

# Focusing on Novelty: A Crawling Strategy to Build Diverse Language Models

Luciano Barbosa  
AT&T Labs – Research  
180 Park Ave  
Florham Park, NJ 07932  
lbarbosa@research.att.com

Srinivas Bangalore  
AT&T Labs – Research  
180 Park Ave  
Florham Park, NJ 07932  
srini@research.att.com

## ABSTRACT

Word prediction performed by language models has an important role in many tasks as e.g. word sense disambiguation, speech recognition, hand-writing recognition, query spelling and query segmentation. Recent research has exploited the textual content of the Web to create language models. In this paper, we propose a new focused crawling strategy to collect Web pages that focuses on novelty in order to create diverse language models. In each crawling cycle, the crawler tries to fill the gaps present in the current language model built from previous cycles, by avoiding visiting pages whose vocabulary is already well represented in the model. It relies on an information theoretic measure to identify these gaps and then learns link patterns to pages in these regions in order to guide its visitation policy. To handle constantly evolving domains, a key feature of our crawler approach is its ability to adjust its focus as the crawl progresses. We evaluate our approach in two different scenarios in which our solution can be useful. First, we demonstrate that our approach produces more effective language models than the ones created by a baseline crawler in the context of a speech recognition task of broadcast news. In fact, in some cases, our crawler was able to obtain similar results to the baseline by crawling only 12.5% of the pages collected by the latter. Secondly, since in the news domain avoiding well-represented content might lead to novelty, i.e. up-to-date pages, we show that our diversity-based crawler can also be helpful to guide the crawler for the most recent content in the news. The results show that our approach was able to obtain on average 50% more up-to-date pages than the baseline crawler.

## Categories and Subject Descriptors

H.3.3 [Information Search and Retrieval]: Search process.

## General Terms

Algorithms, Design, Experimentation.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CIKM'11, October 24–28, 2011, Glasgow, Scotland, UK.  
Copyright 2011 ACM 978-1-4503-0717-8/11/10 ...\$10.00.

## Keywords

Web Crawling, Language Modeling, Novelty, Perplexity.

## 1. INTRODUCTION

Web is an invaluable data repository. The text data on the Web has been harnessed for tasks as diverse as named entity recognition [25], word sense disambiguation [23] and machine translation [17] in natural language processing; search and question answering [18] in information retrieval; and pronunciation modeling and language modeling in speech recognition [19].

What makes the text on the Web so attractive for these applications? Apart from the sheer size of the textual repository, Web text is compelling for two main reasons. First, Web text in general is *diverse* and not limited to a particular domain. This aspect is particularly important as language technologies are beginning to cope with ever increasing need to handle open domain input in tasks such as search, question-answering, and language translation. Second, and perhaps more important, is that the Web text from sources, such as news websites, blogs, microblogs and others, is *dynamic*, and tracks the current news and popular events.

For these reasons, recent research has exploited the textual content of the Web to create models for natural language tools, in particular, language models [14, 21]. Typically, language models are built on a training corpus of sentences with the assumption that the distribution of  $n$ -grams in the training set is the same as the distribution of  $n$ -grams in the task context where the language model would be used. This assumption, also called as the *independent and identically distributed (iid)* assumption, is reasonable for tasks which are domain limited and where the target data does not change over time. However, in open domain applications such as question-answering, broadcast news speech recognition, where the input to the models change based on the current events, the *iid* assumption results in a mismatch between the training and target contexts. This mismatch can be interpreted as holes or gaps in the training data. To address this issue, language models are typically transformed to match the target distributions using adaptation techniques [5].

In this paper, we handle this mismatch problem in a different way. To fill the gaps present in the language model, we design techniques to collect Web text whose vocabulary is not well represented in the model, producing a more diverse language model. Unlike a traditional static-corpus driven method where models trained on static corpora would soon be outdated due to the changing profiles of the input, Web

text based models can potentially be continuously updated to keep them from becoming stale.

Previous approaches [21, 6] have mined sentences from Web text to increase the corpus of a language model. Their primary goal is to collect data most similar to the domain of interest. It is to be expected that with the increase in size of the training data from this kind of content a language model would be more *specialized* to the domain of interest. Along similar lines, regular focused crawling also aims to gather Web pages in a well-defined topic [8, 11]. Topics are defined using a few example documents and the task of the crawler is to increase the coverage of pages in the defined topic.

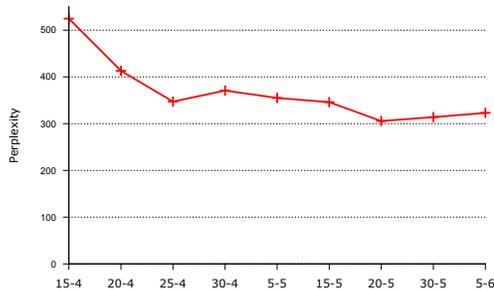
In contrast to these previous approaches, we want to *un-focus* the crawler on previous collected data, from which the language model had originated, and crawl for regions of novelty with respect to the language model. The crawler identifies these regions by using an information theoretic measure, and then guides its visitation policy based on the patterns of links in these regions. To handle constantly evolving domains, wherein new events appear constantly, an important feature of our crawler is its ability to adjust its focus as the crawl progresses, to be able to reflect in its policy the most current state of its environment. In summary, the main contributions of this work are:

- A novel crawling strategy that is tightly coupled with the goal of creating diverse language models by filling in the gaps in the current language model;
- The usage of an information theoretic measure to guide the crawling policy instead of using regular supervised learning techniques [8];
- The combination of techniques from different research communities – NLP, Machine Learning, Information Theory and Information Retrieval – to address the important problem of creating a corpus for language models.

The remainder of the paper is organized as follows. In Section 2, we introduce concepts of a language model and explain the need for diversification of a language model. We present in detail the main components that compose our crawling in Section 3 and present experimental results in Section 4. We review related work in Section 5 and conclude in Section 6.

## 2. LANGUAGE MODELS

Many speech and language applications need a mechanism to rank sentences according to their well-formedness based on the grammar of a language. However, designing such a ranking function has eluded us despite many years of research. Instead, the likelihood of a sentence (denoted as  $P(W)$ , where  $W$  is a sentence) in a corpus of text is used as an approximation for grammaticality, with the assumption that grammatically well-formed sentences occur more often than ungrammatical ones. However, due to sparseness concerns, the probability of the joint event  $W = w_1, w_2, \dots, w_m$  is approximated using an independence assumption as shown in Equation 2, also known as an  $n$ -gram model.



**Figure 1: The perplexity of a news story from May 20, 2010 over language models from news Websites ranging from mid April to beginning of June 2010.**

$$P(W) = \prod_{i=1}^m P(w_i | w_1, \dots, w_{i-n+1}) \quad (1)$$

$$\approx \prod_{i=1}^m P(w_i | w_{i-1}, \dots, w_{i-n+1}) \quad (2)$$

The individual probabilities in the product are computed using maximum likelihood estimation from a corpus of word sequences, and when a given  $n$ -gram is not observed in the corpus, then its probability is estimated using a lower order  $n - 1$  gram.

Besides evaluating the effectiveness of a language model in the context of a task, such as machine translation or speech recognition, a metric that is commonly used to compare language models that is independent of the task is called *Perplexity*. As seen in Equation 2, an  $n$ -gram model can be viewed as a model for predicting the  $n^{th}$  word given the preceding  $n - 1$  words of history. Low perplexity implies a better fit of the model for the word sequence. Perplexity of a language model  $P$  on an  $m$ -word sequence  $W$  is defined as in Equation 3.

$$Perplexity(P, W) = P(W)^{-\frac{1}{m}} \quad (3)$$

$$= \sqrt[m]{\prod_{i=1}^m \frac{1}{P(w_i | w_{i-1}, \dots, w_{i-n+1})}} \quad (4)$$

To give a better intuition about perplexity and how perplexity can be associated to time, we show how the perplexity of a fixed page evolves over different language models built over time. Figure 1 presents the perplexity of a news article from May 20th, 2010 over language models created from news Websites between April 15th and June 6th<sup>1</sup>. As it can be seen, the closer the language model is to the story, the smaller the perplexity. In fact, the lowest value is on the day that the story was released. The perplexity starts to increase again afterwards. Our assumption is that the vocabulary of pages that have high perplexity over a language model represent the gaps on it. In Section 3, we explain how we exploit this to create our diversity crawler.

## 3. CRAWLING FOR DIVERSITY

Our primary goal in this work is to create a crawling strategy that prioritizes diversity. In order to do so, the crawler searches for Web regions where the vocabulary of the pages is poorly covered by previous crawls. Thus, the current

<sup>1</sup>The news article was not used the model of May 20th.

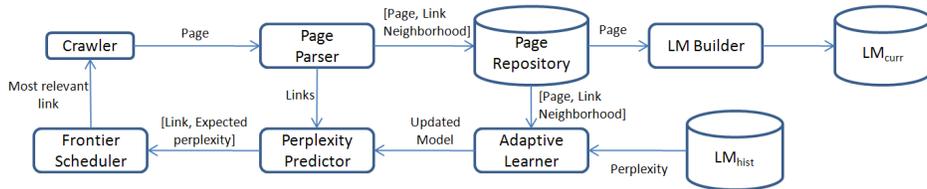


Figure 2: Architecture of the Diversity Crawler.

crawl will be guided based on the language model created from previous crawls. Our assumption is that pages whose vocabulary is well-covered on the language model contain some common patterns in their links. These patterns will then guide the link visitation of the current crawl to avoid collecting similar pages.

Figure 2 presents the architecture of our crawler. Let  $LM_{hist}$  represent the language model from previous crawls and  $LM_{curr}$  the language model to be created from the current crawl. The crawler uses the *Perplexity Predictor* to guide its visitation policy by predicting the expected perplexity with respect to  $LM_{hist}$  of a page given its link. This prediction is sent to the *Frontier Scheduler* that then decides the next link to be visited by the crawler. The downloaded pages are stored in the *Page Repository* which are used by the *LM Builder* to create the new language model,  $LM_{curr}$ . In the next cycle,  $LM_{curr}$  replaces  $LM_{hist}$ . Since the language model changes every cycle, the crawler needs to adapt its visitation policy accordingly. This is done by the *Adaptive Learner*. It creates a new model for the *Perplexity Predictor* as the crawl progresses based on  $LM_{hist}$  and on the pages/links collected in the current crawl. In the remainder of this section, we describe in detail these core components of the system.

### 3.1 Perplexity Predictor

As we mentioned before, to create diverse language models, the crawler should focus on novel regions for  $LM_{hist}$ . We model them as regions that contain pages with high perplexity values over  $LM_{hist}$ . To guide the crawler visitation policy to these regions, the crawler relies on the links to pages in these regions, similar to previous focused crawlers [7, 20]. But, in contrast with these approaches, which try to focus the crawler on a particular topic, we aim to *de-emphasize* the pages previously covered, since our objective is to increase the diversity of the language model.

Formally, let  $P = p_1, \dots, p_{|P|}$  be the pool of  $|P|$  pages and  $N = n_1, \dots, n_{|P|}$  be the neighborhoods of the links pointing to these pages. The link neighborhood is composed of the words in the anchor of the link, around it (10 words before and after the anchor) and in its URL. Ideally, we would like to visit a set,  $\mathbf{VP}$ , with the  $k$ -best pages with the highest perplexity according to  $LM_{hist}$ , as shown in Equation 5. However, this involves crawling all the pages and evaluating the perplexity of the text on each page which is an expensive operation. We instead would like to estimate the perplexity of the text on a page *without* crawling it.

$$\mathbf{VP} = \operatorname{argmax}_{p_i}^{(k)} \operatorname{Perplexity}(LM_{hist}, p_i) \quad (5)$$

In order to do so, we assume that the text in the neighborhood of the link that points to the page has some predictive power about the perplexity of the page. Our assumption here is that the link neighborhood contains information that indicates the content of the page. This indication

can be at different levels. There might be links that contain no indication of the page’s content, when its link neighborhood is empty or just contains non-descriptive words; a broad indication, for instance, a URL that contains the term “sports” probably has content about sports in its body but just inspecting it, it is not clear which topic within sports that page is talking about; or a more descriptive indication of the page’s content, for instance, a URL with the string “yankees-clinch-playoff-berth-beat-blue-jays” indicates that its content is about the baseball teams Yankees and Blue Jays, and their participation in playoffs.

Now, the problem of estimating the perplexity of a page based on the neighborhood of a page is a regression problem with the text features of the neighborhood serving as the independent variables. However, since we are not interested in the exact perplexity value but just in a shallow approximation of the real value for the crawling schedule, and to reduce learning time [16] (learning is performed on the fly), we aggregated links with similar perplexity values in the same class. To discretize the perplexity values, we use the equal-frequency binning approach, since we want to avoid the problem of having unbalanced training data [22], and set the number of bins equal to 3 (we empirically tried other values but obtained poorer results). As the overall distribution of perplexity values changes over time, its distribution within each bin also changes.

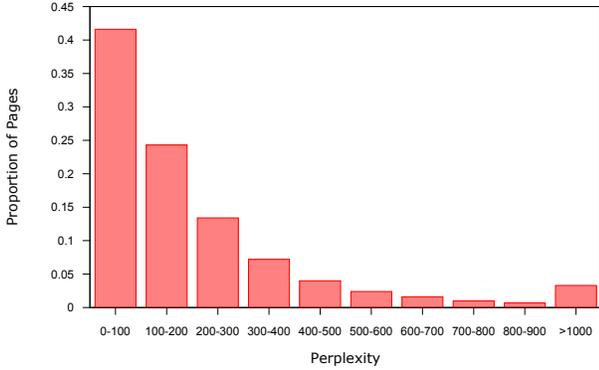
To give a better intuition about the distribution of perplexity values in this task, in Figure 3, we plot the distribution of the perplexity of pages from a random crawl of 100,000 pages at time  $t_1$  from a set of news sites over a language model created in a previous day ( $LM_0$ ) composed by 100,000 pages collected using the same strategy on these sites. As one can see, the perplexity is skewed towards small values: for 40% of the pages, it is between 0 and 100, and the higher the perplexity value, the smaller the number of pages. As one would expect, and we show in Section 4, this distribution varies over time, skewing more towards small values, which makes the problem of finding high-perplexity pages much harder.

Thus, given the text neighborhood  $n_i$ , the *Perplexity Predictor* classifies  $n_i$  into a particular perplexity class  $c$  using the features ( $\phi$ ) computed from the text of the neighborhood as shown in Equation 6.

$$c_i^* = \operatorname{argmax}_c P(c|\phi(n_i)) \quad (6)$$

We tried three different algorithms which are widely used in text classification: <sup>2</sup> Naïve Bayes, Support Vector Machines (SVM) and Maximum Entropy (MaxEnt). From the  $t_1$  crawl used in the previously presented perplexity distribution experiment, we prepared the training data by randomly

<sup>2</sup>We use the data mining software Weka to build the classifiers [13].



**Figure 3: Distribution of perplexity values obtained by pages from a random crawl over a language model created in a previous day.**

Algorithm	Accuracy	F-measure for High
Naïve Bayes	0.48	0.45
SVM	0.57	0.616
MaxEnt	0.57	0.613

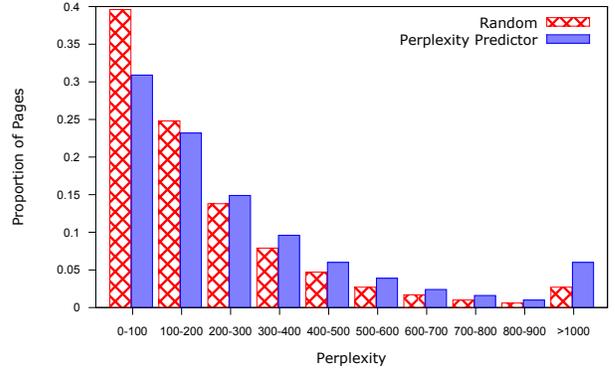
**Table 1: Overall accuracy and F-measure over the high-perplexity class obtained by the 3 classifier algorithms over the test set.**

selecting 48,000 links and a different set of 12,000 links as testing set and, using the  $LM_0$  from the same experiment, we calculated the perplexity of their respective pages. We binned the perplexity values for the training and test pages and trained the different classifiers and evaluated them on the test set.

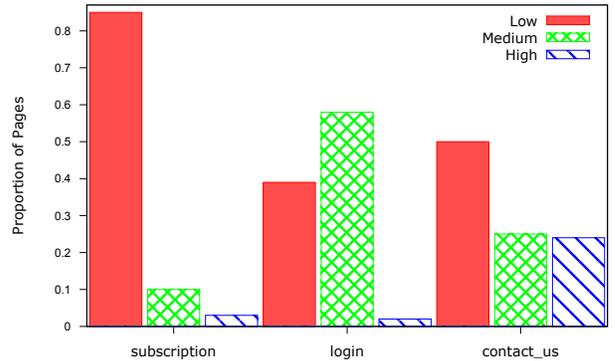
Table 1 presents the accuracy over the three perplexity classes (Low, Medium, High) and F-measure over the class that corresponds to the highest expected perplexity values (High), which is the one we are most interested in, since the links in this class might point to pages that can diversify the language model. The best results in terms of accuracy were obtained by SVM and MaxEnt classifiers. SVM performed slightly better than MaxEnt for F-measure. For this reason, we selected SVM as the algorithm for the Perplexity Predictor.

To verify whether the Perplexity Predictor based crawler performs a better job than a random crawler, we set up two crawls: a random crawler and a crawler guided by the Perplexity Predictor. Figure 4 shows the percentage of pages for different values of perplexity obtained using  $LM_0$ . Whereas 40% of the pages collected by the random crawler is between 0-100, only 30% within this range was collected by the perplexity crawler. Moreover, the perplexity crawler was able to collect a much higher proportion of pages with perplexity greater than 200: 45% versus 35%. Overall, the average perplexity value of the pages obtained by the perplexity crawler was much higher than the one obtained by the random crawler: 361 versus 255. These numbers indicate that Perplexity Predictor is in fact able to focus the crawler on Web regions that contain high-perplexity pages.

One of the reasons of the effectiveness of Perplexity Predictor is its ability to capture the link patterns of stale pages from Web site sections whose vocabulary is similar across sites as, e.g., login, subscription and obituary pages. To give a concrete example of this, we looked at the distribution of *login*, “contact us” and *subscription* pages over the perplexity



**Figure 4: Comparison of perplexity values obtained by pages crawled using the Perplexity Predictor and a random approach.**



**Figure 5: Proportion of login, subscription and “contact us” pages in the 3 different perplexity classes.**

classes. More specifically, we selected for each class (Low, Medium, High), 3,500 links classified by the Perplexity Predictor and plotted, in Figure 5, the proportion of links of such pages in each set. The numbers show that most of the links to these pages were classified as having low perplexity, which is the result we would expect. In fact, looking at features with high information gain in this classification task, patterns in the links as login, log, sign, obituary have high values of information gain.

## 3.2 Frontier Scheduler

Having identified the links to crawl using the Perplexity Predictor module, the role of the Frontier Scheduler is to schedule the next link to be visited by the crawler. The schedule for visitation is defined based on expected perplexity of the links available in the frontier and on other visitation policies, for example, to avoid hitting a same Web server in a short period of time.

The Frontier Scheduler is implemented as a set of  $N$  queues, where each queue corresponds to a Perplexity Predictor class. Within a queue, links are ordered based on their likelihood of belonging to the class associated with the queue. The queue from the highest perplexity class contains more elements than the other ones. In our experiments the proportion of the queue sizes was 1 (low), 2 (medium), 10 (high).

As the accuracy of the Perplexity Predictor is far from perfect and also to avoid some possible bias contained in the links that might hurt the diversity of the crawl, the crawling frontier is updated in batches. When the crawler

---

**Algorithm 1** *LM Builder*

---

- 1: **Input:**  $N$ ,  $predictorLM$ ,  $webPages$ ,  $devSet$   
{ $N$ : number of language models to be merged,  
 $predictorLM$ : LM used for the Perplexity Predictor in the current crawl,  
 $webPages$ : Web pages from current crawl,  
 $devSet$ : development set.}
  - 2:  $sents = ParsePages(webPages)$   
{Parse the Web pages into sentences.}
  - 3:  $lm = CreateLM(sents)$   
{Create LM from sentences.}
  - 4:  $currentLM = CombineLMs(predictorLM, lm, devSet)$   
{Combine LMs used by the  $predictorLM$  and the current LM.}
  - 5:  $predictorLM = SelectLMs(devSet, N)$   
{Select the best  $N$  LMs from the pool of previous LMs and combine them.}
  - 6: **Output:**  $predictorLM$ ,  $currentLM$
- 

starts, all seeds are placed in the highest perplexity queue. At each step, the crawler selects the link with the highest estimated perplexity from the first nonempty queue. When a page is downloaded, its links are extracted and added to a separate persistent frontier according to the prediction of the Perplexity Predictor. Only when the queues in the crawling frontier become empty, the crawler loads the queues from the persistent frontier.

### 3.3 LM Builder

After the crawler finishes its cycle, the LM Builder creates a language model from the crawled pages, as we show in Algorithm 1. First, it parses the Web pages downloaded by the crawler in the Page Repository into sentences. It does this by breaking the documents based on the HTML tags that correspond to the newline characters: “p”, “dt”, “dd”, “tr”, “li”, “br” and all header tags (“h1”, “h2”, etc). After removing all of the HTML tags, pieces of text located between these special tokens are considered to be sentences. All the punctuation and special characters are also removed.

The next step is to create a language model from these sentences. We build trigram models from these sentences for all the experiments presented in this paper. The trigram model is built using a language modeling toolkit [12], with Katz back-off scheme for the  $n$ -grams not observed in the training corpus. The resulting model is represented as a weighted finite-state acceptor which is used for evaluating the perplexity of a given input sentence.

The resulting language model is then merged with the previous ones, created in previous cycles. The ideal scenario would be if one could merge an infinite number of LMs. But in practice, we need to limit the total number of LMs to be merged. First, because the higher the number of models, the longer is the merging process. This is an issue because we want to provide up-to-date LM to be used for the word prediction. Second, because the higher the number of models, the longer the perplexity computation, since its calculation time is dependent on the size of the LM. The crawler needs to calculate on the fly the perplexity of the pages just crawled in order to guide its policy according to the current LM (see Section 3.4). Large perplexity calculation times lead the online learning to be prohibitive. Due to all of that, we have to limit the number of models to be merged. A simple approach would be to consider only the  $n$  previous models to merge. This, however, does not work

well for our approach because language models from the past have influence in how language models are built in present. Thus, for instance, the language model from the first cycle has a greater influence over data in many cycles ahead than LMs built without this dependency.

Our solution to that is to limit the total number of LMs to merge by choosing the  $k$  most suitable LMs from the pool of previous LMs. Given a set of language models  $LM = \{LM_1, \dots, LM_n\}$ , the objective is to provide a weighted interpolation of the language model as shown in Equation 7. The weights in this equation are estimated using a Expectation-Maximization algorithm so as to minimize the perplexity of a tuning set of sentences.

$$LM_{merged} = \sum_{i=1}^n w_i * LM_i \quad (7)$$

The set of  $k$  selected LMs is merged using a development set to set their weights ( $predictorLM$ ) and used by the crawler for the next crawl cycle. After the next cycle, the LM generated from the crawl is added to  $predictorLM$  using interpolation creating the LM for the current word prediction ( $currentLM$ ). Finally, a new  $predictorLM$  is created from the current pool of LMs for the next cycle.

### 3.4 Adaptive Learner

Previously, in Section 3.3, we mentioned that after every crawl a language model, the  $predictorLM$ , is built to guide the crawler policy in the next crawling cycle. This is primarily done by updating the Perplexity Predictor according to the new  $predictorLM$ . Recalling that the Perplexity Predictor matches patterns in the links to pages (link neighborhood) with perplexity values from a language model, the task is to obtain from the current crawl this information: link neighborhood  $\rightarrow$  perplexity. An important point here is, since we want to deal with dynamic environments with new events (patterns) constantly arising, the patterns of the links used by the Perplexity Predictor need to be up-to-date. That is the reason why we need to use the patterns in the links of the current crawl and, as a result, the Adaptive Learner process needs to be performed on the fly. The task of updating the Perplexity Predictor is performed by the Adaptive Learner. It works as follows. Initially, the crawler starts with a random link visitation policy, since it does not know a good strategy to follow for the new language model,  $predictorLM$ . After a specified number of crawled pages, a learning iteration is performed by collecting, for each page  $p$ , the link neighborhood of the links that point to  $p$ , and the perplexity of  $p$  over  $predictorLM$ . A training data is produced as a result of this process and then, a new Perplexity Predictor is created (see Section 3.1), updating the crawler’s link policy. As the last step, the Adaptive Learner updates the values of expected perplexity of the current links in the frontier based on the new Perplexity Predictor. The Adaptive Learner is invoked periodically, when the crawler visits a pre-determined number of pages.

## 4. EXPERIMENTAL EVALUATION

In this section, we assess our crawling strategy to build diverse language models. We chose to perform this evaluation in the news domain. Looking for novelty is an important requirement in this domain since new content is constantly being generated. We measure the performance of our approach

in two different tasks. First, in the context of an automatic speech recognition (ASR) task of broadcast news. Secondly, in the context of crawling up-to-date news content, since in the news domain avoiding well-collected content from the past might lead to novelty.

## 4.1 Experimental Setup

**Data.** We compiled a list of 4,874 news Web sites, and used Web pages crawled from them to compose the language models' corpus. Since we aim to create generic language models, we tried to cover a great variety of Web sites – local, national and international news – as well as news from different topics: business, sports, technology, student life etc.

**Crawler and Language Model Strategies.** We executed two different crawling strategies to create the corpus for the language models:

- **Baseline Crawler:** to the best of our knowledge there is no previous crawling strategy to handle the problem we are addressing in this paper. However, a reasonable approach to obtain diverse content is to randomly select the next link to be visited by the crawler in order to avoid been trapped in a particular region/content. This is the strategy implemented by the baseline. Thus, from the corpus generated by this crawler, we composed the language models in two different ways: a single language model from the current crawl, which we denote as *Baseline Crawler (Single)* in the remaining of this section; and a language model created by the combination of the LM of the current crawl with previous LMs generated by this crawler (see Section 3.3), termed as *Baseline Crawler(Merged)*;
- **Diversity Crawler:** This is our approach that uses perplexity to guide the crawler's policy. As we mentioned in Section 3.4, the crawler starts with a random policy. The learning iteration to create and update the Perplexity Predictor, performed by the Adaptive Learner, runs when the crawler collects 15,000, 30,000 and 50,000 pages. Similar to the baseline crawler, we created two different language model compositions: a single language model from the current crawl – *Diversity Crawler (Single)*; and a combination of the current LM with previous LMs produced by this crawler – *Diversity Crawler (Merged)*.

Each crawling cycle is one day long. Thus, the two crawlers were executed every day during 23 days. Each crawler collected a total of 100,000 pages, generating a corpus of about 8 million sentences, on average. Initially, a random crawl was performed to create the initial LM (day 0), and was used by the Diversity Crawler to define its policy in the next cycles. To interpolate the language models, we use a window of eight days. The models chosen to compose from this window is based on the methodology discussed in Section 3.3.

The development set used to set the interpolated weights has to reflect the current news stories, otherwise it would give higher weights to a model built in the past that do not have much influence in the present. Our assumption is that the root pages from the news Web sites usually contain the most up-to-date information in the site and a good summary of the current main stories. For this reason, the development

set used is a subset of the initial pages of the news sites. This sample is not used in the corpus that create the language model.

**Evaluation in the ASR task.** There are two main metrics used to evaluate the performance of language models in the context of an ASR system: word-error rate (WER) and perplexity. WER measures the number of substitutions, deletions and insertions between the actual output of an ASR system with respect to a reference transcript. To use it, it is necessary to build a full speech recognition system, which involves other components than the LM such as an acoustic model and a pronunciation model. As a result of that, computing the WER is very expensive and it does not isolate the contribution of the language model. The perplexity of a reference transcript over a language model, on the other hand, can be easily calculated and does not depend on the other components, only the language model itself. Furthermore, previous work has showed there is a strong correlation between perplexity and WER for in-domain data [15, 9], i.e., when the language model data and the speech task are in the same domain, which is the case in this experiment, as we show later in this section.

Based on that, in this evaluation, we measure the perplexity of the daily language models from news sites over reference audio transcripts from broadcast news in the same day<sup>3</sup>. For this purpose, we collected daily transcripts of TV and radio programs from three Web sources:

- **CNN<sup>4</sup>:** cnn.com provides an archive of transcripts of their TV shows as well as of some CNN international and HLN programs;
- **NPR<sup>5</sup>:** NPR makes available an API that can be used to retrieve audio and transcripts for their main programs. These programs cover a broad range of topics: politics, economy, sports etc. We were able to obtain transcripts from 8 programs;
- **Livedash<sup>6</sup>:** Livedash.com is a website in which the user can search over the transcripts of TV shows as well as read the actual transcripts of these shows. From this source, we obtained transcripts from a total of 21 programs in different topics such as sports (e.g. ESPN), business (e.g. CNBC) and general news (e.g. MSNBC, Fox News).

As our goal is to build broad-domain language models, we tried to cover as many different transcripts as possible to evaluate the approaches. It must be noted that, as some programs are aired only on weekdays and others only on weekends, the total number of transcripts available may vary over time.

**Evaluation in the news crawling task.** Regarding the news crawling task, we want to verify, besides generating diverse language models, whether our strategy is also useful to collect fresh pages in the news domain. More specifically, we are interested in assessing how many up-to-date pages

<sup>3</sup>The language models built from news sites each day in this experiment contain on average 97% of the words in the daily transcripts, which means that we can consider we are dealing with in-domain data.

<sup>4</sup>CNN transcripts: <http://archives.cnn.com/TRANSCRIPTS/>

<sup>5</sup>NPR API: <http://www.npr.org/api/index.php>

<sup>6</sup>Livedash: <http://www.livedash.com/>

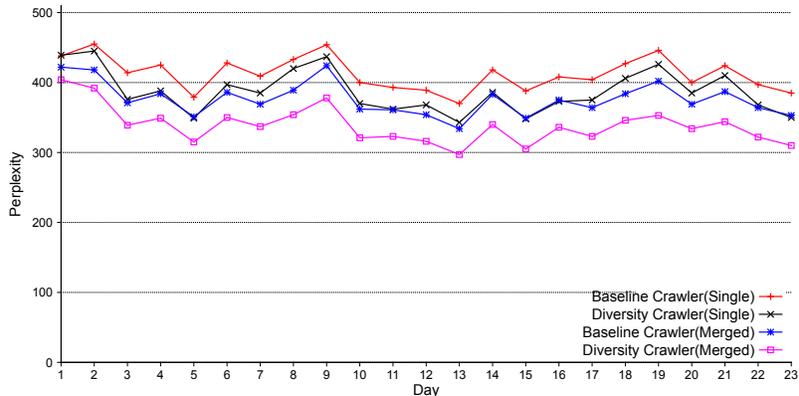


Figure 6: Perplexity obtained by the 4 language model configurations over the transcripts.

from the news sites were collected daily by each crawler. One possible way to calculate that is to look at the HTTP Last-Modified header of these pages. However most of the sites do not provide such field. Instead, we counted the number of pages collected by each crawler whose URL indicates the page was posted in the current day. Patterns as  $\langle \text{year} \rangle / \langle \text{month} \rangle / \langle \text{day} \rangle$ ,  $\langle \text{year} \rangle - \langle \text{month} \rangle - \langle \text{day} \rangle$  and  $\langle \text{year} \rangle \langle \text{month} \rangle \langle \text{day} \rangle$  were used for this purpose. Although this is an underestimation of the actual number of up-to-date pages, since some of them might not have any indication of posting date in their URL, we believe it is a reasonable measure to compare the different approaches.

## 4.2 Assessing the Diversity Crawler

First, we present results regarding the ASR task. Figure 6 presents the perplexity obtained by the 4 different LM configurations over the daily transcripts. The first thing to note is that the Diversity Crawler(Merged) got the lowest perplexity values on all days of the experiment, followed by Baseline Crawler(Merged), Diversity Crawler(Single) and Baseline Crawler(Single). These numbers confirm that our crawling approach combined with the model merging is in fact effective in building more fresh and diverse language models. In addition to that, the Diversity Crawler(Single) outperforms the Baseline Crawler(Single) in all days. This means that, by only using the LM of the day’s crawl, our crawler was able to detect more novelty on the LM than the Baseline Crawler. It does so by avoiding pages whose vocabulary already appeared in the past, or is not useful for the day’s news as, for instance, stale pages in the websites (e.g. “contact us”, “about” etc, see Section 3.1). Another interesting result from these numbers is that the Diversity Crawler(Single), which only uses the current LM produced by the Diversity Crawler, had a performance close to the Baseline Crawler (Merged) and, on some days, almost the same (e.g. 3, 15 and 23). Recall that the Baseline Crawler (Merged) uses around 800,000 pages in its corpus whereas Diversity Crawler(Single) only 100,000. In terms of numbers of tokens contained in the LM, the LM of Diversity Crawler(Single) is usually less than half of the Baseline Crawler (Merged).

With regard to the changing behavior of these configurations over time, they present similar trends, as shown in Figure 6. These trends are mostly influenced by the performance of the LM of the day. Initially, the difference of performance between Diversity Crawler (Merged) and Baseline

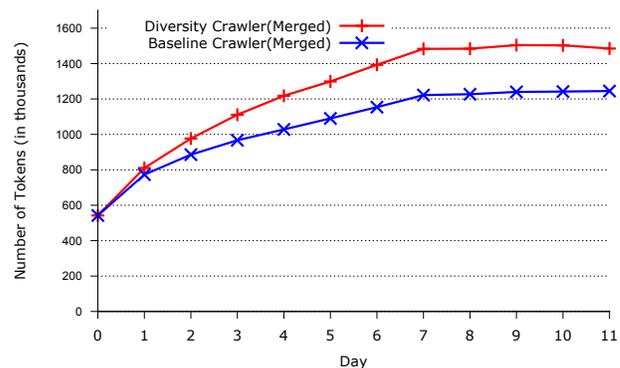


Figure 7: Size of the language model generated from the merging for the first 12 days of the experiment.

Crawler (Merged) increases for the first eight days, which is the size of the window of the model merging. This occurs because on each day, the Diversity Crawler generates a much fresher and more diverse LM. This can clearly be seen by looking at the size of the LMs produced by each crawling strategy with merging. Figure 7 depicts the number of tokens that composes the merged LMs each day for the first twelve days (after this period their size did not change much). At the beginning, both approaches started from the same LM, i.e., the language model created on the first day by running a Baseline Crawler. However, on each day, the Diversity Crawler (Merged) obtained many more new tokens than the Baseline Crawler. Since both approaches crawled the same number of pages, this increased difference of LM’s sizes suggests that the vocabulary of the pages collected by the Baseline Crawler has a greater overlap with previous crawls than for the Diversity Crawler. This confirms that the Diversity Crawler is in fact able to produce more diverse language models. For the days 7-10, the number of LMs are fixed to eight, the difference of LMs’ sizes fluctuates within a small interval, since their sizes did not change much. An interesting observation from these numbers is that an LM created by the Baseline Crawler (Merged) with eight models (e.g., day 7) has similar size than an LM with only five models (e.g. day 4) created by the Diversity Crawler(Merged).

**News Crawling Task.** Figure 8 presents the total number of up-to-date pages collected daily by each crawling strategy. The Diversity Crawler outperforms the baseline in all days of the experiment. On average, it collected 50% more up-to-

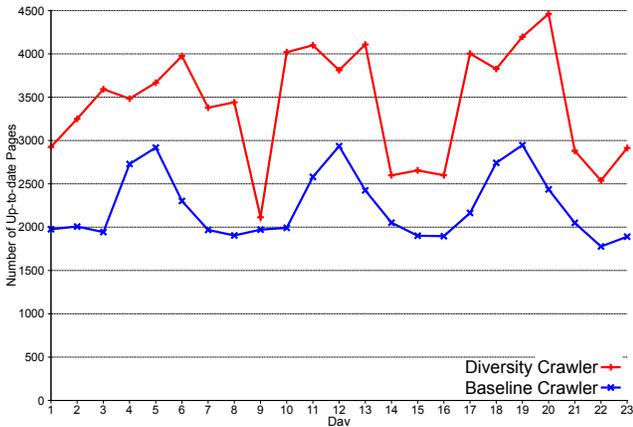


Figure 8: Number of up-to-date pages collected by each crawler.

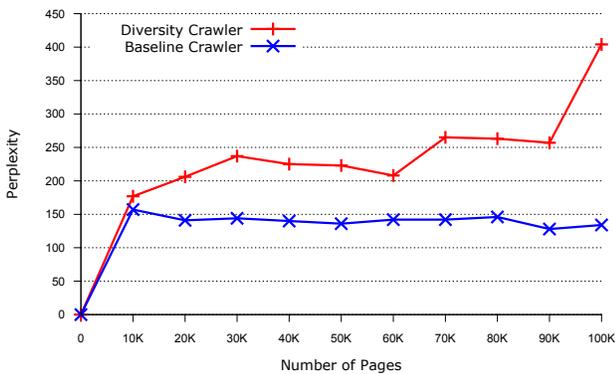


Figure 9: Average perplexity obtained by the Diversity and Baseline Crawlers during a crawling cycle.

date pages than the baseline and, in some days (day 10, for instance), almost twice more. These numbers confirm that in the news domain, focusing on the gaps of language models created from previous crawls lead the crawler to explore regions of more novelty where there are more up-to-date pages. An additional observation from Figure 8 is the occurrence of a weekly cycle of fresh content. This is clearer for the Baseline Crawler, although the Diversity Crawler presents a similar trend. Peaks were at days 5,12 and 19, which were Wednesdays in our experiment, and after these days there was a decay in the number of up-to-date pages towards the weekend.

**Performance within Cycles.** To illustrate the behavior of the Diversity Crawler during each crawling cycle and to show the dynamic nature of the newscasts, we look now at numbers about its performance within cycles. Figure 9 depicts the perplexity of the pages collected by the Diversity Crawler and Baseline Crawler during the crawl of day 1. The perplexities of these pages were calculated over the LM of day 0 ( $LM_{hist}$ ), and their values were averaged in intervals of 10,000 pages (0-10k, 10k-20k etc). As our goal is to obtain as diverse set of pages as possible in relation to the  $LM_{hist}$ , the higher the number of high-perplexity pages that were collected, the better is the performance of the crawling approach, since the crawler is filling more gaps in the LM. These numbers show that the Diversity Crawler outperforms the Baseline Crawler throughout the entire cycle. Moreover, the Diversity Crawler improves its performance

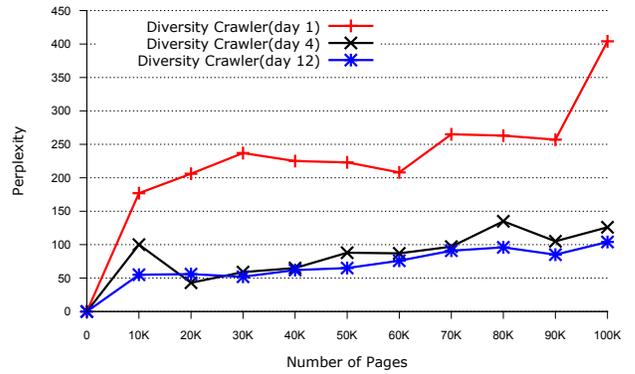


Figure 10: Average perplexity obtained by the crawler during three different crawling cycles.

over time, mainly because its Adaptive Learner component is able to learn on the fly the patterns of the links to high-perplexity pages, whereas the performance of the Baseline Crawler remains almost constant.

**Fewer Gaps over Time.** As the Diversity Crawler fills the gaps in the language model in each new crawling cycle, fewer gaps remain to be filled which makes the problem of finding new gaps even harder. To illustrate this, we plotted, in Figure 11, the overall distribution of the perplexity over the corresponding  $LM_{hist}$  from pages collected by the Diversity Crawler in the first three days of our experiment. As one can see, the distribution of the perplexity becomes more and more skewed towards pages with low perplexity making the problem of finding high-perplexity pages harder. This is reflected in the overall crawler’s performance during each cycle. Figure 10 shows the performance over time of the Diversity Crawlers at three different days: 1, 4, 12. It is clear that the crawler from the first day obtained many more high-perplexity pages than the later ones. It is worth noting that in all three crawls the Diversity Crawler’s performance improves over time. Again, this occurs because of the online learning component of the crawler, which is able to learn patterns in the links to the pages in the LM gaps.

**Interpolation Weights.** As a final point, we present some details about the weights assigned to different models by interpolation (Section 3.3). Figure 12 shows these weights for the Baseline and Diversity Crawlers in the first eight days of our experiment. Both curves indicate a decay factor over time, as we would expect given that we are dealing with time-sensitive data. However, the weights assigned to the more recent models in the Baseline Crawler are much higher than the ones assigned to the models created by the Diversity Crawler. Furthermore, the first LM in the Diversity Crawler interpolation has the third highest weights among the other models in this mixture, whereas in the Baseline Crawler interpolation the first LM has the lowest weight. These weights reflect the strategy used by each crawler to collect the data. As the Diversity Crawler’s visitation policy of the current crawl is influenced by previous LMs, complimentary in some sense, these LMs have a higher influence over the current data and, consequently, have higher weights than the ones created by the Baseline Crawler, which produces independent LMs. Recall that these weights are assigned by interpolation based on the development set that we provide (a sample of the initial pages of the Websites, see Section 1). The exponential decay of weights indicates that this development set in fact contains up-to-date data.

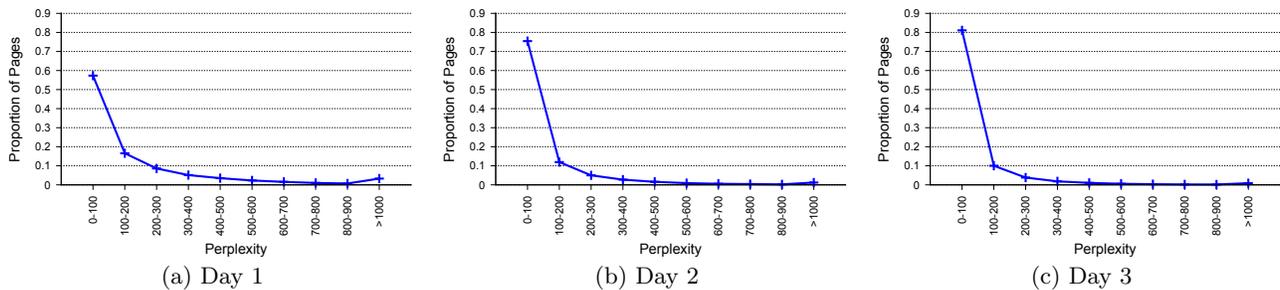


Figure 11: Evolution of the perplexity distribution.

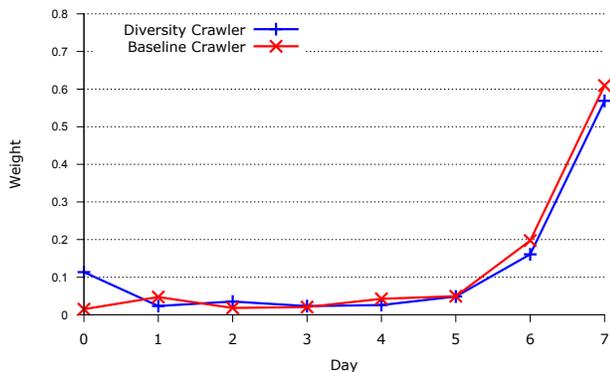


Figure 12: Weights assigned by the merging process to the language models created by the Diversity and Baseline Crawlers for the first eight days of the experiment.

## 5. RELATED WORK

This work applies focused crawler techniques to the well-known NLP problem of building language models. In this section, we discuss previous approaches in these two areas related to ours.

### 5.1 Focused Crawling

There is a vast literature in the area of focused crawlers [11, 8, 7, 20, 1, 4, 3]. Their primary goal is to collect Web pages in a well-defined topic. For instance, Chakrabarti et al. [7] look for Web pages in domains such as astronomy, linux, cancer etc; Barbosa and Freire [4] propose a crawling approach that tries to locate Web forms in domains as airfare, hotel, cars etc. In direct contrast with these previous approaches, our goal in this work is to build a broad-domain language model by avoiding crawling pages which are well represented in previous crawls. Although we have different goals, we use similar strategies. We exploit patterns in the links, anchors and around the links, similar to [20, 7], to guide the crawler towards regions with pages with high perplexity, and use online learning to change the crawling policy according to some feedback about the “quality” of the pages currently crawled [1, 7]. Online learning is specially important in very dynamic scenarios, since a policy used in a previous crawling cycle might not have the same effect in the current one.

### 5.2 Language Modeling

Huan et al. [14] presented a very extensive study of language models built from a Web crawl for three particular problems: query spelling, query bracketing and query segmentation. They showed, for instance, that the anchor lan-

guage model is more similar to the queries (lower perplexity) than the body of the page and also obtained the best performance in almost all the presented scenarios for these three tasks. This work is complementary to ours. It motivates the usage of language models for IR tasks whereas ours presents a solution to produce better language models.

Sarikaya et al. [21] proposed a query-based method to collect Web data to build a language model for spoken dialog domains in combination with a in-domain LM, created from dialogs in the domain. The queries are generated from utterances of dialogs and the resulting pages are cleaned by selecting the sentences in these pages more similar to sentences in the domain. Their experiments, in the financial transaction domain, showed there is a great reduction in the word error rate by adding the Web LM. Our approach also tries to build LMs from the Web to help in the problem of word prediction. The approach of using queries to build LMs works well in well-defined and focused domains as the one used in their paper (financial transactions). However, when the goal is to build LMs for more general purposes, a more scalable approach needs to be used as the one we propose in this paper, since there is a limit for the numbers of queries that can be issued to a search engine.

There are other methods that are more focused on the process of building the language model from Web data and/or time-sensitive data. For instance, Wang and Li [24] introduced a method that adapts the LM as chunks of data are available, as in the scenario of Web crawling. Another work [10] proposed techniques to build general-purpose language models by partitioning the data into mixture components giving different weights for these components and by taking into account the recency of the words (recent words have a higher probability of appearing in the future). Along the same lines, [26] also tries to use term recency to build the language models. We performed something similar in terms of word recency when we combine the language models since the weights of the language models are set based on a development set from very up-to-date pages.

## 6. CONCLUSIONS

We have presented in this paper a crawling strategy that collects a corpus from which diverse language models are generated for the open domain of newscasts. Instead of focusing on a particular domain as regular focused crawlers, our crawler tries to diversify the language model by detecting Web regions whose vocabulary is not well represented in the model (gaps in the model). It does so by identifying high-perplexity pages. The crawler then learns patterns of links to high-perplexity pages in order to guide its link visitation policy. In addition to that, as the newscast environment is so dynamic, the crawler uses an online learning compo-

ment to adapt its policy according to the current state of the environment. We have shown in the experimental evaluation that the Diversity Crawler is able in fact to create more diverse language models than a baseline crawler, and because the crawler is able to fill in the language model's gaps, the problem of finding gaps becomes even harder after some crawling cycles.

It is important to point out that, although the framework that we present in this paper was applied for building diverse language models, it can also be employed by any kind of domain in which diversity in the data is a key component. For instance, regular Web crawlers might apply the crawling technique proposed in this paper to schedule its visitation policy in order to increase the diversity of the search engine index, which would be very helpful, for example, to improve diversity in search results [2].

## 7. REFERENCES

- [1] C. C. Aggarwal, F. Al-Garawi, and P. S. Yu. Intelligent crawling on the world wide web with arbitrary predicates. In *WWW*, pages 96–105, 2001.
- [2] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong. Diversifying search results. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pages 5–14, 2009.
- [3] D. Ahlers and S. Boll. Adaptive geospatially focused crawling. In *Proceeding of the 18th ACM conference on Information and knowledge management*, pages 445–454, 2009.
- [4] L. Barbosa and J. Freire. An adaptive crawler for locating hidden-web entry points. In *WWW*, pages 441–450, 2007.
- [5] J. Bellegarda. Statistical language model adaptation: review and perspectives. *Speech Communication*, 42(1):93–108, 2004.
- [6] I. Bulyko, M. Ostendorf, M. Siu, T. Ng, A. Stolcke, and Ö. Çetin. Web resources for language modeling in conversational speech recognition. *ACM Transactions on Speech and Language Processing (TSLP)*, 5(1):1, 2007.
- [7] S. Chakrabarti, K. Punera, and M. Subramanyam. Accelerated focused crawling through online relevance feedback. In *WWW*, pages 148–159, 2002.
- [8] S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: A new approach to topic-specific web resource discovery. *Computer Networks*, 31(11-16):1623–1640, 1999.
- [9] S. Chen, D. Beeferman, and R. Rosenfeld. Evaluation metrics for language models. In *DARPA Broadcast News Transcription and Understanding Workshop*, pages 275–280. Citeseer, 1998.
- [10] P. Clarkson and A. Robinson. Language model adaptation using mixtures and an exponentially decaying cache. In *Acoustics, Speech, and Signal Processing, 1997. ICASSP-97., 1997 IEEE International Conference on*, volume 2, pages 799–802. IEEE, 1997.
- [11] M. Diligenti, F. Coetzee, S. Lawrence, C. L. Giles, and M. Gori. Focused Crawling Using Context Graphs. In *VLDB*, pages 527–534, 2000.
- [12] V. Goffin, C. Allauzen, E. Bocchieri, D. Hakkani-Tur, A. Ljolje, S. Parthasarathy, M. Rahim, G. Riccardi, and M. Saraclar. The AT&T Watson speech recognizer. In *Proceedings of ICASSP*, pages 1033–1036, 2005.
- [13] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten. The WEKA data mining software: An update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.
- [14] J. Huang, J. Gao, J. Miao, X. Li, K. Wang, F. Behr, and C. Giles. Exploring web scale language models for search query processing. In *Proceedings of the 19th international conference on World wide web*, pages 451–460. ACM, 2010.
- [15] R. Iyer, M. Ostendorf, and M. Meteer. Analyzing and predicting language model improvements. In *Automatic Speech Recognition and Understanding, 1997. Proceedings., 1997 IEEE Workshop on*, pages 254–261. IEEE, 1997.
- [16] R. Jin, Y. Breitbart, and C. Muoh. Data discretization unification. *Knowledge and Information Systems*, 19(1):1–29, 2009.
- [17] P. Koehn. Europarl: A parallel corpus for statistical machine translation. In *MT summit*, volume 5. Citeseer, 2005.
- [18] C. Kwok, O. Etzioni, and D. Weld. Scaling question answering to the Web. *ACM Transactions on Information Systems (TOIS)*, 19(3):242–262, 2001.
- [19] C. Munteanu, G. Penn, and R. Baecker. Web-based language modelling for automatic lecture transcription. In *Proceedings of 8th Annual Conference of the International Speech Communication Association*, pages 2353–2356, 2007.
- [20] J. Rennie and A. McCallum. Using Reinforcement Learning to Spider the Web Efficiently. In *ICML*, pages 335–343, 1999.
- [21] R. Sarikaya, A. Gravano, and Y. Gao. Rapid language model development using external resources for new spoken dialog domains. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 573–576, 2005.
- [22] C. Seiffert, T. Khoshgoftaar, J. Van Hulse, and A. Napolitano. Building useful models from imbalanced data with sampling and boosting. In *Proc. 21st Annual FLAIRS Conference*, pages 306–311, 2008.
- [23] C. Stokoe, M. Oakes, and J. Tait. Word sense disambiguation in information retrieval revisited. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 159–166. ACM, 2003.
- [24] K. Wang and X. Li. Efficacy of a constantly adaptive language modeling technique for web-scale applications. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4733–4736, 2009.
- [25] C. Whitelaw, A. Kehlenbeck, N. Petrovic, and L. Ungar. Web-scale named entity recognition. In *Proceeding of the 17th ACM Conference on information and Knowledge Management*, pages 123–132. ACM, 2008.
- [26] D. Zhang, J. Lu, R. Mao, and J. Nie. Time-Sensitive Language Modelling for Online Term Recurrence Prediction. *Advances in Information Retrieval Theory*, pages 128–138, 2010.