Segmenting Web-Domains and Hashtags using Length Specific Models

Sriram Srinivasan
Yahoo! SDC
Bangalore
srsriram@yahoo-inc.com

Sourangshu Bhattacharya
Yahoo! Labs
Bangalore
sourangb@yahoo-inc.com

Rudrasis Chakraborty
Indian Statistical Institute
Kolkata
rudrasischa@ieee.org

ABSTRACT

Segmentation of a string of English language characters into a sequence of words has many applications. Here, we study two applications in the internet domain. First application is the web domain segmentation which is crucial for monetization of broken URLs. Secondly, we propose and study a novel application of twitter hashtag segmentation for increasing recall on twitter searches. Existing methods for word segmentation use unsupervised language models. We find that when using multiple corpora, the joint probability model from multiple corpora performs significantly better than the individual corpora. Motivated by this, we propose weighted joint probability model, with weights specific to each corpus. We propose to train the weights in a supervised manner using max-margin methods. The supervised probability models improve segmentation accuracy over joint probability models. Finally, we observe that length of segments is an important parameter for word segmentation, and incorporate length-specific weights into our model. The length specific models further improve segmentation accuracy over supervised probability models. For all models proposed here, inference problem can be solved using the dynamic programming algorithm.

We test our methods on five different datasets, two from web domains data, and three from news headlines data from an LDC dataset. The supervised length specific models show significant improvements over unsupervised single corpus and joint probability models. Cross-testing between the datasets confirm that supervised probability models trained on all datasets, and length specific models trained on news headlines data, generalize well. Segmentation of hashtags result in significant improvement in recall on searches for twitter trends.

Categories and Subject Descriptors

I.2.7 [**Computing Methodologies**]: Artificial IntelligenceNatural Language Processing

General Terms

Algorithms,Experimentation

Keywords

Web domain segmentation, Hashtag segmentation, Compound splitting, Word Segmentation, Structured Learning

1. INTRODUCTION

The structure of written English language, and many other languages, impose the assumption that words are separated by spaces. While this assumption is generally valid, there are many scenarios where space cannot be reliably used as a word delimiter. For example, in automatic speech transcription [14], the continuous stream of phonemes have to be delimited by word boundaries. In optical character recognition [1], word boundaries cannot be determined reliably. In machine translation [4], splitting of compound words can lead to simple words, which can be translated. In each of these cases, the input string needs to be *segmented* in order to determine the word boundaries, and hence the words.

In this paper, we discuss two web applications, where word boundaries cannot be determined using spaces or other word separators. The first application is *domain match* [11], an Internet service, which offers to display advertisements and search results on broken URLs, i.e. URLs which do not have any content. Domain match is one of the important products for many Internet / E-commerce companies, e.g. Google, Yahoo!, etc, both in terms of volume and revenue. For this product, the only information available is the URL. Hence, the words extracted from the URL are crucial for relevance ranking. A dominant component of the URL is the domain name, which is typically formed by concatenating words without delimiters, e.g., `www.newestcarsforsale.com`. Hence, we need to segment the domain name string, in order to extract words.

The second is a novel application proposed in this paper. We propose to improve search results on twitter, by segmenting hashtags and using the extracted words for enhancing the search. Twitter hashtags are mostly concatenation of words, preceded by a hash, e.g. *#stirringthepot*. There can be two scenarios where hashtags need to be segmented: the search string contains a hashtag or the hashtag is present in a tweet. In the former case, we replace the query with the phrase generated from the words extracted by segmenting the hashtag. This would result in matches from tweets, which do not contain the hashtag explicitly, but only the search phrase (advanced phrase matching can also be used). The second scenario is handled in the experiments in this paper. In the second scenario, we segment hashtags in all the tweets, and add the extracted terms to the tweet for the purpose of searching. Hence, when searched with a phrase, the search result will also contain tweets that have a corresponding hashtag but not the search phrase.

Word segmentation [14] or *compound splitting* [6] or *Word breaking* [16] is the problem of finding the best segmentation of an input string into a sequence of words. A typical method is to design a scoring function for segmentations, and return the highest scoring segmentation as the "best". The reason is that often, there are multiple "valid" segmentations of the input string, where all the token are from the standard dictionary. For example, for domains:

www.homesandgardens.com:
homes and gardens
home sand gardens
or for news headlines:
greekdeputyofferstoresign:
greek deputy offers to resign
greek deputy offer store sign
or for twitter hashtags:
#youdidthistoyourself:
you did this to yourself
you did this toy ours elf

Most of the existing word segmentation techniques for English (or other Indo-European languages) [16, 6, 4] are unsupervised. They assume a language model [14, 16], estimate the parameters for the language model, and use the language model to score various alternative segmentations. Finding the best segmentation of the string, involves doing maximum a posteriori (MAP) inference on the language model. Parameters for the language model are estimated from an appropriate corpus of the same language.

For the web domain, many different text corpora are available, each capturing different language styles. Wang et al. [16] use four different corpora: titles, anchor text, body, and query text, for domain segmentation. One can also imagine using wikipedia, and other user generated content, like comments, tweets, etc. Incorporating the information in multiple corpora is a crucial problem for word segmentation. Wang et al. [16] try each of the corpora in turn and also a mixture distribution of the corpora. They conclude that "titles" corpus, which matches the language style of the web domains, perform the best.

In this paper, we try a joint probability approach to capturing the information in multiple corpora. In particular, we assume that the language models from the different corpora are independent. Hence the joint probability of a segmentation is the product of probabilities obtained from individual corpora. Inference on the joint probability model can be performed using dynamic programming algorithm, which guarantees global optimum to be found. We find that the joint probability model performs significantly better than the individual models (see experimental results). Surprisingly, we did not find the joint probability approach (equation 4) in existing word segmentation literature.

Encouraged by this observation, we devised a weighted scoring function similar to the joint probability model, giving separate weights to each corpora. We propose to learn these weights through max-margin training paradigm[12], popular in machine learning. The max margin training algorithm only requires an implementation of inference algorithm.

Finally, we notice that there are many strings which are not segmented correctly by any of the corpora. These strings cannot be segmented correctly using any combination of corpus specific weights. We also observe that length of the segments play a crucial role in scoring its contribution towards the entire segmentation. Our final contribution here is to use length-specific weights for scoring word segmentations. In experiments, the length specific models are found to perform the best.

In order to assess the performance of a given model, we use the top-1 accuracy, which is the fraction of unsegmented strings for which the correct segmentation is retrieved as the highest scoring one. This metric is different from the ones used in existing papers. For example, Wang et al. [16], use precision at position 3 as their metric. This is because their dataset consists of multiple correct segmentations. Since we have only one correct segmentation for each string, precision and recall do not make sense. Also, our metric is also different from "word precision" metric used by Goldwater et al. [3] and Venkataraman [14]. The word precision counts the fraction of words identified correctly, as opposed to fraction of entire segmentations in top-1 accuracy.

We use the top-1 accuracy metric because, for the applications considered here, it is important to get the words corresponding to the correct segmentation only. For example, the words in the second best segmentation of the hashtag "*#youdidthistoyourself*", "*you did this toy ours elf*", may contain the words "*toy*" and "*elf*". Even though these are valid dictionary words, they are totally irrelevant to the context, and hence are liable to throw irrelevant search results. Hence, for our applications, the top-1 accuracy is most important.

In order to test the models developed here, we perform extensive experimentation on five different datasets: two from Yahoo! domains traffic and three from the publicly available LDC data [8]. Experimental results show that the length-specific model outperforms all other supervised and unsupervised models tested here. The improvement over best unsupervised model (the joint probability model) is $\sim 7\%$ for unigram models and $\sim 3.5\%$ for the bigram models. The joint probability model also improves over single corpus models by over 5% for unigram models and $\sim 2\%$ for bigram models on most datasets. Cross-testing on datasets other than the one trained on, reveals that probability supervised models generalize well, and length-specific models trained on news datasets generalize well. Experiments using twitter trends show significant improvement in recall, when words from segmented hashtags are also considered for searching.

1.1 Related Work

The problem of word segmentation has been studied in various contexts. Venkataraman [14] uses unsupervised word segmentation techniques for finding words in automatically transcribed speech. Recently, Macherey et al. [6] use unsupervised techniques for word segmentation, highlighting the application of this technique to multiple European languages. The work of Wang et al. [16] is the closest to our work. They use unsupervised techniques with multiple corpora for word segmentation. However, they try mixture models instead of the joint probability approach (see equation 4) taken here. They conclude that using one "matched" corpus performs better than the mixture model.

Most studies on word segmentation use unsupervised language models [16, 6, 14]. Conditional random fields (CRFs) [5], which are supervised techniques, are a popular tool for Chinese word segmentation[9]. However, CRFs as applied for Asian language word segmentation cannot be applied directly for English language word segmentation. This is because words in Asian languages have much lower lengths. Hence, capturing the dependency with the previous character is sufficient. Also, characters have a lot of meaningful features attached to them. In case of English language, defining features for words makes more sense. This means that Markov order of the CRF can be arbitrarily high, thus rendering inference infeasible.

Another prominent line of work pioneered by Goldwater et al. [3] uses cache models. They use Chinese restaurant process (CRP) to model the probability of a segmentation S given a base probability distribution P, of the word (unigram) occurrence. Let $G(S)$ be the probability of the segmentation S. The conditional distribution for the word $s_i,\ i = 1, \ldots, |S|$, given all other words s_{-i}, is given by:

$$P(s_i|s_{-i}) = \frac{n_{s_i}^{(s_{-i})}}{|s| - 1 + \alpha_0} + \frac{\alpha_0 * P(s_i)}{|s| - 1 + \alpha_0}$$

where $n_{s_i}^{(s_{-i})}$ is the number of instances of the word s_i in the remaining string s_{-i}, and α_0 is a parameter of the CRP.

Goldwater et al. also extend their models to bigrams using Hierarchical Dirichlet Processes (HDP). In HDP, given the previous word s_{i-1}, the next word s_i is distributed according to a *Dirichlet process* (DP) specific to s_{i-1}. All the word specific DPs are linked by the fact that they share a common base distribution G, which is generated from another DP. We implemented the above two models. The results are reported in section 4.1.5.

2. WORD SEGMENTATION

2.1 Preliminaries and Definitions

The problem of *word segmentation* [16, 3] is to find the best way of splitting an input string into words or segments. The task takes a sequence of characters, or a **string**, x from a finite alphabet Σ as input. We use $|x|$ to denote the length of input string. In this case Σ is the set of all 26 character in the English language. In theory, x can be arbitrarily long. However, the inference algorithm, and hence the training algorithm will scale as $|x|^2$ or worse. In this paper, we consider three types inputs: web domains, new headlines, and twitter hashtags, which are of length less than 100.

A **segmentation**, S, is a list of **segments** $s_i,\ i = 1, \ldots, |S|$, where $s_i = (t_i, u_i)$ where, t_i and u_i are starting and ending indices of s_i, in string x. Hence, $1 \leq t_i, u_i \leq |x|$ and $t_i = u_{i-1} + 1$. We consider only one correct segmentation for an input string, unlike multiple correct segmentations considered in [16]. As explained above, this is most suitable for the problems we consider here. Thus, in this paper, we consider datasets of form $\mathcal{D} = \{(x_j, S_j)|j = 1 \ldots n\}$.

As in the previous works [16, 14], we take a scoring function based approach towards word segmentation. First we define scoring functions which score various segmentations of a string. The best segmentation is found through an inference algorithm which uses dynamic programming to efficiently search through the space of all segmentations. Finally, we describe learning schemes which learn the parameters of the scoring function to suit the training data. The next section defines various scoring functions.

2.2 Unsupervised Segmentation and Scoring

The main components of a model for unsupervised segmentation are the *language model* and the *backoff function* (also known as smoothing). n-gram language models have been successfully used in many information retrieval tasks [7] and also in word segmentation[16, 14]. The simplest model is the unigram model which assigns a probability $P(S)$ to a sentence (segmentation in our case) S, in terms of probabilities $P(s_i)$ of the constituent words (segments) s_i.

$$P(S) = \prod_{i=1}^{|S|} P(s_i)$$

The unigram model assumes words to be independent. Hence they don't take context of a word into account. A natural extension to the unigram model is the bi-gram model [7], which takes the context of a word into account. The bigram model is given by:

$$P(S) = P(s_1) \prod_{i=2}^{|S|} P(s_i|s_{i-1})$$

where $P(s_i|s_{i-1})$ is the bigram probability of word s_i given the previous word is s_{i-1}. One can use trigram or higher order models. However, in many reported results for word segmentation (e.g. [14]), the improvement in accuracy with trigrams over bigrams is marginal. Moreover, the trigrams dataset does not fit into memory. Hence, one has to use a cache mechanism to perform inference with trigrams. This becomes prohibitive when used with the learning algorithm, which calls the inference algorithm many times, causing many cache misses. Also, it was observed in [15] that payoff of using higher order language models taper off after bigrams. Hence, we restrict our experiments to unigram and bigram models.

In the above formulation, we note that $P(S)$ does not depend on the unsegmented string x. As mentioned in [16], one way to handle this is to use the bayes rule to decompose $P(S|x) \propto P(x|S)P(S)$. Wang et al. [16] also propose to weigh the "transformation model" $P(x|S)$ with a coefficient α. Thus, their final scoring function becomes:

$$Sc(S) = \alpha \log(P(x|S)) + \log(P(S)) \qquad (1)$$

In our experiments, we observe that inclusion of the transformation model does not affect the accuracy of the segmentation much. A possible reason is that even though $P(S)$ scores all sentences, the inference algorithm only compares $P(S)$ for segmentations which arise from splitting x. This is also consistent with the results reported in [16].

The probability of segment s_i can be calculated by using frequency of occurrence of the corresponding word in a corpus. Let $C(s_i)$ be the count of number of times the word in segment s_i appears in a corpus. Also let T^1 be the total count of all words in the corpus. $P(s_i)$ can be estimated as $P(s_i) = \frac{C(s_i)}{T^1}$. However, this kind of scoring means that if any segment s_i in a segmentation is not present in the corpus, the probability of the entire segmentation becomes zero. This problem has been pointed out in the literature [15] as "out of vocabulary" (OOV) words. The problem becomes more acute when one tries to combine models from multiple corpora. Hence, a backoff or smoothing[2] technique is used to assign probabilities to unseen words. Such models are called "open vocabulary" language models [15].

Following [14], we use the Witten and bell backoff for assigning probabilities to out of vocabulary words (OOVs). The unigram backoff probability is given by:

$$P(s_i) = \begin{cases} \frac{C(s_i)}{N^1+T^1} & \text{if } C(s_i) > 0 \\ \frac{N^1}{N^1+T^1} P_\Sigma(s_i) & \text{otherwise} \end{cases}$$

where, N^1 is the number of unique unigrams, T^1 is the number of total unigrams, and $P_\Sigma(s_i) = \frac{P_\#(1-P_\#)^{|s_i|-1}}{|\Sigma|^{|s_i|}}$ is the probability a word of length $|s_i|$ appearing from alphabet of size $|\Sigma|$, and $P_\#$ being the probability of a word ending. $P_\#$ can be estimated from a training data, or from a corpus with language style similar to the segmentation problem. The bigram backoff probability is given by:

$$P(s_i|s_{i-1}) = \begin{cases} \frac{T^2}{N^2+T^2}\frac{C(s_{i-1}s_i)}{C(s_{i-1})} & \text{if } C(s_{i-1}s_i) > 0 \\ \frac{N^2}{N^2+T^2} P(s_i) & \text{otherwise} \end{cases}$$

We obtain the scoring functions for unsupervised probability models by taking log on both sides of the unigram and bigram probability models mentioned above. Thus the scoring functions for unsupervised unigram ($U1$) and bigram ($U2$) are given respectively as:

$$U1: \quad \log(P(S)) = \textstyle\sum_{i=1}^{|S|} \log(P(s_i)) \quad (2)$$

$$U2: \quad \log(P(S)) = \log(P(s_1)) + \textstyle\sum_{i=2}^{|S|} \log(P(s_i|s_{i-1})) \quad (3)$$

The main disadvantage of the unsupervised model described above is that it has only one parameter $P(s_i)$ to express the relative frequency of s_i. However, in practice, we have multiple corpora, each giving a different $P(s_i)$. In the next section, we extend the scoring functions given above to take probabilities from multiple corpora into account.

2.3 Multiple corpora and Supervised Probability Model

Let there be K corpora. Following the scheme described in the previous section, we can assign probabilities to each segment s_i, and for each of the K corpora. Let $(\log(P_1(s_i)), \ldots, \log(P_K(s_i)))$ be the log probabilities for segment s_i, from each of the corpora. The first idea is to build a language model from each of the corpora, and through validation on an annotated set, use the most "matching" corpus [16]. We call this the *single corpus* approach. In [16], Wang et al. show that the "titles" corpus matches best with the web domains.

Another unsupervised approach is to use the joint probability of a segmentation S belonging to all corpora as the score for the segmentation. Assuming independence between the corpora, the score for segmentation S becomes:

$$P(S) = \prod_{k=1}^{K} P_k(S) \quad (4)$$

We call this, the *joint probability model.* Taking log on both sides, we get the scoring function for unigram ($J1$) and bigram ($J2$) joint probability models as:

$$J1: \quad \log(P(S)) = \textstyle\sum_{k=1}^{K}\sum_{i=1}^{|S|} \log(P_k(s_i)) \quad (5)$$

$$J2: \quad \log(P(S)) = \textstyle\sum_{k=1}^{K} \left(\log(P_k(s_1)) + \sum_{i=2}^{|S|} \log(P_k(s_i|s_{i-1}))\right) (6)$$

We see in our experiments (section 4), that the joint probability model works better than the models generated using individual corpora. This motivates us to define the supervised probability models below.

Another method of combining the corpus specific probabilities is to construct a mixture model. This technique has been tried in [16]. The probability of segmentation S can be expressed as $P(S) = \sum_{k=1}^{K} w_k P_k(S)$. Substituting unigram probability model in this equation, we get:

$$P(S) = \sum_{k=1}^{K} w_k (\prod_{i=1}^{|S|} P(s_i))$$

Unfortunately, this scoring function is in the sum of products form, which cannot be reduced to a form where dynamic programming can be used to find the best segmentation. Wang et al. [16] use the beam search heuristic for finding best segmentation. Also, results in [16] show that this model does not perform better than the best matching single corpus model. Hence, we do not try the mixture models here.

Motivated by the success of the joint probability model, we define the supervised probability model as:

$$P(S) = \prod_{k=1}^{K} (P_k(S))^{w_k}$$

where, $w_k \geq 0$ are the weights corresponding to each corpus. Taking log on both sides, we get $\log(P(S)) = \sum_{k=1}^{K} w_k \log(P_k(S))$. Expanding using equation 2, we get the score of segmentation S as:

$$SP1: \quad \log(P(S)) = \textstyle\sum_{i=1}^{|S|}\sum_{k=1}^{K} w_k \log(P_k(s_i)) \quad (7)$$

Here, we note that $\log(P_k(s_i))$ is a negative quantity. Also, the left hand side, $\log(P(S))$ is a log probability and should be negative. Thus, we restrict $w_k \geq 0$. We call this the supervised unigram probability model ($SP1$). In section 3, we describe ways to learn w_ks.

Similarly, let $(\log(P_1(s_i|s_{i-1})), \ldots, \log(P_K(s_i|s_{i-1})))$ be the log bigram probabilities from K corpora. The supervised bigram probability model ($SP2$) is defined as:

$$SP2: \quad \log(P(S)) = \textstyle\sum_{k=1}^{K} w_k \left(\log(P_k(s_1)) + \sum_{i=2}^{|S|} \log(P_k(s_i|s_{i-1}))\right) (8)$$

Wang et al. [16], also use the binomial transformation model, $P(x|S) = P_\#^{|S|}(1-P_\#)^{(|x|-|S|+1)}$. The log probability formula for this model can be simplified as:

$$\begin{aligned} \log(P(x|S)) &= |S|\log(P_\#) + (|x| - |S| + 1)\log(1-P_\#) \\ &= |S|(\log(P_\#(1-P_\#)) + (|x|+1)\log(1-P_\#) \end{aligned}$$

The last term is constant and can be ignored while scoring. Thus from equation 1, we get the scoring function as:

$$Sc(S) = \sum_{k=1}^{K} w_k \log(P_k(S)) + w_{K+1}|S|$$

Here, $w_{K+1} = \alpha(\log(P_\#(1-P_\#))$ and we have ignored the constant term $\alpha(|x|+1)\log(1-P_\#)$. We call this the *extended probability model* (EP) which has $|S|$ as an additional feature over the probability model. This model can be trained analogously to the supervised probability model.

2.4 Length Specific Probability Models

In the previous section, we defined the supervised probability models, where score of a segmentation is a weighted sum of log probabilities over all segments and corpora (see equation 7). The weights are specific to each corpora. However, there may be many scenarios where none of the corpora are able assign the highest probability to the correct segmentation. In such cases, no amount of training can get the correct segmentation for these examples.

For example, consider the domain string **eatontownrealestate** whose correct segmentation is **eatontown real estate**. We use two corpora, the search and webscope corpora, which are described in more detail in the experiments section. The log probability feature vectors for **realestate**, **real**, and **estate** are $(-10.5, -16.9)$, $(-7.2, -8.0)$, and $(-7.6, -9.2)$, respectively. One can see that in both the corpora the joined form *realestate* get a higher log probability. There are also many cases where one corpus leads to splitting of legitimate words, e.g. **codington** is split into **coding** and **ton**.

The problem here is that there are too few parameters in the above models to express the segmentation scores. In this section, we define models which use the length of segments to alleviate these problems. Length of segments is important to consider because the two basic operations: splitting and joining of segments, which are used to generate alternate segmentations result in change in length of segments. Splitting always results in segments which have lower lengths than the original ones, and vice versa. Thus, weights specific to length give additional parameter to the learning algorithm for scoring the correct segmentation highest.

Let L be the length of longest token in all the corpora. We use the length specific weights $w_{kl}, k = 1, \ldots, K \;\; l = 1, \ldots, L$ to define the unigram *length specific probability* (LS1) model as:

$$\log(P(S)) = \sum_{k=1}^{K} \sum_{l=1}^{L} \sum_{i=1}^{|S|} w_{kl} \mathbf{1}(|s_i| = l) \log(P(s_i)) \qquad (9)$$

where $\mathbf{1}(|s_i| = l)$ is the indicator for length of s_i being equal to l. Thus the log probability features are chosen only for the appropriate length. Similarly, the bigram *length specific probability* (LS2) model is given by:

$$\log(P(S)) = \textstyle\sum_{k=1}^{K} \sum_{l=1}^{L} w_{kl} \left(\mathbf{1}(|s_1| = l) \log(P(s_1)) + \sum_{i=2}^{|S|} \mathbf{1}(|s_i| = l) \log(P(s_i|s_{i-1}))\right) \qquad (10)$$

The length specific models are linear models. Hence, the dynamic programming algorithm can be used for inference (finding the best segmentation) in these models.

A concern with this model is that there are too many parameters which could result in overfitting of the training data. However, we show in the experiments section that using max margin training methods, these models show good generalization. In the next section, we describe the inference algorithm for unigram and bigram models.

2.5 Inference Algorithms

Models given in the previous section give the score of a segmentation S given an input string x. Here, we describe the inference algorithm which gives the best segmentation $S^* = \arg\max_S Score(S, x; \mathbf{w})$. For conciseness of description we write a single form of the unigram scoring function:

$$Sc(S, x; \mathbf{w}) = \sum_{i=1}^{|S|} \mathbf{w}^T \mathbf{F}(s_i, x)$$

where $\mathbf{w} = (w_1, \ldots, w_M)$ and $\mathbf{F}(s_i, x) = (F_1(s_i, x), \ldots, F_M(s_i, x))$ are the weights and features. It is easy to see that the unsupervised and supervised unigram probability models (equations 2,5,7), and the length specific unigram probability model (equation 9) can all be expressed in this form.

Number of all possible segmentations is exponential in length of the input string, $|x|$. We describe a dynamic programming algorithm for computing it efficiently for the unigram model. Let $V(i)$ be the maximum possible score over all segmentations of the string $x(1 \ldots i)$. We can compute $V(i)$ recursively as:

$$\begin{aligned} V(0) &= 0 \\ V(i) &= \max_{j=0,\ldots,i-1} (V(j) + \mathbf{w}^T \mathbf{F}(x((j+1) \ldots i)) \end{aligned}$$

Here, with abuse of notation, we write $\mathbf{F}(x((j+1) \ldots i))$ to denote the feature vector for the word $x((j+1) \ldots i)$. The equation fixes the last segment as $x((j+1) \ldots i)$ and varies j over the appropriate range to find the highest scoring segmentation for the string $x(1 \ldots i)$. The best segmentation for x can be obtained by storing the last best j for each $V(i)$ and tracing back from $V(|x|)$. Some of the learning algorithms also require the second best segmentation, which can be obtained by maintaining the second best and updating it when ever the best is changed. Such a scheme was also described in [10].

For the bigram models, the feature depends on both the current and the previous segments. The scoring function can be written as:

$$Sc(S, x; \mathbf{w}) = \sum_{i=1}^{|S|} \mathbf{w}^T \mathbf{F}(s_{i-1}, s_i, x) \qquad (11)$$

where for $i = 1$, $s_{i-1} = \phi$. $\mathbf{F}(\phi, a, x)$ return the appropriate unigram features for a, and $\mathbf{F}(a, b, x)$ return the appropriate bigram features for (a, b). It is easy to see that unsupervised and supervised bigram probability models (equations 3,6,8), and the length specific bigram probability model (equation 10) can all be expressed in this form.

The dynamic programming scheme for finding $S^* = \arg\max_S Score(S, x; \mathbf{w})$ can be derived as an extension to the above scheme. Let $V(i, j)$ be the maximum possible score over all segmentations of the string $x(1 \ldots i)$, where the last segment is $x((j+1) \ldots i)$. We have the restriction $0 \le j < i \le |x|$. V can be calculated recursively as:

$$\begin{aligned} &V(0,0) = 0 \\ &V(i,0) = \mathbf{w}^T \mathbf{F}(\phi, x(1 \ldots i)), \; i = 1, \ldots, |x| \\ &V(i,j) = \\ &\max_{k=0,\ldots,j-1} (V(j,k) + \mathbf{w}^T \mathbf{F}(x((k+1) \ldots j), x(j+1 \ldots i)) \end{aligned}$$

The best segmentation can computed by storing the backlinks (as in the previous case) and tracing them back from $\max_{j=0,\ldots,|x|-1} V(|x|, j)$. The unigram inference algorithm takes $O(|x|^2)$ time for inference on x, while the bigram inference algorithm takes $O(|x|^3$ time.

3. LEARNING SCHEMES

In the previous sections, we described linear scoring functions where the features derived from various sources (corpora) are weighted by a weight vector $\mathbf{w} = (w_1, \ldots, w_M)$. We also derived the inference algorithms for unigram and bigram models. In this section, we describe learning methods which, given an annotated training dataset of strings and segmentations, estimates the best set of values for the parameters $\mathbf{w}$.

Two learning paradigms are possible: unsupervised - without using training data, other than the corpora, and supervised - using additional training data. Wang et al. [16] use unsupervised methods to learn weights of a mixture language model. Here we focus on supervised methods, which use additional data.

Let $\mathcal{D} = \{(x_n, S_n^*) | n = 1, \ldots, N\}$ be the training dataset, where, x_n is the string and S_n^* is its best segmentation. Typically, a learning algorithm maximizes (or minimizes) a gain (or loss) function, as a function of $\mathbf{w}$, given the dataset $\mathcal{D}$.

Maximum-likelihood principle proposes to optimize the parameters $\mathbf{w}$ such that the likelihood of the dataset $\mathcal{D}$, is maximized. For a given $\mathbf{w}$, the scores of the supervised models can be converted into a conditional probability score as:

$$P(S|x;\mathbf{w}) = \frac{1}{Z(\mathbf{w})} e^{Sc(S,x;\mathbf{w})} = \frac{1}{Z(\mathbf{w})} e^{\mathbf{w}^T \sum_{i=1}^{|S|} \mathbf{F}(s_{i-1}, s_i, x)}$$

This idea has been used in conditional random fields [5]. Traditional CRFs do not allow features on segments. Semi-Markov CRFs [10], allow features to be defined using segments, but do not allow dependence on the context of the segment. Hence, bigram models cannot be trained using this method. Also, the implementation of CRFs provided at http://crf.sourceforge.net, does not allow use of negative features. This is a problem as all the log probability features are negative. Hence, we do not use the maximum likelihood training paradigm. In the next section we formulate the problem as a max-margin learning problem and solve it using the struct-SVM framework [13].

3.1 Structured learning formulation

In this section, we describe the max-margin formulation [13, 12] for learning the weights $\mathbf{w}$, which takes the scores of competing segmentations into account. To do so, we define the gain function for a segmentation S of the n^{th} data point as: $G(\mathbf{w}; x_n, S_n^*) = Sc(S_n^*) - Sc(S)$. This measures the gain in score of the annotated segmentation S_n^* over the segmentation S. Margin, $\mathcal{M}_n$ for the n^{th} data point is defined as:

$$\mathcal{M}_n = \max_S G(\mathbf{w}; x_n, S_n^*)$$

Let γ be a lower bound on the margin, $\mathcal{M}_n \geq \gamma$. In the max-margin formulation, we are interested in maximizing γ. However, this problem is unbounded. Hence, we regularize using the constraint $\|\mathbf{w}\| \leq 1$. Thus, for the bigram scoring function (equation 11) the problem can be written as:

$$\mathbf{min}_{\mathbf{w}} \|\mathbf{w}\|^2; \quad \textbf{subject to:}$$

$$\mathbf{w}^T \left(\sum_{i=1}^{S_n^*} \mathbf{F}(s_{i-1}^*, s_i^*, x) - \sum_{i=1}^{S_n} \mathbf{F}(s_{i-1}, s_i, x) \right) \geq 1, \forall S_n, n$$

This problem may become infeasible in many cases. Hence, we add slack variables $\xi_n \geq 0$ for each sequence $x_n \in \mathcal{D}$. Thus, the final formulation becomes:

$$\mathbf{min}_{\mathbf{w},\xi} \;\; \|\mathbf{w}\|^2 + C \sum_{n=1}^{N} \xi_n \quad \textbf{subject to:}$$

$$\mathbf{w}^T \left(\sum_{i=1}^{S_n^*} \mathbf{F}(s_{i-1}^*, s_i^*, x) - \sum_{i=1}^{S_n} \mathbf{F}(s_{i-1}, s_i, x) \right) \geq 1 - \xi_n, \forall S_n, n$$

Note that here, we describe the learning formulation for the bigram model, which trivially applies to the unigram model.

This formulation is a special case of the structured learning framework described in [13, 12]. We use the algorithm described by Joachims et al. [13] to learn the parameters $\mathbf{w}$, by solving the above optimization problem. The algorithm requires a routine which computes the best and the second best segmentation of a sequence x, given parameter values $\mathbf{w}$. We use algorithms described in section 2.5 to provide these routines. In the next section, we report the experimental results obtained using the methods described here.

4. EXPERIMENTAL RESULTS

The unsupervised and supervised models described here, were tested extensively on multiple datasets and corpora. The datasets were constructed from three sources: Yahoo! domain match traffic[11], English gigaword data from Linguistic Data Consortium (LDC) [8] and twitter api[1]. Log probability features were obtained using two sources: Yahoo! search logs, and Yahoo! web N-grams corpus. The scheme for learning parameters of supervised models was implemented using the Struct-SVM package by Joachims et al. [13], available at `http://www.cs.cornell.edu/People/tj/svm_light/svm_struct.html`.

In this section, we report the results of experiments conducted on five datasets using various models described above. We also, report results for experiments conducted on twitter hashtag segmentation.

4.1 Evaluation of Segmentation Techniques

4.1.1 Experimental setup

We use five datasets, described in table 1, in our experiments to evaluate various segmentation schemes. The first two datasets *domains-head* (dom-head) and *domains-full* (dom-full) are created from Yahoo! domain match data. The segmentations are annotated by Yahoo! human experts. Dom-head contains 3865 sample domains which are from the "head" domains (top 20% of all domains, sorted by monthly traffic). Dom-full contains 9784 random samples from the entire domain match domains database. Both datasets were filtered to have only alphanumeric characters.

The third, fourth and fifth datasets (also called news datasets) consist of headlines from the APW, AFP and CNA, sources in the LDC gigaword data [8]. Six months up to a year of headlines from the 2008 archives were chosen from each of the sources. Headlines containing numbers and other punctuations were discarded. The unsegmented forms of the headlines were constructed by removing the spaces between successive words. The segments contain mix of proper nouns and regular English words.

[1]https://dev.twitter.com/docs

Table 1: Dataset Description

Dataset	Average number of tokens	Number of examples	Average unsegmented length
Dom-head	2.24	3865	12.41
Dom-full	2.21	9784	11.40
AFP	7.42	23929	39.45
APW	6.94	24376	37.38
CNA	7.17	21667	39.90

The corpora of terms was built using the following sources:

- Yahoo! search logs totaling to 3 million unique terms and 220 million unique bigrams.

- Yahoo! web N-grams data totaling to 200,000 terms and 190 million bigrams.

The parameter $P_{\#}$ is estimated from the respective training datasets as the ratio of total number of segments and the sum of unsegmented string lengths.

Max-margin training algorithm described in section 3 was implemented using the Struct-SVM [13] package. Struct-SVM requires the inference algorithm (algorithm to compute the best segmentation), which computes the best and the second best segmentation for a given parameter value. We implemented the dynamic programming based inference algorithm described in section 2.5 for inference with the unigram and bigram models. Our implementation using Struct-SVM allows, training both unigram and bigram models. The implementation can handle negative feature values.

In our test datasets, we have only *one* correct segmentation of the unsegmented string. The correct segmentations of the domains, in datasets Dom-head and Dom-full, are provided by the human experts. For the news datasets (AFP, APW, and CNA), the news headline gives the correct segmented form, and the unsegmented form is obtained by concatenating the words in the news headlines without the spaces. This is different from the setting described in [16], where the human experts provide a ranked list of multiple valid segmentations for each unsegmented domain.

The scoring function based techniques developed in this paper are capable of returning a ranked list of possible valid segmentations for each unsegmented string. As motivated in section 1, our main metric of interest accuracy at position 1. We call this metric the **Top-1 accuracy**. Top-1 accuracy is the fraction of unsegmented strings from the entire dataset for which the result retrieved at position 1 matches with the corresponding correct segmentation. The usual metrics of precision and recall do not make sense in our case because there is only one relevant document for each unsegmented string.

All metrics reported in this paper for supervised methods are obtained by averaging results from 3-fold cross-validation on the entire dataset. The datasets are permuted randomly and split into 3 equal parts. For each of the parts, accuracy is computed by training on the remaining parts and testing on the held-out part. The average of accuracy over all the held-out parts is reported. This procedure ensures that the accuracy is not artificially overestimated due to overfitting of the data.

Table 2: Top 1 accuracy: Unsupervised models

Unigram			
Dataset	Single Corpus		Joint
	Search	Web n-gram	Probability
Dom-head	78.08	73.01	82.06
Dom-full	66.47	75.22	80.41
AFP	68.69	70.6	77.64
APW	69.4	70.4	79.26
CNA	66.7	69.53	75.47
Bigram			
Dataset	Single Corpus		Joint
	Search	Web n-gram	Probability
Dom-head	87.34	76.58	88.35
Dom-full	75.86	77.45	82.13
AFP	85.72	77.06	88.47
APW	86.63	76.61	88.54
CNA	86.44	78.42	87.87

For unsupervised methods, metrics are calculated on the entire dataset. All the accuracies reported here are measured for the entire segmentation (sentence precision) as opposed to the word precision metric used in earlier works [3, 14]. Sentence precision is a stricter metric compared to word precision.

4.1.2 *Comparison of Unsupervised Models*

In this section we report accuracies for unsupervised models on all the datasets. Table 2, reports top-1 accuracy for the unsupervised models. The joint probability model performs better than each single corpus model, clearly suggesting that combining multiple corpora help in word segmentation.

The two corpora (search and web n-gram), when used independently, perform differently on the datasets. This shows that the language styles of the two corpora are different.

We also observe that when the divergence in performance of the individual corpora (using single corpus model) is high, the performance of the joint probability model improves marginally. However, when the performance of individual corpora are similar, the performance of joint probability model improves significantly. This is apparent for news datasets. This suggests that learning weights for each corpus is helpful. When the individual accuracies are similar, the learning algorithm can choose nearly equal weights. However, when the accuracies differ, the learning algorithm can give higher weight to the corpus with more matching language style.

Also, we observe that the improvement in accuracy from unigram to bigram models is higher for news datasets than for domains. This is because the average number of segments in domains datasets (~ 2) is much lower than that of news datasets (~ 5).

The results suggest that combining information from multiple corpora, by taking product of probabilities works well. This is a motivation for the supervised models considered in this paper.

4.1.3 *Supervised Models*

In this section, we report accuracies for the supervised models developed in this paper. We also report accuracy for the best unsupervised model for comparison. As expected,

Table 3: Top 1 accuracy: all models

Unigram				
Dataset	Unsupervised	Prob-Supervised	EP1	LS-1
Dom-head	82.06	85.56	85.47	88.74
Dom-full	80.41	80.97	81.15	87.02
AFP	77.64	78.95	78.39	85.45
APW	79.26	80.52	80.65	86.47
CNA	75.47	78.04	77.83	86.13
Bigram				
Dataset	Unsupervised	Prob-Supervised	EP2	LS-2
Dom-head	88.35	88.43	88.42	89.72
Dom-full	82.13	82.34	82.49	84.73
AFP	88.47	89.13	90.66	91.15
APW	88.54	89.89	90.96	91.97
CNA	87.87	90.13	90.89	91.13

Table 4: Cross Testing - Probability Supervised

Trained on	Dom-head	Dom-full	AFP	APW	CNA
Unigram					
Dom-head	–	79.09	**78.56**	**80.25**	**77.5**
Dom-full	**83.02**	–	**78.53**	**80.11**	**76.31**
AFP	**84.06**	**80.63**	–	**80.45**	**77.21**
APW	**84.57**	80.17	**78.14**	–	**78.02**
CNA	**84.5**	80.13	**78.9**	**80.31**	–
Bigram					
Dom-head	–	81.67	88.29	**89.02**	**90.01**
Dom-full	87.89	–	87.91	87.73	86.78
AFP	88.02	81.87	–	**89.44**	**89.89**
APW	87.99	81.88	**89.02**	–	**89.69**
CNA	87.94	81.88	**89.12**	**89.66**	–

all supervised models perform better than the best unsupervised model (see table 3). For unigram models, the supervised probability model improves significantly over unsupervised model for most datasets. However, the improvement for bigram models is modest. This is due to the fact that bigram models are more powerful than unigram models for word segmentation.

The extended probability model shows insignificant improvement over the supervised probability model for unigram. This is due to the fact that unigram probability model can be rewritten to get the log counts and the number of token as the features. Hence, the contribution from incorporating additional number of tokens features is marginal. In contrast, the extended probability model shows significant improvement over supervised probability model.

The length specific model outperforms all other models. This confirms our hypothesis that length is an important parameter for scoring segmentations. The difference in accuracy between the length specific (LS) and the best unsupervised model is ($\sim$ 7%) for unigrams on all datasets and ($\sim$ 3.5%) for bigrams on news datasets. The accuracy of bigram models for domains does not improve much due to low average number of segments ($\sim$ 2).

4.1.4 Generalization of Supervised Models

Results in the previous sections, show that supervised models, trained on a training set and tested on samples from the same datasets, perform better than unsupervised models. This may suggest that supervised methods merely tune models for those datasets, and not find a generally better model. Here, we test this hypothesis, by training the length specific unigram and bigram models on each of the five datasets, and reporting accuracies on other datasets as well. Tables 4 and 5 reports the cross-dataset testing results, for unigram and bigram models. The rows represent the training dataset and the columns represent the test dataset. We report top 1 accuracy. The numbers in bold represent experiments which resulted in improvement over the best unsupervised accuracy on the same test dataset.

Table 4 reports the cross testing results for the supervised probability models. The results indicate that supervised probability models generalize well on all datasets. In case of unigram models, the supervised probability model improves over the best unsupervised model for all datasets,

Table 5: Cross Testing - Length Specific

Trained on	Dom-head	Dom-full	AFP	APW	CNA
Unigram					
Dom-head	–	**83.3**	–	–	–
Dom-full	**84.24**	–	–	–	–
AFP	**84.89**	**83.36**	–	**86.27**	**82.49**
APW	**83.33**	**83.56**	**85.84**	–	**83.52**
CNA	81.99	**82.51**	**83.49**	**83.25**	–
Bigram					
Dom-head	–	80.15	–	–	–
Dom-full	77.23	–	–	–	–
AFP	86.77	80	–	**91.67**	**89.32**
APW	86.75	79.08	**91.14**	–	**89.32**
CNA	86.54	79.84	**90.34**	**90.35**	–

while, in case of bigram models, the improvement is seen only for news datasets.

Table 5 reports cross-testing results for length specific models. The length specific models trained on each of the news datasets generalize well on all the other datasets. In case of unigram models, for most datasets, the accuracies obtained are better than accuracy of the best unsupervised model, while for bigram models, the improvement is seen only for news datasets.

The length specific models trained on domains datasets, do not perform well on news datasets. During training on domains datasets, the weights corresponding to long lengths are not tuned. This results in incorrect unsegmented string to outscore the correct segmentation in news datasets.

We observe that while cross-testing, the length specific bigram models trained on domains dataset perform worse than unigram models trained on domains, confirming that bigram length-specific models trained on domains dataset do not generalize well.

Table 6: Accuracy for Cache Models

	Dom-head	AFP	APW	CNA
CRP	58.36	49.88	43.27	42.53
HDP	71.33	50.60	43.05	38.00

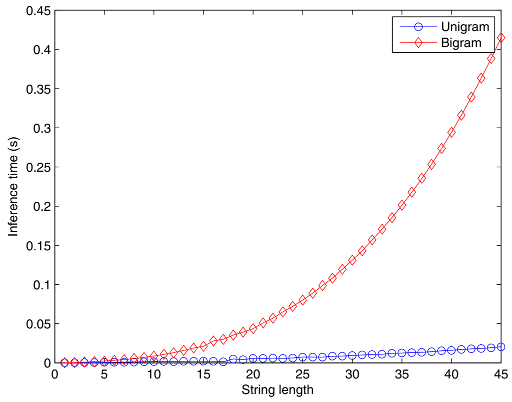

Figure 1: Average inference time as function of string length

4.1.5 *Cache models*

Table 6, reports results from CRP and HDP models proposed in Goldwater et al. [3] (described in section 1.1). Gibbs sampling schemes described in [3] were implemented and repeated 1000 times to estimate mode of the respective distributions. Table 6 reports top 1 accuracies for four datasets studied here. It is clear that these models are worse than, even unsupervised models.

4.1.6 *Inference time*

Figure 1 shows inference time for unigram and bigram models as a function of input string lengths. The inference for all practical domains and hashtags take only fraction of a second. For unigram models, the inference time is of the order of milli second. As expected, the bigram inference takes much more time than unigram inference.

4.2 Twitter Search Enhancement

As proposed earlier (section 1), we intend improve the recall of tweets on twitter search, by segmenting hashtags in tweets and using extracted words to surface additional relevant tweets. In order to demonstrate the feasibility of this idea, we created a dataset of twitter trends from all locations in the US, UK, Canada, India, Australia and New Zealand. Trends were collected for 9 days, starting from 21st September 2011 using the twitter trends API. The trends having hashtags or single words were discarded. We also collected 1% random sample of all public tweets ($\sim$ 1.5M per day) using twitter streaming API, for 9 days, starting from 21st September 2011.

Hashtags in the tweets were segmented using the LS2 model trained on AFP dataset. For each trend t, we identified the set of tweets $B_1(t)$ which contain t. Also, for each trend t, we identified the segmented hashtags which matched t and the set of tweets $B_2(t)$ containing these hashtags. Table 7 reports results from the experiment for 9 days, starting

Table 7: Results for twitter experiment

Date	# trends	$\|A_1\|$	$\|A_2\|$	Avg RI for A_2
9/21	159	35	9	29.3
9/22	159	46	7	42.3
9/23	148	49	17	19.8
9/24	213	37	12	25.12
9/25	262	44	18	44.84
9/26	233	42	10	9.22
9/27	185	50	18	8.9
9/28	184	52	20	22.7
9/29	204	47	12	54.6

from 21st September 2011. The second column reports the number of trends collected on each day, and the third column gives the number of trends t such that $|B_1(t)| \geq 100$. We restrict our attention to only such trends, as improvement over smaller number of matches are too noisy to make inferences. The recall improvement for trend t, is calculated as $RI(t) = \frac{|B_2(t) \setminus B_1(t)|}{|B_1(t)|} * 100$.

Let $A_1 = \{t : |B_1(t)| \geq 100\}$ and $A_2 = \{t : t \in A_1 \wedge RI(t) \geq 5\}$. Third column ($|A_2|$) of table 7 gives the number of trends which show 5% or more improvement in recall. It is clear that a significant fraction of the trends considered (third column), show atleast 5 % improvement in recall. The average recall improvement (fourth column) for such trends is significant.

5. CONCLUSION

In this paper, we study the problem of word segmentation from the point of view of two web applications: web domain segmentation and and twitter hashtag segmentation. Language models are the standard tools used for word segmentation. We try the joint probability model, which is a simple unsupervised way of combining the information from multiple corpora. The joint probability model significantly improves the accuracy of word segmentation over the single corpus models.

This motivates us to define the supervised probability model which uses weighted scoring function which weighs the different corpora differently. We propose to learn these weights using max-margin methods. The experimental results show that this model outperforms all unsupervised models.

Finally, we observe that length of a segment plays significant role in the contribution of the segment towards the score of an entire segmentation. Using this, we define the length-specific model, which uses weights specific to the length of the segments, for scoring a segmentation. Experimental results show that the length specific models outperform all other models tested here. Moreover the length-specific models show significant improvement in the top-1 accuracy over the best unsupervised models.

Acknowledgment

The authors are thankful to Chiranjib Bhattacharyya, Sanjay Chawla, and anonymous reviewers for helpful comments.

6. REFERENCES

[1] R. Casey and E. Lecolinet. A survey of methods and strategies in character segmentation. *IEEE*

Transactions on Pattern Analysis and Machine Intelligence, 1996.

[2] S. F. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, ACL '96, pages 310–318, Stroudsburg, PA, USA, 1996. Association for Computational Linguistics.

[3] S. Goldwater, T. L. Griffiths, and M. Johnson. Contextual dependencies in unsupervised word segmentation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, ACL-44, pages 673–680, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.

[4] P. Koehn and K. Knight. Empirical methods for compound splitting. In *Proceedings of the tenth conference on European chapter of the Association for Computational Linguistics - Volume 1*, EACL '03, pages 187–193, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[5] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, pages 282–289, 2001.

[6] K. Macherey, A. M. Dai, D. Talbot, A. C. Popat, and F. Och. Language-independent compound splitting with morphological operations. In *ACL HLT*, 2011.

[7] C. D. Manning, P. Raghavan, and H. Schtze. *Introduction to Information Retrieval.* Cambridge University Press, New York, NY, USA, 2008.

[8] R. Parker, D. Graff, J. Kong, K. Chen, and K. Maeda. English gigaword fourth edition, 2009.

[9] F. Peng, F. Feng, and A. McCallum. Chinese segmentation and new word detection using conditional random fields. In *Proceedings of the 20th international conference on Computational Linguistics*, COLING '04, Stroudsburg, PA, USA, 2004. Association for Computational Linguistics.

[10] S. Sarawagi and W. W. Cohen. Semi-markov conditional random fields for information extraction. In *NIPS*, 2004.

[11] S. Srinivasan and S. Bhattacharya. Learning to tokenize web domains. In *Proceedings of the 20th international conference companion on World wide web*, WWW '11, pages 129–130, New York, NY, USA, 2011. ACM.

[12] B. Taskar, C. Guestrin, and D. Koller. Max-margin markov networks. In *NIPS*, 2003.

[13] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector machine learning for interdependent and structured output spaces. In *ICML*, 2004.

[14] A. Venkataraman. A statistical model for word discovery in transcribed speech. *Computational Linguistics*, 27(3):351–372, 2001.

[15] K. Wang, X. Li, and J. Gao. Multi-style language model for web scale information retrieval. In *Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '10, pages 467–474, New York, NY, USA, 2010. ACM.

[16] K. Wang, C. Thrasher, and B.-J. P. Hsu. Web scale nlp: a case study on url word breaking. In *Proceedings of the 20th international conference on World wide web*, WWW '11, pages 357–366, New York, NY, USA, 2011. ACM.