Dynamic Feature Induction: The Last Gist to the State-of-the-Art

Jinho D. Choi
Department of Mathematics and Computer Science
Emory University
Atlanta, GA 30322, USA
jinho.choi@emory.edu

Abstract

We introduce a novel technique called dynamic feature induction that keeps inducing high dimensional features automatically until the feature space becomes 'more' linearly separable. Dynamic feature induction searches for the feature combinations that give strong clues for distinguishing certain label pairs, and generates joint features from these combinations. These induced features are trained along with the primitive low dimensional features. Our approach was evaluated on two core NLP tasks, part-of-speech tagging and named entity recognition, and showed the state-of-the-art results for both tasks, achieving the accuracy of 97.64 and the F1-score of 91.00 respectively, with about a 25% increase in the feature space.

1 Introduction

Feature engineering typically involves two processes: the process of discovering novel features with domain knowledge, and the process of optimizing combinations between existing features. Discovering novel features may require linguistic background as well as good understanding in machine learning such that it is often difficult to do. Optimizing feature combinations can be also difficult but usually requires less domain knowledge and more importantly, it can be as effective as discovering new features. It has been shown for many tasks that approaches using simple machine learning with extensive feature engineering outperform ones using more advanced machine learning with less intensive feature engineering (Xue and Palmer, 2004; Bengtson and Roth, 2008; Ratinov and Roth, 2009; Zhang and Nivre, 2011).

Recently, people have tried to automate the second part of feature engineering, the optimization of feature combinations, through leading-edge models such as neural networks (Collobert et al., 2011). Coupled with embedding approaches (Mikolov et al., 2013; Le and Mikolov, 2014; Pennington et al., 2014), neural networks can find the optimal feature combinations using techniques such as random weight initialization and back-propagation, and have established the new state-of-the-art for several tasks (Socher et al., 2013; Devlin et al., 2014; Yu et al., 2014). However, neural networks are not as good at optimizing combinations between sparse features, which are still the most dominating factors in natural language processing.

This paper introduces a new technique called dynamic feature induction that automates the optimization of feature combinations (Section 3), and can be easily adapted to any NLP task using sparse features. Dynamic feature induction allows humans to focus on the first part of feature engineering, the discovery of novel features, while machines handle the second part. Our approach was experimented with two core NLP tasks, part-of-speech tagging (Section 4) and named entity recognition (Section 5) and showed the state-of-the-art results for both tasks.

2 Background

2.1 Nonlinearity in NLP

Linear classification algorithms such as Perceptron, Winnow, or Support Vector Machines with a linear kernel have performed exceptionally well for various NLP tasks (Collins, 2002; Zhang and Johnson, 2003; Pradhan et al., 2005). This is not because our feature space is linearly separable by nature, but sparse fea-

Proceedings of NAACL-HLT 2016, pages 271–281,
San Diego, California, June 12-17, 2016.

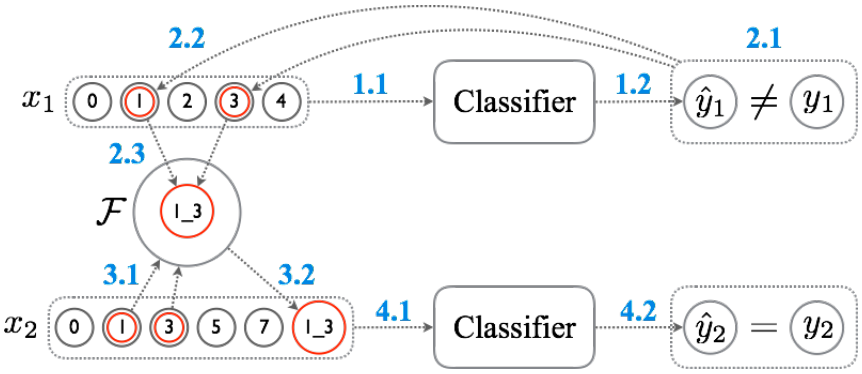

Figure 1: Overview of dynamic feature induction.

tures introduced to NLP yield very high dimensional vector space such that it is rather forced to be linearly separable. For example, NLP features for a word w_i typically involve the word forms of w_{i-1} and w_i (e.g., f_{i-1}, f_i). If the feature space is not linearly separable with these features, a common trick is to introduce 'higher' dimension features by joining 'lower' dimension features together (e.g., f_{i-1}-f_i). The more joint features we introduce, the higher chance we get for the feature space being linearly separable although these joint features can be very overfitted.

Let us define low dimensional features as the primitive features such as f_{i-1} or f_i, and high dimensional features as the joint features such as f_i-f_{i+1}. [1] Low dimensional features are well explored for most NLP tasks; it is the high dimensional features that are quite sensitive to specific tasks. Finding high dimensional features can be a manual intensive work and this is what dynamic feature induction intends to take over.

2.2 Related Work

Kudo and Matsumoto (2003) introduced the polynomial kernel expansion that explicitly enumerated the feature combinations. Our approach is distinguished because they used a frequency-based PrefixSpan algorithm (Pei et al., 2001) whereas we used the online learning weights for finding the feature combinations. Goldberg and Elhadad (2008) suggested an efficient algorithm for computing polynomial kernel SVMs by combining inverted indexing and kernel expansion. Their work is focused more on improving support vector machines whereas our work is generalized to any linear classification algorithm.

Okanohara and Tsujii (2009) introduced an approach for generating feature combinations using ℓ_1 regularization and grafting (Perkins et al., 2003). Although we share similar ideas, their grafting algorithm starts with an empty feature set whereas ours starts with low dimensional features, and their correlation parameters $\alpha_{i,y}$ are pre-computed whereas ours are dynamically determined. Strubell et al. (2015) suggested an algorithm that dynamically selected strong features during decoding. Our work is distinguished because we do not run multiple training phases as they do for figuring our strong features.

3 Dynamic Feature Induction

The intuition behind dynamic feature induction is to keep populating high dimensional features by joining low dimensional features together until the feature space becomes 'more' linearly separable.[2] Figure 1 shows how features are induced during training:

1. Given a training instance (x_1, y_1), where x_1 is a feature set consisting of 5 features and y_1 is the gold label, the classifier predicts the label $\hat{y_1}$.

2. Let us refer "strong features for y against $\hat{y}$" to features that give strong clues for distinguishing y from $\hat{y}$. If $\hat{y_1}$ is not equal to y_1 (2.1), strong features for y_1 against $\hat{y_1}$ in x_1 are selected (2.2), and combinations of these features are added to the induced feature set $\mathcal{F}$ (2.3).

3. Given a new training instance (x_2, y_2), combinations of features in x_2 are checked by $\mathcal{F}$ (3.1), and appended to x_2 if allowed (3.2).

[1] The joint features tend to yield a much higher dimensional feature space than the primitive features.

[2] The term 'more' is used because dynamic feature induction does not guarantee for the feature space to be linearly separable.

4. The extended feature set x_2 is fed into the classifier. If $\hat{y}_2$ is equal to y_2, no feature combination is induced from x_2.

Thus, high dimensional features in $\mathcal{F}$ are incrementally induced and learned along with low dimensional features during training. During decoding, each feature set is extended by the induced features in $\mathcal{F}$, and the prediction is made using the extended feature set. The size of $\mathcal{F}$ can grow up to $|\mathcal{X}|^2$, where $|\mathcal{X}|$ is the size of low dimensional features. However, we found that $|\mathcal{F}|$ is more like $1/4 \cdot |\mathcal{X}|$ in practice.

The following sections explain our approach in details. Sections 3.1, 3.2, and 3.3 describe how features are induced and learned during training. Sections 3.4 and 3.5 describe how the induced features are stored and expanded during decoding.

3.1 Feature Induction

Algorithm 1 shows an online learning algorithm that induces and learns high dimensional features during training. It takes the set of training instances D and the learning rate η, and returns the weight vector $\mathbf{w}$ and the set of induced features $\mathcal{F}$.

Algorithm 1 Feature Induction

Input: D: training set, η: learning rate.
Output: $\mathbf{w}$: weight vector, $\mathcal{F}$: induced feature set.

1: $\mathbf{w} \leftarrow \mathbf{g} \leftarrow 0$
2: $\mathcal{F} \leftarrow \varnothing$
3: **until** max epoch is reached **do**
4: **foreach** $(x, y) \in D$ **do**
5: $\hat{y} \leftarrow \arg\max_{y' \in \mathcal{Y}} (\mathbf{w} \cdot \phi(x, y', \mathcal{F}) - I_y(y'))$
6: **if** $y \neq \hat{y}$ **then**
7: $\partial \leftarrow \phi(x, y, \mathcal{F}) - \phi(x, \hat{y}, \mathcal{F})$
8: $\mathbf{g} \leftarrow \mathbf{g} + \partial \circ \partial$
9: $\mathbf{w} \leftarrow \mathbf{w} + (\eta / (\rho + \sqrt{\mathbf{g}})) \cdot \partial$
10: $\mathbf{v} \leftarrow [\mathbf{w} \circ \phi(x, y, \varnothing)]_y - [\mathbf{w} \circ \phi(x, \hat{y}, \varnothing)]_{\hat{y}}$
11: $\mathcal{L} \leftarrow \arg k \max_{\forall i} \mathbf{v}_i$
12: **for** $i = 2$ **to** $|\mathcal{L}|$ **do**
13: $\mathcal{F} \leftarrow \mathcal{F} \cup \{(\mathcal{L}_1, \mathcal{L}_i)\}$
14: **return** $\mathbf{w}, \mathcal{F}$

The algorithm begins by initializing the weight vector $\mathbf{w}$, the diagonal vector $\mathbf{g}$, and the induced feature set $\mathcal{F}$ (lines 1-2). For each instance $(x, y) \in D$ where y is the gold-label for the feature set x, it predicts $\hat{y}$ maximizing $\mathbf{w} \cdot \phi(x, y', \mathcal{F}) - I_y(y')$, where I is defined as follows (lines 4-5):

$$I_y(y') \leftarrow \begin{cases} 1, & \text{if } y = y'. \\ 0, & \text{otherwise.} \end{cases}$$

The feature map ϕ takes $(x, y, \mathcal{F})$, and returns a $d \times l$-dimensional vector, where d and l are the sizes of features and labels, respectively; each dimension contains the value for a particular feature and a label.[3] If certain combinations between features in x exist in $\mathcal{F}$, they are appended to the feature vector along with the low dimensional features (see Section 3.5 for more details). The indicator function I allows our algorithm to be optimized for the hinge loss for multiclass classification (Crammer and Singer, 2002):

$$\ell_h = \max[0, 1 + \mathbf{w} \cdot (\phi(x, \hat{y}, \mathcal{F}) - \phi(x, y, \mathcal{F}))]$$

If y is not equal to $\hat{y}$ (line 6), the partial vector ∂ is measured (line 7), and $\mathbf{g}$ and $\mathbf{w}$ are updated (lines 8-9) by AdaGrad (Duchi et al., 2011), where the learning rate η is adjusted by $\mathbf{g}$ (in our case, ρ =1E-5). Once $\mathbf{w}$ is updated, the d-dimensional vector $\mathbf{v}$ is generated by subtracting $[\mathbf{w} \circ \phi(x, \hat{y}, \varnothing)]_{\hat{y}}$ from $[\mathbf{w} \circ \phi(x, y, \varnothing)]_y$ (line 10), where $[\ldots]_y$ returns only the portion of the values relevant to y (Figure 2).

The i'th element in $\mathbf{v}$ represents the strength of the i'th feature for y against $\hat{y}$; the greater $\mathbf{v}_i$ is, the stronger the i'th feature is. Next, indices of the top-k entries in $\mathbf{v}$ are collected in the ordered list $\mathcal{L}$ (line 11), representing the strongest features for y against $\hat{y}$.[4] Finally, the pairs of the first index in $\mathcal{L}$, representing the strongest feature, and the other indices in $\mathcal{L}$ are added to the induced feature set $\mathcal{F}$ (lines 12-13). For example, if $\mathcal{L} = [i, j, k]$ such that $\mathbf{v}_i \geq \mathbf{v}_j \geq \mathbf{v}_k > 0$, two pairs, (i, j) and (i, k), are added to $\mathcal{F}$.

For all our experiments, $k = 3$ is used; increasing k beyond this cutoff did not show much improvement. Notice that all induced features in $\mathcal{F}$ are derived by joining only low dimensional features together. Our algorithm does not join a high dimensional feature with either a low dimensional feature or another high dimensional feature. This was done intentionally to prevent from the feature space being exploded; such features can be induced by replacing $\varnothing$ with $\mathcal{F}$ in the line 10 as follows:

$$\mathbf{v} \leftarrow [\mathbf{w} \circ \phi(x, y, \mathcal{F})]_y - [\mathbf{w} \circ \phi(x, \hat{y}, \mathcal{F})]_{\hat{y}}$$

[3] In most cases, these values are either 0 or 1.

[4] '$\arg k \max$' returns the ordered list of indices whose values in $\mathbf{v}$ are $^{1)}k$-largest and $^{2)}$greater than 0.

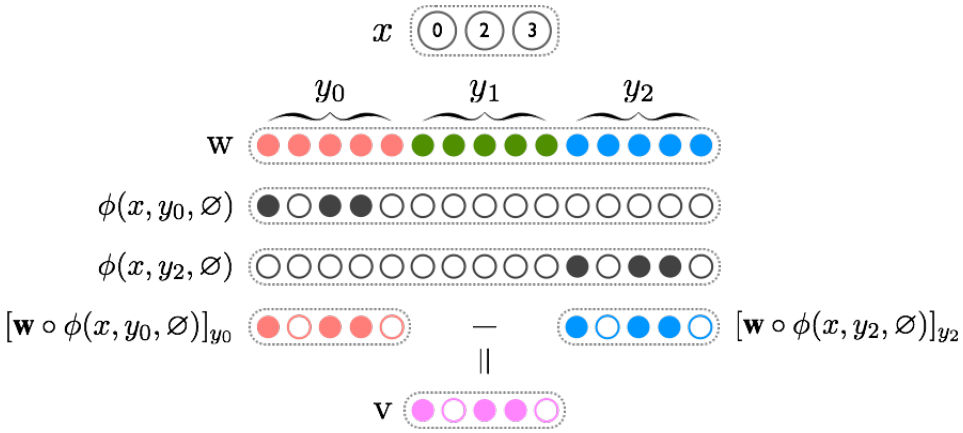

Figure 2: Given the weight vector $\mathbf{w}$ and the feature map ϕ, $[\mathbf{w} \circ \phi(x, y, \varnothing)]_y$ takes the Hadamard product between $\mathbf{w}$ and $\phi(x, y, \varnothing)$, then truncates the resulting vector with respect to the label y.

It is worth mentioning that we did not find it useful for joining intermediate features together (e.g., (j, k) in the above example). It is possible to utilize these combinations by weighting them differently, which we will explore in the future. Additionally, we experimented with the combinations between strong and weak features (joining i'th and j'th features, where $\mathbf{v}_i > 0$ and $\mathbf{v}_j < 0$), which again was not so useful. We are planning to evaluate our approach on more tasks and data, which will give us better understanding of what combinations are the most effective.

3.2 Regularized Dual Averaging

Each high dimensional feature in $\mathcal{F}$ is induced for making classification between two labels, y and $\hat{y}$, but it may or may not be helpful for distinguishing labels other than those two. Our algorithm can be modified to learn the weights of the induced features only for their relevant labels by adding the label information to $\mathcal{F}$, which would change the line 13 in Algorithm 1 as follows:

$$\mathcal{F} \leftarrow \mathcal{F} \cup \{(\mathcal{L}_1, \mathcal{L}_i, y, \hat{y})\}$$

However, introducing features targeting specific label pairs potentially confuses the classifier, especially when they are trained with the low dimensional features targeting all labels. Instead, it is better to apply a feature selection technique such as ℓ_1 regularization so the induced features can be selectively learned for labels that find those features useful. We adapt regularized dual averaging (Xiao, 2010), which efficiently finds the convergence rates for online convex optimization, and works most effectively with sparse feature vectors. To apply regularized dual averaging, the line 1 in Algorithm 1 is changed to:

$$\mathbf{w} \leftarrow \mathbf{g} \leftarrow \mathbf{c} \leftarrow 0; \;\; t \leftarrow 1$$

$\mathbf{c}$ is a $d \times l$-dimensional vector consisting of accumulative penalties. t is the number of weight vectors generated during training. Although $\mathbf{w}$ is technically not updated when $y = \hat{y}$, it is still considered a new vector. Thus, t is incremented for every training instance, so $t \leftarrow t + 1$ is inserted after the line 5. $\mathbf{c}$ is updated by adding the partial vector ∂ as follows (to be inserted after the line 7):

$$\mathbf{c} \leftarrow \mathbf{c} + \partial$$

Thus, each dimension in $\mathbf{c}$ represents the accumulative penalty (or reward) for a particular feature and a label. At last, the line 9 is changed to:

$$\mathbf{w} \leftarrow (\eta/(\rho + \sqrt{\mathbf{g}})) \cdot \ell_1(\mathbf{c}, t, \lambda)$$

$$\ell_1(\mathbf{c}, t, \lambda) \leftarrow \begin{cases} \mathbf{c}_i - \mathrm{sgn}(\mathbf{c}_i) \cdot \lambda \cdot t, & |\mathbf{c}_{\forall i}| > \lambda \cdot t. \\ 0, & \text{otherwise.} \end{cases}$$

The function ℓ_1 takes $\mathbf{c}$, t, and the regularizer parameter λ tuned during development. If the absolute value of the accumulative penalty $\mathbf{c}_i$ is greater than $\lambda \cdot t$, the weight $\mathbf{w}_i$ is updated by λ and t; otherwise, it is assigned to 0. For our experiments, RDA was able to throw out irrelevant features successfully, and showed improvement in accuracy; in fact, dynamic feature induction without RDA did not show as much improvement over low dimensional features.

3.3 Locally Optimal Learning to Search

Features in most NLP tasks are extracted from structures (e.g., sequence, tree). For structured learning, we adapt "locally optimal learning to search" (Chang et al., 2015b), that is a member of imitation learning similar to DAGGER (Ross et al., 2011). LOLS not only performs well relative to the reference policy, but also can improve upon the reference policy, showing very good results for tasks such as part-of-speech tagging and dependency parsing. We adapt LOLS by setting the reference policy as follows:

1. The reference policy π determines how often the gold label y is picked over the predicted label $\hat{y}$ to build a structure. For all our experiments, π is initialized to 0.95.
2. For the first epoch, since π is 0.95, y is randomly picked over $\hat{y}$ for 95% of the time.
3. After every epoch, π is multiplied by 0.95. This allows the next epoch to pick y less often than the previous epoch (e.g., π becomes $0.95^2 = 0.9025$ for the 2nd epoch so y is picked about 90% of the time instead of 95%).

For our experiments, LOLS gave only marginal improvement, probably because the tasks we evaluated, part-of-speech tagging and named entity recognition, did not yield complex structures. However, we still included this in our framework because we wanted to evaluate our approach on more tasks such as dependency parsing where learning to search algorithms show a clear advantage (Goldberg and Nivre, 2012; Choi and McCallum, 2013; Chang et al., 2015a).

3.4 Feature Hashing

Feature hashing is a technique of converting string features to vectors (Ganchev and Dredze, 2008; Weinberger et al., 2009). Given a string feature f and a hash function h, the index of f in the vector space is determined by taking the remainder of the hash code:

$$k \leftarrow h_{\text{string}\rightarrow\text{int}}(f) \textbf{ mod } \delta$$

The divisor δ is tuned during development. Feature hashing allows to convert string features into sparse vectors without reserving an extra space for a map whose keys and values are the string features and their indices. Given a feature index pair (i, j) representing strong features for y against $\hat{y}$ (Section 3.1), the index of the induced feature can be measured as follows:

$$k \leftarrow h_{\text{int}\rightarrow\text{int}}(i \cdot |\mathcal{X}| + j) \textbf{ mod } \delta$$

For efficiency, feature hashing is adapted to our system such that the induced feature set $\mathcal{F}$ is actually not a set but a δ-dimensional boolean array, where each dimension represents the validity of the corresponding induced feature. Thus, the line 13 in Algorithm 1 is changed to:

$$k \leftarrow h_{\text{int}\rightarrow\text{int}}(\mathcal{L}_1 \cdot |\mathcal{X}| + \mathcal{L}_i) \textbf{ mod } \delta$$
$$\mathcal{F}_k \leftarrow \texttt{True}$$

For the choice of h, `xxHash` is used, that is a fast non-cryptographic hash algorithm showing the perfect score on the Q.Score.[5]

3.5 Feature Expansion

Algorithm 2 describes how high dimensional features are expanded from low dimensional features during training and decoding. It takes the sparse vector $\mathbf{x}^l$ containing only low dimensional features and returns a new sparse vector $\mathbf{x}^{l+h}$ containing both low and high dimensional features.

Algorithm 2 Feature Expansion

Input: $\mathbf{x}^l$: sparse feature vector containing only low dimensional features.
Output: $\mathbf{x}^{l+h}$: sparse feature vector containing both low and high dimensional features.

1: $\mathbf{x}^{l+h} \leftarrow \text{copy}(\mathbf{x}^l)$
2: **for** $i \leftarrow 1$ **to** $|\mathbf{x}^l|$ **do**
3: **for** $j \leftarrow i+1$ **to** $|\mathbf{x}^l|$ **do**
4: $k \leftarrow h_{\text{int}\rightarrow\text{int}}(i \cdot |\mathcal{X}| + j)$ **mod** δ
5: **if** $\mathcal{F}_k$ **then** $\mathbf{x}^{l+h}$.append(k)
6: **return** $\mathbf{x}^{l+h}$

The algorithm begins by copying $\mathbf{x}^l$ to $\mathbf{x}^{l+h}$ (line 1). For every combination $(i, j) \in \mathbf{x}^l \times \mathbf{x}^l$, where i and j represent the corresponding feature indices (lines 2-3), it first measures the index k of the feature combination (line 4), then checks if this combination is valid (Section 3.4). If the combination is valid, meaning that ($\mathcal{F}_k = \texttt{True}$), k is added to $\mathbf{x}^{l+h}$ (line 5). Finally, $\mathbf{x}^{l+h}$ is returned with the expanded high dimensional features.

[5] https://github.com/Cyan4973/xxHash

4 Part-of-Speech Tagging

4.1 Corpus

The Wall Street Journal corpus from the Penn Treebank III is used (Marcus et al., 1993) with the standard split for part-of-speech tagging experiments.

Set	Sections	Sentences	ALL	OOV
TRN	0-18	38,219	912,344	0
DEV	19-21	5,527	131,768	4,467
TST	22-24	5,462	129,654	3,649

Table 1: Distributions of the Wall Street Journal corpus. TRN: training, DEV: development, TST: evaluation, ALL: all words, OOV: out-of-vocabulary words.

4.2 Tagging and Learning Algorithms

A one-pass, left-to-right tagging algorithm is used for our experiments. Such a simple algorithm is chosen because we want to see the performance gain purely from our approach, not by a more sophisticated tagging algorithm (Toutanova et al., 2003; Shen et al., 2007), which may improve the performance further.

For learning, the final algorithm from Section 3 is used. Additionally, mini-batch is applied, where each batch consists of training instances from k-number of sentences, causing the sizes of these batches different. We found that grouping instances with respect to the sentence boundary was more effective than batching them across arbitrary sentences. For all our experiments, the learning rate $\eta = 0.02$ and the mini-batch boundary $k = 5$ were used without tuning.

4.3 Ambiguity Classes

The ambiguity class of a word is the concatenation of all possible tags for that word. For example, if the word 'study' can be tagged by NN (common noun) or VB (base verb), its ambiguity class becomes NN_VB. Instead of building ambiguity classes only from the training dataset, we automatically tagged a mixture of large datasets, the English Wikipedia articles[6] and the New York Times corpus,[7] and pre-constructed ambiguity classes using the automatic tags before training. This was motivated by Moore (2015), who showed extraordinary results on the out-of-vocabulary words by limiting the classification to the ambiguity classes collected from such large corpora.

[6] dumps.wikimedia.org/enwiki

[7] catalog.ldc.upenn.edu/LDC2008T19

We used the ClearNLP POS tagger (Choi and Palmer, 2012) for tagging the data (about 141M words), threw away tags appearing less than a certain threshold, and created the ambiguity classes. For each word, tags appearing less than 20% of the time for that word were discarded. As the result, about 2M ambiguity classes were collected from these datasets.

4.4 Feature Template

Table 2 shows the template for low dimensional features. Digits inside the curly brackets imply the context windows with respect to the word w_i to be tagged. For example, $f_{\{0,\pm1\}}$ represents the word-forms of w_i, w_{i-1}, and w_{i+1}. No joint features (e.g., f_0-f_1) are included in this template; they should be automatically induced by dynamic feature induction.

Orthographic (Giménez and Màrquez, 2004) and word shape (Finkel et al., 2005) features are adapted from the previous work. The positional features indicate whether w_i is the first or the last word in the sentence. Word clusters are trained on the same datasets in Section 4.3 using Brown et al. (1992).

$f_{:\{0,\pm1,\pm2\}}, f_{u:\{0,\pm1,\pm2\}}, s_{:\{0,\pm1\}}, c_{:\{0,\pm1\}},$
$\pi_{2:\{0\}}, \pi_{3:\{0\}}, \sigma_{1:\{0\}}, \sigma_{2:\{0\}}, \sigma_{3:\{0\}}, \sigma_{4:\{0\}},$
$p_{:\{0,-1,-2,-3\}}, a_{:\{0,1,2,3\}}, \mathcal{O}_{:\{0\}}, \mathcal{P}_{:\{0\}}$

Table 2: Feature template for part-of-speech tagging. f: word-form, f_u: uncapitalized word-form, s: word shape, c: word cluster, π_k: k'th prefix, σ_k: k'th suffix, p: part-of-speech tag, a: ambiguity class, $\mathcal{O}$: orthographic feature set, $\mathcal{P}$: positional feature set.

4.5 Development

The regularization parameter λ (Section 3.2) and the modulo divisor δ (Section 3.4) are tuned during development through grid search on $\lambda \in$ [1E-9, 1E-6] and $\delta \in$ [1.5M, 5M]. Table 3 shows the accuracies achieved by our models on the development set.

Model	ALL	OOV	FEAT
M_0: baseline	97.09	86.14	365,400
M_1: M_0 + ext. ambi.	97.37	91.92	365,409
M_2: M_1 + clusters	97.45	91.96	372,181
M_3: M_1 + dynamic	97.42	92.10	468,378
M_4: M_2 + dynamic	97.48	92.21	473,134

Table 3: Part-of-speech tagging accuracies on the development set. FEAT: the number of features generated by each model.

M_0 used the tagging and the learning algorithms in Section 4.2 and the feature template in Section 4.4, where the ambiguity classes were collected only from the training dataset; dynamic feature induction was not used for M_0. By applying the external ambiguity classes in Section 4.3, M_1 achieved about a 5.8% improvement on OOV. M_2 gained small improvements by adding word clusters. Coupled with dynamic feature induction, M_3 and M_4 gained about 0.04% and 0.2% improvements on average for ALL and OOV.

For both M_3 and M_4, about 100K more features were generated from M_1 and M_2, implying that about 25% of the features were automatically induced by dynamic feature induction. It is worth pointing out that improving upon M_1 was a difficult task because it was already reaching near the state-of-the-art. The external ambiguity classes by themselves were strong enough to make accurate predictions such that the induced features did not find a critical role in the classification.

4.6 Evaluation

Table 4 shows the accuracies achieved by the models from Section 4.5 and the previous state-of-the-art approaches on the evaluation set.

Approach	ALL	OOV	EXT
Manning (2011)	97.29	89.70	
Manning (2011)	97.32	90.79	✓
Shen et al. (2007)	97.33	89.61	
Sun (2014)	97.36	-	
Moore (2015)	97.36	91.09	✓
Spoustová et al. (2009)	97.44	-	✓
Søgaard (2011)	97.50	-	✓
Tsuboi (2014)	97.51	91.64	✓
This work: M_0	97.18	86.35	
This work: M_1	97.37	91.34	✓
This work: M_2	97.46	91.23	✓
This work: M_3	97.52	91.53	✓
This work: M_4	**97.64**	**92.03**	✓

Table 4: Part-of-speech tagging accuracies on the evaluation set. EXT: whether or not the approach used external data.

The results on the evaluation set appear much more promising. Still, the biggest gain was made by M_1, but our final model M_4 was able to achieve a 0.8% improvement on OOV over M_2, and showed the state-of-the-art results on both ALL and OOV. Interestingly, M_2 showed a slightly lower accuracy on OOV than M_1 even with the additional word cluster features. On the other hand, M_2 did show a slightly higher accuracy on ALL, indicating that the model was probably too overfitted to the in-vocabulary words.[8] However, M_4 was still able to achieve improvements over M_2 on both ALL and OOV, implying that dynamic feature induction facilitated the classifier to be trained more robustly.

5 Named Entity Recognition

5.1 Corpus

The English corpus from the CoNLL'03 shared task is used (Tjong Kim Sang and De Meulder, 2003) for named entity recognition experiments.

Set	Articles	Sentences	Words
TRN	946	14,987	203,621
DEV	216	3,466	51,362
TST	231	3,684	46,435

Table 5: Distributions of the English corpus from the CoNLL'03 shared task. TRN: training, DEV: development, TST: evaluation.

5.2 Feature Template

Table 6 shows the feature template for NER, adapting the specifications in Table 2. Following the state-of-the-art approaches (Table 8), word clusters are trained on the Reuters Corpus Volume I (Lewis et al., 2004) using Brown et al. (1992). Named entity gazetteers are collected from DBPedia.[9] Word embeddings are trained on the datasets in Section 4.3 using Mikolov et al. (2013) and appended to the sparse feature vectors as dense vectors. Note that the word embedding features did not participate in dynamic feature induction; it was not intuitive how to combine sparse and dense features together so we left it as a future work.

$f_{:\{0,\pm1\}}, f_{u:\{0,\pm1,\pm2\}}, s_{:\{0,\pm1\}}, l_{:\{0\}}, c_{:\{0,1,2\}},$
$e_{:\{0,\pm1,\pm2,\pm3,\pm4\}}, \pi_{1:\{0\}}, \pi_{3:\{1\}}, \sigma_{1:\{0\}}, \sigma_{3:\{-1,0\}},$
$p_{:\{0,\pm1,\pm2\}}, n_{:\{-1,-2,-3\}}, z_{:\{\pm1,0,2,3\}}, \mathcal{O}_{:\{0\}}, \mathcal{O}_{:\{1\}}$

Table 6: Feature template for named entity recognition. f: word-form, f_u: uncapitalized word-form, s: word shape, l: lemma, c: word cluster, e: word embedding, π_k: k'th prefix, σ_k: k'th suffix, p: part-of-speech tag, n: named entity tag, z: named entity gazetteer, $\mathcal{O}$: orthographic feature set.

[8] A similar trend is shown in Table 3 for M_1 and M_2.

[9] wiki.dbpedia.org/downloads2015-04

5.3 Development

The regularization parameter and the modulo divisor are tuned during development through the same grid search in Section 4.5. Table 7 shows the precisions and the recalls achieved by our models on the development set (the F1-scores are shown in Table 8).

Model	P	R	FEAT
M_0: baseline	90.87	89.15	164,440
M_1: M_0 + gazetteers	92.30	90.61	164,720
M_2: M_1 + clusters	93.66	91.79	169,232
M_3: M_2 + embeddings	94.14	92.43	169,682
M_4: M_3 + dynamic	94.50	93.10	208,860

Table 7: Precision and recall on the development set for named entity recognition. P: precision, R: recall.

M_0 used the tagging and the learning algorithms in Section 4.2 and the feature template in Section 5.2, excluding the gazetteer, cluster, and embedding features; dynamic feature induction was not applied to M_0. $M_{\{1,2,3\}}$ gained incremental improvements from the gazetteer, cluster, and embedding features, respectively. M_4 showed 0.36% and 0.67% improvements on precision and recall respectively, and generated about 40K more features compared to M_3. This is about 23% increase in features that is similar to the increase shown in Table 3.

5.4 Evaluation

Table 8 shows the F1-scores achieved by our models and the previous state-of-the-art approaches.[10]

Approach	DEV	TST
Turian et al. (2010)	93.25	89.41
Suzuki and Isozaki (2008)	**94.48**	89.92
Ratinov and Roth (2009)	93.50	90.57
Lin and Wu (2009)	-	90.90
Passos et al. (2014)	94.46	90.90
This work: M_0	90.00	84.44
This work: M_1	91.45	86.85
This work: M_2	92.72	89.64
This work: M_3	93.27	90.57
This work: M_4	93.79	**91.00**

Table 8: F1-scores on the development and the evaluation sets for named entity recognition.

[10]Ratinov and Roth (2009) reported the F1-score of 90.80 on the evaluation set, but that model was trained on both the training and the development sets so not compared in this table.

All models showed improvements over their predecessors; the improvements made in TST were more dramatic than the ones made in DEV although they followed a very similar trend. Notice that M_3, not using dynamic feature induction, showed very similar scores to Ratinov and Roth (2009). This was not surprising because M_3 adapted many features suggested by them, except for the non-local features.[11]

M_4 achieved about 0.5% improvements over M_3, showing the state-of-the-art result on TST. Considering that M_3 was already near state-of-the-art, this improvement was meaningful. It was interesting that Suzuki and Isozaki (2008) achieved the state-of-the-art result on DEV although their score on TST was much lower than the other approaches. This might be because features extracted from the huge external data they used were overfitted to DEV, but more thorough analysis needs to be done. On the other hand, Passos et al. (2014) achieved the near state-of-the-art result on DEV while it also got a very high score on TST by utilizing phrase embeddings, which we will look into in the future.

6 Conclusion

In this paper, we introduced a novel technique called dynamic feature induction that automatically induces high dimensional features so the feature space can be more linearly separable. Our approach was evaluated on two NLP tasks, part-of-speech tagging and named entity recognition, and showed the state-of-the-art results on both tasks. The improvements achieved by dynamic feature induction might not be statistically significant, but important because they gave the last gist to the state-of-the-art; without this last gist, our system would have not reached the bar.

It is worth mentioning that we also experimented with several feature templates including many joint features without applying dynamic feature induction. The results we got from these manually induced features were not any better (often worse) than the ones achieved by dynamic feature induction, which was very encouraging. In the future, we will experiment our approach on more NLP tasks such as dependency parsing and conference resolution where induced features should play a more critical role.

[11]We transformed the original data into the BILOU notation, which was also suggested by Ratinov and Roth (2009).

We concede that our approach is more empirically motivated than theoretically justified. For instance, the choice of k (line 11) or the combination configuration for $\mathcal{L}$ (line 13) in Algorithm 1 are rather empirically derived. All the parameters are automatically tuned by running grid searches on the development sets (Sections 4.5 and 5.3); it would be intellectually intriguing to find a more principled way of adjusting these hyper-parameters than just brute-force search.

The locally optimal learning to search is used to help structured learning although it gives a relatively smaller impact to the tasks involving sequence classification such as part-of-speech tagging and named entity recognition. This framework is used because we plan to apply our approach on more structurally oriented tasks such as dependency parsing and AMR parsing. Our work is also related to feature grouping, which has been shown to be beneficial in learning high-dimensional data (Zhong and Kwok, 2011; Suzuki and Nagata, 2013). It will be interesting to compare our work to the previous work and see the strengths and weaknesses of our approach.

Acknowledgments

We gratefully acknowledge the support of the Yahoo Academic Career Enhancement Award, the IPsoft Development Enhancement Grant, the University Research Committee Award, and the Infosys Research Enhancement Grant. Any contents expressed in this material are those of the authors and do not necessarily reflect the views of these awards and grants.

References

Eric Bengtson and Dan Roth. 2008. Understanding the Value of Features for Coreference Resolution. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, EMNLP'08, pages 294–303.

Peter F. Brown, Peter V. deSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. 1992. Class-based n-gram Models of Natural Language. *Computational Linguistics*, 18(4):467–480.

Kai-Wei Chang, He He, Hal Daumé III, and John Langford. 2015a. Learning to Search for Dependencies. *arXiv:1503.05615*.

Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daume, and John Langford. 2015b. Learning to Search Better than Your Teacher. In *Proceedings of the 32nd International Conference on Machine Learning*, ICML'15, pages 2058–2066.

Jinho D. Choi and Andrew McCallum. 2013. Transition-based Dependency Parsing with Selectional Branching. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, ACL'13, pages 1052–1062.

Jinho D. Choi and Martha Palmer. 2012. Fast and Robust Part-of-Speech Tagging Using Dynamic Model Selection. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, ACL'12, pages 363–367.

Michael Collins. 2002. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *Proceedings of the conference on Empirical methods in natural language processing*, EMNLP'02, pages 1–8.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural Language Processing (Almost) from Scratch. *Journal of Machine Learning Research*, 12:2493–2537.

Koby Crammer and Yoram Singer. 2002. On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines. *Journal of Machine Learning Research*, 2:265–292.

Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard Schwartz, and John Makhoul. 2014. Fast and Robust Neural Network Joint Models for Statistical Machine Translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, ACL'14, pages 1370–1380.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *The Journal of Machine Learning Research*, 12(39):2121–2159.

Jenny Finkel, Shipra Dingare, Christopher Manning, Malvina Nissim, and Beatrice Alex. 2005. Exploring the Boundaries: Gene and Protein Identification in Biomedical Text. *BMC Bioinformatics*, 6:S5.

Kuzman Ganchev and Mark Dredze. 2008. Small Statistical Models by Random Feature Mixing. In *Proceedings of the ACL Workshop on Mobile NLP*, pages 604–613.

Jesús Giménez and Lluís Màrquez. 2004. SVMTool: A general POS tagger generator based on Support Vector Machines. In *Proceedings of the 4th International Conference on Language Resources and Evaluation*, LREC'04.

Yoav Goldberg and Michael Elhadad. 2008. splitSVM: Fast, Space-Efficient, non-Heuristic, Polynomial Kernel Computation for NLP Applications. In *Proceedings of the Annual Conference of the Association for Computational Linguistics*, ACL:HLT'08, pages 237–240.

Yoav Goldberg and Joakim Nivre. 2012. A Dynamic Oracle for Arc-Eager Dependency Parsing. In *Proceedings*

of the 24th International Conference on Computational Linguistics, COLING'12.

Taku Kudo and Yuji Matsumoto. 2003. Fast Methods for Kernel-Based Text Analysis. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, ACL'04, pages 24–31.

Quoc V. Le and Tomas Mikolov. 2014. Distributed Representations of Sentences and Documents. In *Proceedings of the 31th International Conference on Machine Learning*, ICML'14, pages 1188–1196.

David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. 2004. RCV1: A New Benchmark Collection for Text Categorization Research. *Journal of Machine Learning Research*, 5:361–397.

Dekang Lin and Xiaoyun Wu. 2009. Phrase Clustering for Discriminative Learning. In *Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics*, ACL'09, pages 1030–1038.

Christopher D. Manning. 2011. Part-of-Speech Tagging from 97% to 100%: Is It Time for Some Linguistics? In *Proceedings of the 12th international conference on Computational linguistics and intelligent text processing*, CICLing'11, pages 171–189.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeff Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *arXiv:1301.3781*.

Robert Moore. 2015. An Improved Tag Dictionary for Faster Part-of-Speech Tagging. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP'15, pages 1303–1308.

Daisuke Okanohara and Jun'ichi Tsujii. 2009. Learning Combination Features with L1 Regularization. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, NAACL'09, pages 97–100.

Alexandre Passos, Vineet Kumar, and Andrew McCallum. 2014. Lexicon Infused Phrase Embeddings for Named Entity Resolution. In *Proceedings of the 18th Conference on Computational Natural Language Learning*, CoNLL'14, pages 78–86.

Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Meichun Hsu. 2001. PrefixSpan: Mining Sequential Patterns by Prefix-Projected Growth. In *Proceedings of the 17th International Conference on Data Engineering*, pages 215–224.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, EMNLP'14, pages 1532–1543.

Simon Perkins, Kevin Lacker, and James Theiler. 2003. Grafting: Fast, Incremental Feature Selection by Gradient Descent in Function Space. *Journal of Machine Learning Research*, 3:1333–1356.

Sameer Pradhan, Kadri Hacioglu, Valerie Krugler, Wayne Ward, James H. Martin, and Daniel Jurafsky. 2005. Support Vector Learning for Semantic Argument Classification. *Machine Learning*, 60(1):11–39.

Lev Ratinov and Dan Roth. 2009. Design Challenges and Misconceptions in Named Entity Recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, CoNLL'09, pages 147–155.

Stéphane Ross, Geoffrey J. Gordon, and Drew Bagnell. 2011. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In *Proceedings of the Workshop on Artificial Intelligence and Statistics*, AI-STATS'11, pages 627–635.

Libin Shen, Giorgio Satta, and Aravind Joshi. 2007. Guided Learning for Bidirectional Sequence Classification. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, ACL'07, pages 760–767.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, EMNLP'13, pages 1631–1642.

Anders Søgaard. 2011. Semi-supervised condensed nearest neighbor for part-of-speech tagging. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, ACL:HLT'11, pages 48–52.

Drahomíra "johanka" Spoustová, Jan Hajič, Jan Raab, and Miroslav Spousta. 2009. Semi-supervised Training for the Averaged Perceptron POS Tagger. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, EACL'09, pages 763–771.

Emma Strubell, Luke Vilnis, Kate Silverstein, and Andrew McCallum. 2015. Learning Dynamic Feature Selection for Fast Sequential Prediction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*, ACL'16, pages 146–155.

Xu Sun. 2014. Structure Regularization for Structured Prediction. In *Proceedings of Advances in Neural Information Processing Systems.*, NIPS'14, pages 2402—2410.

Jun Suzuki and Hideki Isozaki. 2008. Semi-Supervised Sequential Labeling and Segmentation Using Giga-Word Scale Unlabeled Data. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, ACL:HLT'08, pages 665–673.

Jun Suzuki and Masaaki Nagata. 2013. Supervised Model Learning with Feature Grouping based on a Discrete Constraint. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 18–23.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 Shared Task: Language-independent Named Entity Recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*, CONLL '03, pages 142–147. Association for Computational Linguistics.

Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. 2003. Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In *Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, NAACL'03, pages 173–180.

Yuta Tsuboi. 2014. Neural Networks Leverage Corpus-wide Information for Part-of-speech Tagging. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP'14, pages 938–950.

Joseph Turian, Lev-Arie Ratinov, and Yoshua Bengio. 2010. Word Representations: A Simple and General Method for Semi-Supervised Learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL'10, pages 384–394.

Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. 2009. Feature Hashing for Large Scale Multitask Learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML'09, pages 1113–1120.

Lin Xiao. 2010. Dual Averaging Methods for Regularized Stochastic Learning and Online Optimization. *Journal of Machine Learning Research*, 11:2543–2596.

Nianwen Xue and Martha Palmer. 2004. Calibrating Features for Semantic Role Labeling. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP'04, pages 88–94.

Lei Yu, Karl Moritz Hermann, Phil Blunsom, and Stephen Pulman. 2014. Deep Learning for Answer Sentence Selection. In *Proceedings of the NIPS Deep Learning Workshop*.

Tong Zhang and David Johnson. 2003. A Robust Risk Minimization Based Named Entity Recognition System. In *Proceedings of the 7th Conference on Natural Language Learning*, CONLL'03, pages 204–207.

Yue Zhang and Joakim Nivre. 2011. Transition-based Dependency Parsing with Rich Non-local Features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, ACL'11, pages 188–193.

Wenliang Zhong and James Kwok. 2011. Efficient Sparse Modeling with Automatic Feature Grouping. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 9–16, New York, NY, USA, June. ACM.