions have hyperparameters that are not directly tunable by conventional optimizers (e.g.period and amplitude for sin, n in x^n).

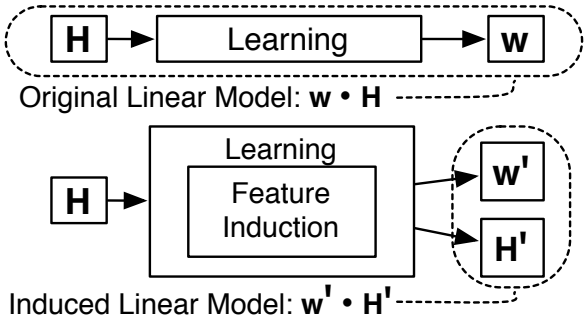


Figure 1: **Top:** A traditional learning procedure, assigning a set of weights to a fixed feature set. **Bottom:** Discretization, our feature induction technique, expands the feature set as part of learning, while still producing a linear model for inference, albeit with more features.

Discretization allows us to avoid many non-linearities (§2.2) while preserving the fast inference provided by feature locality (§2.1). We first discretize real-valued features into a set of indicator

[5]One could also imagine a feature transformation function Φ that returns a vector of bins for a single value returned by a feature function h or a transformation that has access to values from multiple feature functions at once.

features and then use a conventional optimizer to learn a weight for each indicator feature (Figure 1). This technique is sometimes referred to as binning and is closely related to quantization. Effectively, discretization allows us to re-shape a feature function (Figure 2). In fact, given an infinite number of bins, we can perform any non-linear transformation of the original function.

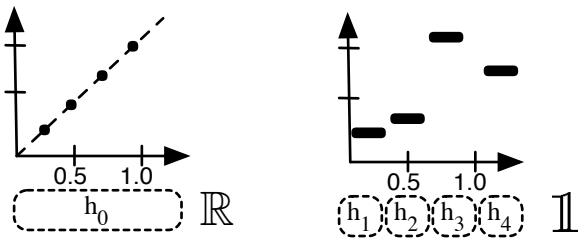


Figure 2: **Left:** A real-valued feature. Bold dots represent points where we could imagine bins being placed. However, since we may only adjust w_0, these "bins" will be rigidly fixed along the feature function's value. **Right:** After discretizing the feature into 4 bins, we may now adjust 4 weights independently, to achieve a non-linear re-shaping of the function.

For indicator discretization, we define Φ_i in terms of a binning function $\text{BIN}_i(x) \in \mathbb{R} \rightarrow \mathbb{N}$:

$$\Phi_i(x) = \langle 1, i \frown \text{BIN}_i(x) \rangle \quad (4)$$

where the $\frown$ operator indicates concatenation of a feature identifier with a bin identifier to form a new, unique feature identifier.

3.1 Local Discretization

Unlike other approaches to non-linear learning in MT, we perform non-linear transformation on *partial* hypotheses as in equation 3 where discretization is applied as $\Phi_i(h_i(d))$, which allows **locally non-linear** transformations, instead of applying Φ to complete hypotheses as in $\Phi_i(H_i(D))$, which would allow **globally non-linear** transformations. This enables our transformed model to produce non-linear responses with regard to the initial feature set $\mathbf{H}$ while inference remains linear with regard to the optimized parameters $\mathbf{w}'$. Importantly, our transformed feature set requires no additional non-local information for inference.

By performing transformations within a local context, we effectively reinterpret the feature set. For example, the familiar target word count feature found in many modern MT systems is often conceptualized as "what is the count of target words in the complete hypothesis?" A hypothesis-level view of discretization would view this as "Did this hypothesis have 5 target words?". Only one such feature will fire for each hypothesis. However, local discretization reinterprets this feature as "How many phrases in the complete hypothesis have 1 target word?" Many such features are likely to fire for each hypothesis. We provide a further example of this technique in Figure 3.

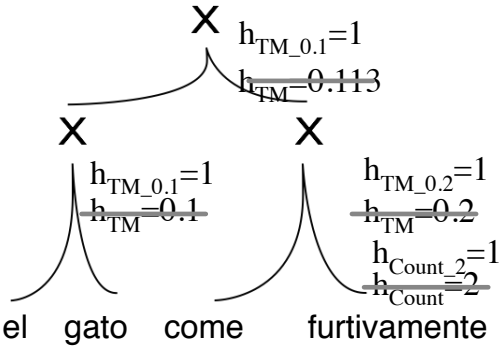


Figure 3: We perform discretization *locally* on each grammar rule or phrase pair, operating on the local feature vectors $\mathbf{h}$. In this example, the original real-valued features are crossed out with a solid gray line and their discretized indicator features are written above. When forming a complete hypothesis from partial hypotheses, we sum the counts of these indicator features to obtain the complete feature vector $\mathbf{H}$. In this example, $\mathbf{H} = \{H_{\text{TM\_0.1}} : 2, H_{\text{TM\_0.2}} : 1, H_{\text{Count\_2}} : 1\}$

In terms of predictive power, this transformation can provide the learned model with increased ability to discriminate between hypotheses. This is primarily a result of moving to a higher-dimensional feature space. As we introduce new parameters, we expect that some hypotheses that were previously indistinguishable under $\mathbf{H}$ become separable under $\mathbf{H}'$ (§2.2). We show specific examples comparing linear, locally non-linear, and globally non-linear models in Figures 4 - 6. As seen in these examples, locally non-linear models (Eq. 3, 4) are not an approximation nor a subset of globally non-linear models, but rather a different class of models.

3.2 Binning Algorithm

To initialize the learning procedure, we construct the binning function BIN used by the indicator di-

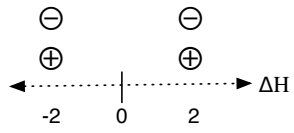
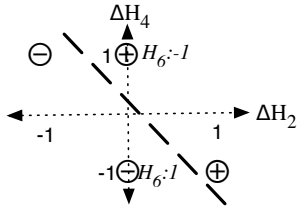
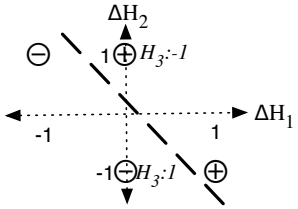
	Linear	Globally Non-Linear	Locally Non-Linear
Ranking	$^*S_3^1$ he says h:1.0 h:1.0 = {H:2.0} S_3^2 she said h:2.0 h:2.0 = {H:4.0} S_4^1 small kitten h:2.0 h:2.0 = {H:4.0} $^*S_4^2$ big lion h:3.0 h:3.0 = {H:6.0}	$^*S_3^1$ he says h:1.0 h:1.0 = {H_2:1} S_3^2 she said h:2.0 h:2.0 = {H_4:1} S_4^1 small kitten h:2.0 h:2.0 = {H_4:1} $^*S_4^2$ big lion h:3.0 h:3.0 = {H_6:1}	$^*S_3^1$ he says h_1:1 h_1:1 = {H_1:2} S_3^2 she said h_2:1 h_2:1 = {H_2:2} S_4^1 small kitten h_2:1 h_2:1 = {H_2:2} $^*S_4^2$ big lion h_3:1 h_3:1 = {H_3:2}
Pairs	(S_3^1, S_3^2) {ΔH:-2.0} ⊕ (S_3^2, S_3^1) {ΔH:2.0} ⊖ (S_4^2, S_4^1) {ΔH:-2.0} ⊖ (S_4^1, S_4^2) {ΔH:2.0} ⊕	(S_3^1, S_3^2) {ΔH_2:1, ΔH_4:-1} ⊕ (S_3^2, S_3^1) {ΔH_2:-1, ΔH_4:1} ⊖ (S_4^2, S_4^1) {ΔH_4:1, ΔH_6:-1} ⊖ (S_4^1, S_4^2) {ΔH_4:-1, ΔH_6:1} ⊕	(S_3^1, S_3^2) {ΔH_1:2, ΔH_2:-2} ⊕ (S_3^2, S_3^1) {ΔH_1:-2, ΔH_2:2} ⊖ (S_4^2, S_4^1) {ΔH_2:2, ΔH_3:-2} ⊖ (S_4^1, S_4^2) {ΔH_2:-2, ΔH_3:2} ⊕
Pairwise Ranking	 **Inseparable**	 **Separable**	 **Separable**

Figure 4: An example showing a **collinearity** over multiple input sentences S_3, S_4 in which the oracle-best hypothesis is "trapped" along a line with other lower quality hypotheses in the linear model's output space. **Ranking** shows how the hypotheses would appear in a k-best list with each partial derivation having its partial feature vector h under it; the complete feature vector H is shown to the right of each hypothesis and the oracle-best hypothesis is notated with a *. **Pairs** explicates the implicit pairwise rankings. **Pairwise Ranking** graphs those pairs in order to visualize whether or not the hypotheses are separable. ($\oplus$ indicates that the pair of hypotheses is ranked correctly according to the extrinsic metric and $\ominus$ indicates the pair is ranked incorrectly. In the pairwise ranking row, some $\oplus$ and $\ominus$ points are annotated with their positions along the third axis H_3 (omitted for clarity). Collinearity can also occur with a single input having at least 3 hypotheses.

	Linear	Globally Non-Linear	Locally Non-Linear
Ranking	$^*S_2^1$ some things h:2.0 h:2.0 = {H:4.0} S_2^2 something h:4.0 = {H:4.0}	$^*S_2^1$ some things h:2.0 h:2.0 = {H_4:1} S_2^2 something h:4.0 = {H_4:1}	$^*S_2^1$ some things h_2:1 h_2:1 = {H_2:2} S_2^2 something h_4:1 = {H_4:1}
Pairs	(S_2^1, S_2^2) {ΔH:0.0} ⊕ (S_2^2, S_2^1) {ΔH:0.0} ⊖	(S_2^1, S_2^2) {ΔH_4:0} ⊕ (S_2^2, S_2^1) {ΔH_4:0} ⊖	(S_2^1, S_2^2) {ΔH_2:2, ΔH_4:-1} ⊕ (S_2^2, S_2^1) {ΔH_2:-2, ΔH_4:1} ⊖
Pairwise Ranking	**Inseparable**	**Inseparable**	**Separable**

Figure 5: An example showing a trivial "collision" in which two hypotheses of differing quality receive the same model score until local discretization is applied. The two hypotheses are indistinguishable under a linear model with the feature set H, as shown by the zero-difference in the "pairs" row. While a globally non-linear transformation does not yield any improvement, local discretization allows the hypotheses to be properly ranked due to the higher-dimensional feature space H_2, H_4. See Figure 4 for an explanation of notation.

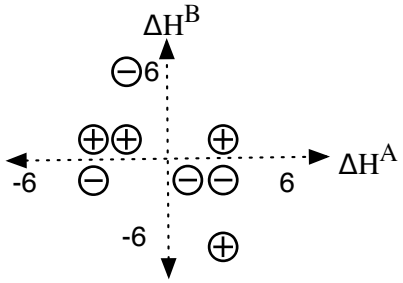
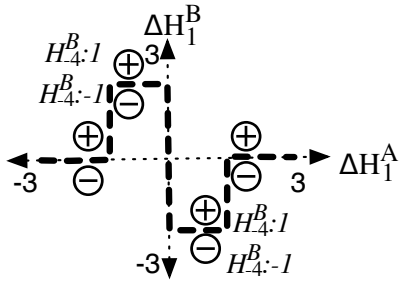
	Linear	**Locally Non-Linear**
Ranking	$^*h^B$:1.0 = {H^B:1.0} h^A:1.0 h^A:1.0 = {H^A:2.0}	$^*h^B_1$:1 = {H^B_1:1} h^A_1:1 h^A_1:1 = {H^A_1:2}
	$^*h^A$:1.0 h^A:1.0 h^B:1.0 = {H^A:2.0, H^B:1.0} h^A:0.0 = {}	$^*h^A_1$:1 h^A_1:1 h^B_1:1 = {H^A_1:2, H^B_1:1} h^A_1:0 = {}
	* h^B:-4.0 h^B:1.0 h^B:1.0 h^B:1.0 = {H^B:-1.0} h^A:1.0 = {H^A:1.0}	$^*h^B_{-4}$:1 h^B_1:1 h^B_1:1 h^B_1:1 = {H^B_{-4}:1, H^B_1:3} h^A_1:1 = {H^A_1:1}
	$^*h^B$:-4.0 h^A:1.0 = {H^A:1.0, H^B:-4.0} h^B:1.0 h^B:1.0 = {H^B:2.0}	$^*h^B_{-4}$:1 h^A_1:1 = {H^A_1:1, H^B_{-4}:1} h^B_1:1 h^B_1:1 = {H^B_1:2}
Pairwise Ranking	 **Inseparable**	 **Separable**

Figure 6: An example demonstrating a non-linear decision boundary induced by discretization. The non-linear nature of the decision boundary can be seen clearly when the induced feature set H^A_1, H^B_1, H^B_{-4} (right) is considered in the original feature space H^A, H^B (left). In the **pairwise ranking** row, two axes (H^A_1, H^B_1) are plotted while the third axis H^B_{-4} is indicated only as stand-off annotations for clarity . Given a larger number of hypotheses, such situations could also arise within a single sentence. See Figure 4 for an explanation of notation.

cretizer Φ. We have two desiderata: (1) any monotonic transformation of a feature should not affect the induced binning since we should not require feature engineers to determine the optimal feature transformation and (2) no bin's data should be so sparse that the optimizer cannot reliably estimate a weight for each bin. Therefore, we construct bins that are (i) populated uniformly subject to (ii) each bin containing no more than one feature value. We call this approach **uniform population feature binning**. While one could consider the predictive power of the features when determining bin boundaries, this would suggest that we should jointly optimize and determine bin boundaries, which is beyond the scope of this work. This problem has recently been considered for NLP by Suzuki and Nagata (2013) and for MT by Liu et al. (2013b), though the latter involves decoding the entire training data.

Let $\mathcal{X}$ be the list of feature values to bin where i indexes feature values $x_i \in \mathcal{X}$ and their associated frequencies f_i. We want each bin to have a uniform size u. For the sake of simplifying our final algorithm, we first create adjusted frequencies f'_i so that very frequent feature values will not occupy more than 100% of a bin via the following algorithm, which iterates over k:

$$u^k = \frac{1}{|\mathcal{X}|} \sum_{i=1}^{|\mathcal{X}|} f_i^k \quad (5)$$

$$f_i^{k+1} = \min(f_i^k, u^k) \quad (6)$$

which returns $u' = u^k$ when $f_i^k < u^k \; \forall i$. Next, we solve for a binning B of N bins where $\overline{\overline{b_j}}$ is the population of each bin:

$$\arg\min_B \frac{1}{N} \sum_{j=1}^{N} |\overline{\overline{b_j}} - u'| \quad (7)$$

We use Algorithm 1 to produce this binning. In our experiments, we construct a translation model for each sentence in our tuning corpus; we then add a feature value instances to $\mathcal{X}$ for each rule instance.

Algorithm 1 POPULATEBINSUNIFORMLY$(\mathcal{X}, N)$

$\triangleright$ Remaining values for b_j, s.t. $\overline{\overline{b_k}} > 0 \; \forall k$
def $R(j) = |\mathcal{X}| - (N - j - 1)$
$\triangleright$ Remaining frequency mass within ideal bound
def $C(j) = j \cdot u' - \sum_k^j \overline{\overline{b_k}}$
$i \leftarrow 1$ $\triangleright$ Current feature value
for $j \in [1, N]$ **do**
  while $i \leq R(j)$ **and** $f_i \leq C(j)$ **do**
    $b_j \leftarrow b_j \cup \{x_i\}$
    $i \leftarrow i + 1$
  end while
  $\triangleright$ Handle value that straddles ideal boundaries by minimizing its violation of the ideal
  if $i \leq R(j)$ **and** $\frac{f_i - C(j)}{f_i} < 0.5$ **then**
    $b_j \leftarrow b_j \cup \{x_i\}$
    $i \leftarrow i + 1$
  end if
end for
return B

4 Structured Regularization

Unfortunately, choosing the right number of bins can have important effects on the model, including:
Fidelity. If we choose too few bins, we risk degrading the model's performance by discarding important distinctions encoded in fine differences between the feature values. In the extreme, we could reduce a real-valued feature to a single indicator feature.
Sparsity. If we choose too many bins, we risk making each indicator feature too sparse, which is likely to result in the optimizer overfitting such that we generalize poorly to unseen data.

While one may be tempted to simply throw more data or millions of sparse features at the problem, we elect to more strategically use existing data, since (1) large in-domain tuning data is not always available, and (2) when it is available, it can add considerable computational expense. In this section, we explore methods for mitigating data sparsity by embedding more knowledge into the learning procedure.

4.1 Overlapping Bins

One very simplistic way we could combat sparsity is to extend the edges of each bin such that they cover their neighbors' values (see Equation 4):

$$\Phi'_i(x) = \langle 1, i \frown \text{BIN}_i(x) \rangle \textbf{ if } x \in \cup_{k=i-1}^{i+1} \text{BIN}_k \quad (8)$$

This way, each bin will have more data points to estimate its weight, reducing data sparsity, and the bins will mutually constrain each other, reducing the ability to overfit. We include this technique as a contrastive baseline for structured regularization.

4.2 Linear Neighbor Regularization

Regularization has long been used to discourage optimization solutions that give too much weight to any one feature. This encodes our prior knowledge that such solutions are unlikely to generalize. Regularization terms such as the ℓ_p norm are frequently used in gradient-based optimizers including our baseline implementation of PRO.

Unregularized discretization is potentially brittle with regard to the number of bins chosen. Primarily, it suffers from sparsity. At the same time, we note that we know much more about discretized features than initial features since we control how they are formed. These features make up a **structured feature space**. With these things in mind, we propose linear neighbor regularization, a structured regularizer that embeds a small amount of knowledge into the objective function: that the indicator features resulting from the discretization of a single real-valued feature are spatially related. We expect similar weights to be given to the indicator features that represent neighboring values of the original real-valued feature such that the resulting transformation appears somewhat smooth.

To incorporate this knowledge of nearby bins, the linear neighbor regularizer $\mathcal{R}_{\text{LNR}}$ penalizes each feature's weight by the squared amount it differs from its neighbors' midpoint:

$$\mathcal{R}_{\text{LNR}}(\mathbf{w}, j) = \left(\frac{1}{2}(w_{j-1} + w_{j+1}) - w_j \right)^2 \quad (9)$$

$$\mathcal{R}_{\text{LNR}}(\mathbf{w}) = \beta \sum_{j=2}^{|\mathbf{h}|-1} \mathcal{R}_{\text{LNR}}(\mathbf{w}, j) \quad (10)$$

This is a special case of the feature network regularizer of Sandler (2010). Unlike traditional regularizers, we do not hope to reduce the active feature count. With the PRO loss l and a ℓ_2 regularizater $\mathcal{R}_2$, our final loss function internal to each iteration of PRO is:

$$\mathcal{L}(\mathbf{w}) = l(\mathbf{x}, \mathbf{y}; \mathbf{w}) + \mathcal{R}_2(\mathbf{w}) + \mathcal{R}_{\text{LNR}}(\mathbf{w}) \quad (11)$$

4.3 Monotone Neighbor Regularization

However, as $\beta \to \infty$, the linear neighbor regularizer $\mathcal{R}_{\text{LNR}}$ forces a linear arrangement of weights – this violates our premise that we should be agnostic to non-linear transformations. We now describe a structured regularizer $\mathcal{R}_{\text{MNR}}$ whose limiting solution is any monotone arrangement of weights. We augment $\mathcal{R}_{\text{LNR}}$ with a smooth damping term $\mathcal{D}(\mathbf{w}, j)$, which has the shape of a bathtub curve with steepness γ:

$$\mathcal{D}(\mathbf{w}, j) = \tanh^{2\gamma} \frac{\frac{1}{2}(w_{j-1} + w_{j+1}) - w_j}{\frac{1}{2}(w_{j-1} - w_{j+1})} \quad (12)$$

$$\mathcal{R}_{\text{MNR}}(\mathbf{w}) = \beta \sum_{j=2}^{|\mathbf{h}|-1} \mathcal{D}(\mathbf{w}, j) \mathcal{R}_{\text{LNR}}(\mathbf{w}, j) \quad (13)$$

$\mathcal{D}$ is nearly zero while $w_j \in [w_{j-1}, w_{j+1}]$ and nearly one otherwise. Briefly, the numerator measures how far w_j is from the midpoint of w_{j-1} and w_{j+1} while the denominator scales that distance by the radius from the midpoint to the neighboring weight.

5 Experimental Setup[6]

Formalism: In our experiments, we use a hierarchical phrase-based translation model (Chiang, 2007). A corpus of parallel sentences is first word-aligned, and then phrase translations are extracted heuristically. In addition, hierarchical grammar rules are extracted where phrases are nested. In general, our choice of formalism is rather unimportant – our techniques should apply to most common phrase-based and chart-based paradigms including Hiero and syntactic systems.

Decoder: For decoding, we will use cdec (Dyer et al., 2010), a multi-pass decoder that supports syntactic translation models and sparse features.

Optimizer: Optimization is performed using PRO (Hopkins and May, 2011) as implemented by the cdec decoder. We run PRO for 30 iterations as suggested by Hopkins and May (2011). The PRO optimizer internally uses a L-BFGS optimizer with the default ℓ_2 regularization implemented in cdec. Any additional regularization is explicitly noted.

Baseline Features: We use the baseline features produced by Lopez' suffix array grammar extractor (Lopez, 2008), which is distributed with cdec. Bidirectional lexical log-probabilities, the coherent phrasal translation log-probability, target word count, glue rule count, source OOV count, target OOV count, and target language model log-probability. Note that these features may be simplified or removed as specified in each experimental condition.

	Zh→En	Ar→En	Cz→En
Train	303K	5.4M	1M
WeightTune	1664	1797	3000
HyperTune	1085	1056	2000
Test	1357	1313	2000

Table 1: Corpus statistics: number of parallel sentences.

Chinese Resources: For the Chinese→English experiments, including the completed work presented in this proposal, we train on the Foreign Broadcast Information Service (FBIS) corpus of approximately 300,000 sentence pairs with about 9.4 million English words. We tune weights on the NIST MT 2006 dataset, tune hyperparameters on NIST MT05, and test on NIST MT 2008.

Arabic Resources: We build an Arabic→English system, training on the large NIST MT 2009 constrained training corpus of approximately 5 million sentence pairs with about 181 million English words. We tune weights on the NIST MT 2006 dataset, tune hyperparameters on NIST MT 2005, and test on NIST MT 2008.

Czech resources: We also construct a Czech→English system based on the CzEng 1.0 data (Bojar et al., 2012). First, we lowercased and performed sentence-level deduplication of the data.[7] Then, we uniformly sampled a training set of 1M sentences (sections 1 – 97) along with a weight-tuning set (section 98), hyperparameter-tuning (section 99), and test set (section 99) from the paraweb domain contained of CzEng.[8] Sentences less than 5 words were discarded due to noise.

Evaluation: We quantify increases in translation quality using case-insensitive BLEU (Papineni et al., 2002). We control for test set variation and optimizer instability by averaging over multiple optimizer replicas (Clark et al., 2011).[9]

[6] All code at `http://github.com/jhclark/cdec`

[7] CzEng is distributed deduplicated at the document level, leading to very high sentence-level overlap.

[8] The section splits recommended by Bojar et al. (2012).

[9] MultEval 0.5.1: `github.com/jhclark/multeval`

Bits	4	8	12
Features	101	1302	12,910
Test BLEU	36.4	36.6	36.8

Table 2: Translation quality for Cz→En system with varying bits for discretization. For all other experiments, we tune the number of bits on held-out data.

Condition	**Zh→En**	**Ar→En**	**Cz→En**
P	20.8* (-2.7)	44.3* (-3.6)	36.5* (-1.1)
`log` P	23.5†	47.9†	37.6†
Disc P	23.4† (-0.1)	47.2† (-0.7)	36.8* (-0.8)
Over. P	20.7* (-2.8)	44.6* (-3.3)	36.6* (-1.0)
LNR P	23.1*† (-0.4)	48.0† (+0.1)	37.3 (-0.3)
MNR P	23.8† (+0.3)	48.7*† (+0.8)	37.6† (±)
MNR C	23.6† (±)	48.7*† (+0.8)	37.4† (-0.2)

Table 3: **Top:** Translation quality for systems with and without the typical `log` transform. **Bottom:** Translation quality for systems using discretization and structured regularization with probabilities P or counts C as the input of discretization. MNR P consistently recovers or outperforms a state-of-the-art system, but without any assumptions about how to transform the initial features. All scores are averaged over 3 end-to-end optimizer replications. * denotes significantly different than log probs (row2) with $p(\text{CHANCE}) < 0.01$ under Clark et al. (2011) and † is likewise used with regard to P (row 1).

6 Results

6.1 Does Non-Linearity Matter?

In our first set of experiments, we seek to answer "Does non-linearity matter?" by starting with our baseline system of 7 typical features (the `log` Probability system) and we then remove the log transform from all of the log probability features in our grammar (the Probs. system). The results are shown in Table 3 (rows 1, 2). If a naïve feature engineer were to remove the non-linear log transform, the systems would degrade between 1.1 BLEU and 3.6 BLEU. From this, we conclude that non-linearity does affect translation quality. This is a potential pitfall for any real-valued feature including probability features, count features, similarity measures, etc.

6.2 Learning Non-Linear Transformations

Next, we evaluate the effects of discretization (Disc), overlapping bins (Over.), linear neighbor regularization (LNR), and monotone neighbor regularization (MNR) on three language pairs: a small Zh→En system, a large Ar→En system and a large Cz→En system. In the first row of Table 3, we use raw probabilities rather than `log` probabilities for $p_{\text{coherent}}(t|s)$, $p_{\text{lex}}(t|s)$, and $p_{\text{lex}}(s|t)$. In rows 3 – 7, all translation model features (without the log-transformed features) are then discretized into indicator features.[10] The number of bins and the structured regularization strength were tuned on the hyperparameter tuning set.

Discretization alone does not consistently recover the performance of the `log` transformed features (row 3). The naïve overlap strategy in fact degrades performance (row 4). Linear neighbor regularization (row 5) behaves more consistently than discretization alone, but is consistently outperformed by the monotone neighbor regularizer (row 6), which is able to meet or significantly exceed the performance of the `log` transformed system. Importantly, this is done without any knowledge of the correct non-linear transformation. In the final row, we go a step further by removing $p_{\text{coherent}}(t|s)$ altogether and replacing it with simple count features: $c(s)$ and $c(s,t)$, with slight to no degradation in quality.[11] We take this as evidence that a feature engineer developing a new real-valued feature may find discretization and monotone neighbor regularization useful.

We also observe that different data sets benefit from non-linear feature transformation in to different degrees (Table 3, rows 1, 2). We noticed that discretization with monotone neighbor regularization is able to improve over a log transform (rows 2, 6) in proportion to the improvement of a log transform over probability-based features (rows 1, 2).

To provide insight into how translation quality can be affected by the number of bits for discretization, we offer Table 2.

In Figure 7, we present the weights learned by the Ar→En system for probability-based features. We see that even without a bias toward a `log` transform, a `log`-like shape still emerges for many SMT features based only on the criteria of optimizing BLEU and a preference for monotonicity. However, the optimizer chooses some important variations on the log curve, especially for low probabilities, that lead to

[10] We also keep a real-valued copy of the word penalty to help normalize the language model.

[11] These features can single-out rules with $c(s) = 1, c(s,t) = 1$, subsuming separate low-count features

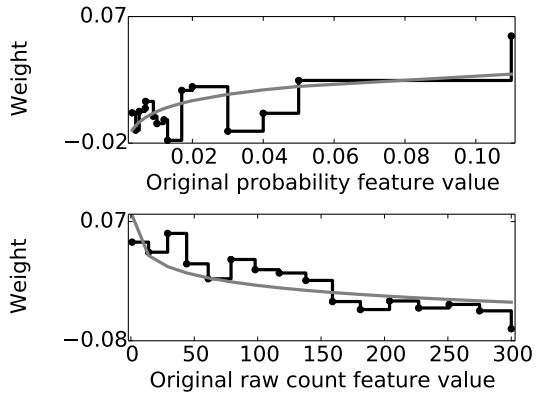


Figure 7: Plots of weights learned for the discretized $p_{\text{coherent}}(e|f)$ (top) and $c(f)$ (bottom) for the Ar→En system with 4 bits and monotone neighbor regularization. $p(e|f) > 0.11$ is omitted for exposition as values were constant after this point. The gray line fits a log curve to the weights. The system learns a shape that deviates from the log in several regions. Each non-monotonic segment represents the learner choosing to better fit the data while paying a strong regularization penalty.

7 Related Work

Previous work on feature discretization in machine learning has focused on the conversion of real-valued features into discrete values for learners that are either incapable of handling real-valued inputs or perform suboptimally given real-valued inputs (Dougherty et al., 1995; Kotsiantis and Kanellopoulos, 2006). Decision trees and random forests have been successfully used in language modeling (Jelinek et al., 1994; Xu and Jelinek, 2004) and parsing (Charniak, 2010; Magerman, 1995).

Kernel methods such as support vector machines (SVMs) are often considered when non-linear interactions between features are desired since they allow for easy usage of non-linear kernels. Wu et al. (2004) showed improvements using non-linear kernel PCA for word sense disambiguation. Taskar et al. (2003) describes a method for incorporating kernels into structured Markov networks. Tsochantaridis et al. (2004) then proposed a structured SVM for grammar learning, named-entity recognition, text classification, and sequence alignment. This was followed by a structured SVM with inexact inference (Finley and Joachims, 2008) and the latent structured SVM (Yu and Joachims, 2009). Even within kernel methods, learning non-linear mappings with kernels remains an open area of research; For example, Cortes et al. (2009) investigated learning non-linear combinations of kernels. In MT, Giménez and Màrquez (2007) used a SVM to annotate a phrase table with binary features indicating whether or not a phrase translation was appropriate in context. Nguyen et al. (2007) also applied non-linear features for SMT n-best reranking.

Toutanova and Ahn (2013) use a form of regression decision trees to induce locally non-linear features in a n-best reranking framework. He and Deng (2012) directly optimize the lexical and phrasal features using expected BLEU. Nelakanti et al. (2013) use tree-structured ℓ_p regularizers to train language models and improve perplexity over Kneser-Ney.

Learning parameters under weak order restrictions has also been studied for regression. Isotonic regression (Barlow et al., 1972; Robertson et al., 1988; Silvapulle and Sen, 2005) fits a curve to a set of data points such that each point in the fitted curve is greater than or equal to the previous point in the curve. Nearly isotonic regression allows violations in monotonicity (Tibshirani et al., 2011).

8 Conclusion

In the absence of highly refined knowledge about a feature, discretization with structured regularization enables higher quality impact of new feature sets that contain non-linearities. In our experiments, we observed that discretization out-performed naïve features lacking a good non-linear transformation by up to 4.4 BLEU and that it can outperform a baseline by up to 0.8 BLEU while dropping the log transform of the lexical probabilities and removing the phrasal probabilities in favor of counts. Looking beyond this basic feature set, non-linear transformations could be the difference between showing quality improvements or not for novel features. As researchers include more real-valued features including counts, similarity measures, and separately-trained models with millions of features, we suspect this will become an increasingly relevant issue. We conclude that non-linear responses play an important role in SMT, even for a commonly-used feature set, an observation that we hope will inform feature engineers.

Acknowledgments

This work was supported by Google Faculty Research grants 2011_R2_705 and 2012_R2_10 and by the NSF-sponsored XSEDE computing resources program under grant TG-CCR110017.

References

Michael Auli, Michel Galley, Chris Quirk, and Geoffrey Zweig. 2013. Joint Language and Translation Modeling with Recurrent Neural Networks. In *Empirical Methods in Natural Language Processing*, number October, pages 1044–1054.

R. E. Barlow, D. Bartholomew, J. M. Bremner, and H. D. Brunk. 1972. *Statistical inference under order restrictions; the theory and application of isotonic regression*. Wiley.

Ondej Bojar, Zdeněk Žabokrtský, Ondej Dušek, Petra Galuščáková, Martin Majliš, David Mareček, Jií Maršík, Michal Novák, Martin Popel, and Aleš Tamchyna. 2012. The Joy of Parallelism with CzEng 1 .0. In *Proceedings of LREC2012*, Istanbul, Turkey. European Language Resources Association.

Peter E Brown, Stephen A Della Pietra, Vincent J Della Pietra, and Robert L Mercer. 1993. The Mathematics of Statistical Machine Translation : Parameter Estimation. *Computational Linguistics*, 10598.

Eugene Charniak. 2010. Top-Down Nearly-Context-Sensitive Parsing. In *Empirical Methods in Natural Language Processing*, number October, pages 674–683.

Colin Cherry and George Foster. 2012. Batch Tuning Strategies for Statistical Machine Translation. In *Proceedings of the North American Association for Computational Linguistics*, pages 427–436.

David Chiang, Yuval Marton, and Philip Resnik. 2008. Online large-margin training of syntactic and structural translation features. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing - EMNLP '08*, pages 224–233, Morristown, NJ, USA. Association for Computational Linguistics.

David Chiang. 2007. Hierarchical Phrase-Based Translation. *Computational Linguistics*, 33(2):201–228, June.

David Chiang. 2012. Hope and fear for discriminative training of statistical translation models. *Journal of Machine Learning Research*, 13:1159–1187.

Jonathan H Clark, Chris Dyer, Alon Lavie, and Noah A Smith. 2011. Better Hypothesis Testing for Statistical Machine Translation: Controlling for Optimizer Instability. In *Association for Computational Linguistics*.

Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. 2009. Learning Non-Linear Combinations of Kernels. In *Advances in Neural Information Processing Systems (NIPS 2009)*, pages 1–9, Vancouver, Canada.

James Dougherty, Ron Kohavi, and Mehran Sahami. 1995. Supervised and Unsupervised Discretization of Continuous Features. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 194–202, San Francisco, CA.

Chris Dyer, Jonathan Weese, Adam Lopez, Vladimir Eidelman, Phil Blunsom, and Philip Resnik. 2010. cdec: A Decoder, Alignment, and Learning Framework for Finite-State and Context-Free Translation Models. In *Association for Computational Linguistics*, number July, pages 7–12.

Thomas Finley and Thorsten Joachims. 2008. Training structural SVMs when exact inference is intractable. In *Proceedings of the International Conference on Machine Learning*, pages 304–311, New York, New York, USA. ACM Press.

Jeffrey Flanigan, Chris Dyer, and Jaime Carbonell. 2013. Large-Scale Discriminative Training for Statistical Machine Translation Using Held-Out Line Search. In *North American Association for Computational Linguistics*, number June, pages 248–258.

Michel Galley, Chris Quirk, Colin Cherry, and Kristina Toutanova. 2013. Regularized Minimum Error Rate Training. In *Empirical Methods in Natural Language Processing*.

Juri Ganitkevitch, Yuan Cao, Jonathan Weese, Matt Post, and Chris Callison-Burch. 2012. Joshua 4.0: Packing, PRO, and Paraphrases. In *Workshop on Statistical Machine Translation*, pages 283–291.

Jesús Giménez and Lluís Màrquez. 2007. Context-aware Discriminative Phrase Selection for Statistical Machine Translation. In *Workshop on Statistical Machine Translation*, number June, pages 159–166.

Kevin Gimpel and Noah A Smith. 2009. Feature-Rich Translation by Quasi-Synchronous Lattice Parsing. In *Empirical Methods in Natural Language Processing*.

Kevin Gimpel and Noah A Smith. 2012. Structured Ramp Loss Minimization for Machine Translation. In *North American Association for Computational Linguistics*.

Xiaodong He and Li Deng. 2012. Maximum Expected BLEU Training of Phrase and Lexicon Translation Models. In *Proceedings of the Association for Computational Linguistics*, Jeju Island, Korea. Microsoft Research.

Mark Hopkins and Jonathan May. 2011. Tuning as Ranking. *Computational Linguistics*, pages 1352–1362.

Frederick Jelinek, John Lafferty, David Magerman, Robert Mercer, Adwait Ratnaparkhi, and Salim Roukos. 1994. Decision Tree Parsing using a Hidden Derivation Model. In *Workshop on Human Language Technologies (HLT)*.

Nal Kalchbrenner and Phil Blunsom. 2013. Recurrent Continuous Translation Models. In *Empirical Methods in Natural Language Processing*.

Sotiris Kotsiantis and Dimitris Kanellopoulos. 2006. Discretization Techniques : A recent survey. In *GESTS International Transactions on Computer Science and Engineering*, volume 32, pages 47–58.

Lemao Liu, Taro Watanabe, Eiichiro Sumita, and Tiejun Zhao. 2013a. Additive Neural Networks for Statistical Machine Translation. In *Proceedings of the Association for Computational Linguistics*.

Lemao Liu, Tiejun Zhao, Taro Watanabe, and Eiichiro Sumita. 2013b. Tuning SMT with A Large Number of Features via Online Feature Grouping. In *Proceedings of the International Joint Conference on Natural Language Processing*.

Adam Lopez. 2008. Tera-Scale Translation Models via Pattern Matching. In *Association for Computational Linguistics Computational Linguistics*, number August, pages 505–512.

David M Magerman. 1995. Statistical Decision-Tree Models for Parsing. In *Association for Computational Linguistics*, pages 276–283.

Anil Nelakanti, Cedric Archambeau, Julien Mairal, Francis Bach, and Guillaume Bouchard. 2013. Structured Penalties for Log-linear Language Models. In *Empirical Methods in Natural Language Processing*, Seattle, WA.

Patrick Nguyen, Milind Mahajan, Xiaodong He, and Microsoft Way. 2007. Training Non-Parametric Features for Statistical Machine Translation. In *Association for Computational Linguistics*.

Franz Josef Och and Hermann Ney. 2002. Discriminative training and maximum entropy models for statistical machine translation. In *Proceedings of the Association for Computational Linguistics*, number July, page 295, Morristown, NJ, USA. Association for Computational Linguistics.

Franz J Och. 2003. Minimum Error Rate Training in Statistical Machine Translation. In *Association for Computational Linguistics*, number July, pages 160–167.

Kishore Papineni, Salim Roukos, Todd Ward, and Weijing Zhu. 2002. BLEU : a Method for Automatic Evaluation of Machine Translation. In *Computational Linguistics*, number July, pages 311–318.

Tim Robertson, F. T. Wright, and R. L. Dykstra. 1988. *Order Restricted Statistical Inference*. Wiley.

S Ted Sandler. 2010. *Regularized Learning with Feature Networks*. Ph.D. thesis, University of Pennsylvania.

Holger Schwenk. 2012. Continuous Space Translation Models for Phrase-Based Statistical Machine Translation. In *International Conference on Computational Linguistics (COLING)*, number December 2012, pages 1071–1080, Mumbai, India.

Mervyn J. Silvapulle and Pranab K. Sen. 2005. *Constrained Statistical Inference: Order, Inequality, and Shape Constraints*. Wiley.

Artem Sokolov, Guillaume Wisniewski, and François Yvon. 2012. Non-linear N-best List Reranking with Few Features. In *Association for Machine Translation in the Americas*.

Jun Suzuki and Masaaki Nagata. 2013. Supervised Model Learning with Feature Grouping based on a Discrete Constraint. In *Proceedings of the Association for Computational Linguistics*, pages 18–23.

Ben Taskar, Carlos Guestrin, and Daphne Koller. 2003. Max-Margin Markov Networks. In *Neural Information Processing Systems*.

Ryan J Tibshirani, Holger Hoefling, and Robert Tibshirani. 2011. Nearly-Isotonic Regression. *Technometrics*, 53(1):54–61.

Kristina Toutanova and Byung-Gyu Ahn. 2013. Learning Non-linear Features for Machine Translation Using Gradient Boosting Machines. In *Proceedings of the Association for Computational Linguistics*.

Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. 2004. Support Vector Machine Learning for Interdependent and Structured Output Spaces. In *International Conference on Machine Learning (ICML)*.

Zhuoran Wang, John Shawe-Taylor, and Sandor Szedmak. 2007. Kernel Regression Based Machine Translation. In *North American Association for Computational Linguistics*, number April, pages 185–188, Rochester, N.

Dekai Wu, Weifeng Su, and Marine Carpuat. 2004. A Kernel PCA Method for Superior Word Sense Disambiguation. In *Association for Computational Linguistics*, Barcelona.

Peng Xu and Frederick Jelinek. 2004. Random Forests in Language Modeling. In *Empirical Methods in Natural Language Processing*.

Chun-Nam John Yu and Thorsten Joachims. 2009. Learning structural SVMs with latent variables. In *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*, pages 1–8, New York, New York, USA. ACM Press.