Information Filtering, Novelty Detection, and Named-Page Finding

Kevyn Collins-Thompson, Paul Ogilvie, Yi Zhang, and Jamie Callan
Language Technologies Institute
Carnegie Mellon University
{kct,pto,yiz,callan}@cs.cmu.edu

1. Introduction

In TREC 11, our group participated in the Novelty track, Filtering track, and the Named-Page Finding task of the Web track. This paper describes our approaches, experiments, and results. As the approach for each task is quite different, the paper contains a section for each of the tasks. The following section describes our experiments in adaptive filtering, Section 3 describes named-page finding, and section 4 discusses the Novelty track.

2. Adaptive Filtering

In the adaptive filtering track, we used the same system as in TREC9 and TREC10. The Rocchio algorithm is used for anytime profile updating. More detailed information about profile updating and the system structure is available in [17]. This year, we focused on comparing different thresholding methods, and also did some experiments on using language model to improve initial query profiles.

2.1 Thresholding

Two evaluation measures were used in this year's adaptive filtering track: T11U=2*R_+-N_+[1] and T11F=1/(1/*recall*+4/*precision*)), where R_+ is the number of relevant documents delivered, and N_+ is the number of non-relevant documents delivered. T11U can be optimized if we can estimate precision, and the corresponding optimal strategy is: deliver if P(*relevant*) > 0.33. T11F can be optimized only if we can estimate both precision and recall.

When filtering, we have several training documents represented as $((x_1,y_1)(x_2,y_2),...(x_t,y_t))$, where x_i is the score of a delivered document with user feedback, and y_i=0 if document i is not relevant, otherwise y_i=1. We also order the tuples according to the constraint $x_1<x_2<...x_i<x_{i+1}<...<x_t$.

	Optimized for T11U	Optimized for T11F
NE	Yes	Yes
ML	Yes	Yes
Empirical Optimal	Yes	No
Logistic Regression	Yes	No
Bayesian Error Model	Yes	No
Greedy Search	Yes	No

Table 1: Candidate Thresholding Algorithms

We tried six threshold-setting algorithms (Table 1):

- NE (Normal-Exponential): Model the scores of the relevant documents with a Normal distribution and model the top ranking non-relevant documents with an exponential distribution as described in [1].
- ML: Maximum Likelihood Estimation as described in [19]. Use the same model as NE, but explicitly model the sample bias while estimating model parameters. This is a modified version of the Maximum Likelihood Estimation thresholding algorithm, because the early stage (when the number of relevant documents or non-relevant documents in the training set is smaller than 4, the threshold is the optimal one calculated by the model, smoothed with the old threshold and the average relevant document score using linear interpolation.
- EO (Empirical Optimal). Let all candidate threshold be $\theta_i=x_i+x_{i-1}/2$ (i=1…t). Set the real threshold at θ_j, where the evaluation measure we want to optimize achieved the empirical optimal value on training data among all of the candidate points.
- Logistic Regression: This is a strict implementation of widely known logistic regression algorithm. Although the thresholding algorithm described in [12] is also based on logistic regression, it is a modified version using calibration to fit the filtering task and data, and would probably get a better result.

[1] The exact official evaluation for utility is T11SU=(max(T11U/MaxU, MinNU)-MinNU)/(1-MinNU), where MaxU=2*(Total Number of relevant documents), MinNU=-0.5. T11SU is normalized T11U.

- Bayesian Error Model: Use uniform error model $P(y=1|x,t,e)=e+(1-2e)\Phi(x-t)$, where $\Phi(x-t)=0$ if x<t, and $\Phi(x-t)=1$ if x>=t. The system use a Beta distribution $p(e) \sim \beta(\alpha_1,\alpha_2)$ to model the prior of error e. Bayesian estimation of $P(y=1|x, \textit{training data})$ is used to estimate the precision. More information about this is available in [1].
- Greedy Search: This is a greedy algorithm that increases the threshold if a relevant document is delivered, while decrease the threshold if a non-relevant document is delivered. While the step size depends on 1) the difference between the score of the document and the current threshold and 2) how many changes the system has made before. So the strategy will modify the threshold if given a feedback for document d_i using: $t_{new}=t_{old}-2\delta_1*\max(\delta_2,score_{di}-t_{old})$ if d_i is relevant, otherwise $t_{new}=t_{old}+\delta_1*\max(\delta_2,score_{di}-t_{old})$. Where δ_1 decreases as number of feedback increases, δ_2 is set to 0.005 arbitrarily.

Notice that Logistic Regression and the Bayesian Error Model are focused on model $P(y|x)$ and do not model the marginal distribution $P(x)$. These two models can help estimate precision, but are not capable of estimating the recall. Theoretical optimization for T11F is not possible for them without getting an estimate of $P(x)$. This is a general problem for using discriminative models for thresholding, and one solution is to use the empirical distribution of x as described in [12] to help estimate recall, but it requires a lot of computation.

	TREC8	TREC9	TREC11
NE	0.324	0.360	0.365
ML	0.340	0.363	0.373
Empirical Optimal	0.213	0.344	0.315
Logistic Regression	0.212	0.300	0.289
Bayesian Error Model	0.268	0.303	0.311
Greedy Search	0.051	0.155	0.090

Table 2: T11U performance with different thresholding algorithms.

We compared these different thresholding algorithms on TREC8 and TREC9 filtering track data, using T11U as the evaluation measure. Maximum Likelihood Estimation works consistently the best on both data sets. The system was tuned using the TREC8 and TREC9 filtering track data. After submitting our results, we did more experiments on TREC11 data. The final results on the different datasets are shown in Table 2. Using the Normal and Exponential model to model the score distribution worked well on all three dataset, although their performance was not good on TREC10 data(not reported here. [18]).

2.2 Modifying an initial profile without training documents

NIST provides topics that contain title, description and narrative fields to describe the user's information interests, but what should be the initial profile description is unknown. If we look at the following sample topic from TREC8:

Query: q353

Title: Antarctica exploration

Description: Identify systematic explorations and scientific investigations of Antarctica, current or planned.

Narrative: Documents discussing the following issues are relevant: - systematic explorations and scientific investigations of Antarctica (e.g., seismology, ionospheric physics, possible economic development) - other research currently conducted or planned for the future - banning of mineral mining Documents discussing tourism are non-relevant. Documents discussing "disrupting scientific experiments" are non-relevant unless a specific experiment is identified.

We can see that the title is a better description but contains only 2-4 words. The Description and the Narrative are too long and noisy. We tried a mixture model to model how a user generates a description/narrative. Assuming a query is generated by a mixture of profile independent *general query model* M_g and profile dependant *core query model* M_q. Words such as "Identify" or "Documents" are more likely to be generated by M_g, while the words "Antarctica" is more likely to be generated from M_q. Using algorithms described in [21], we can find the *core query model* M_q and use it to do feature selection or reweighing for the initial query. Thus for each profile, we have 7 options for setting initial queries based on how we use title, description and narrative fields provided by NIST. The results on the TREC8 and the TREC11 dataset are shown in Table 3. Using the mixture model helped for the TREC8 dataset, while we didn't see significant improvement on TREC11 dataset.

Initial query	TREC8	TREC11
Title	0.3185	0.3621
Title + Description	0.3297	0.3732
Title + Description + Mixture model for reweighing words	0.3405	0.3736
Title + Description + Mixture model for feature selection	0.3524	0.3668
Title + Description + Narrative	0.3559	0.3669
Title + Description + Narrative + Mixture model for reweighing words	0.3424	0.3741
Title + Description + Narrative + Mixture model for feature selection	0.3402	0.3673

Table 3: T11U performance with different initial profile settings.

		UDESC	FDESC	Uml	Fml
= Max		0	1	0	0
> Med		97	90	92	88
< Med		3	10	8	12
Topic 1-50	T11U	0.445	0.431	0.447	0.433
	T11F	0.422	0.401	0.410	0.396
Topic 51-100	T11U	0.291	0.293	0.290	0.292
	T11F	0.041	0.038	0.034	0.035

Table 4: Performance of our official runs.

2.3 Filtering Track Results

This year, we submitted four very similar runs. UDESC and FDESC used description and title fields, while Uml and Fml used the mixture model for term reweighting. Table 4 shows our results compared with other systems. For most topics, our system performance is above the median performance.

Figure 1 shows the results for one of the 4 runs on each topic, compared with Max, Median and Baseline (the performance of a system that delivers nothing). The system performance on the first 50 topics is much better than the performance on the last 50 topics. The first 50 are created and annotated by NIST annotators, while the last 50 topics were intersections of Reuters categories. Figure 2 compares the first 50 topics with the last 50 topics by sorting the topics according to the number of relevant documents in that topic. Although the number of relevant documents is similarly distributed for the first 50 and the last 50 topics, the best performance is quite different. When we look at some relevant documents for NIST annotated topics and "intersection" topics, we feel it is hard to learn "intersection" profiles according to training documents with a bag of word model (Rocchio). More detailed research and analysis is needed to fully understand the difference between them.

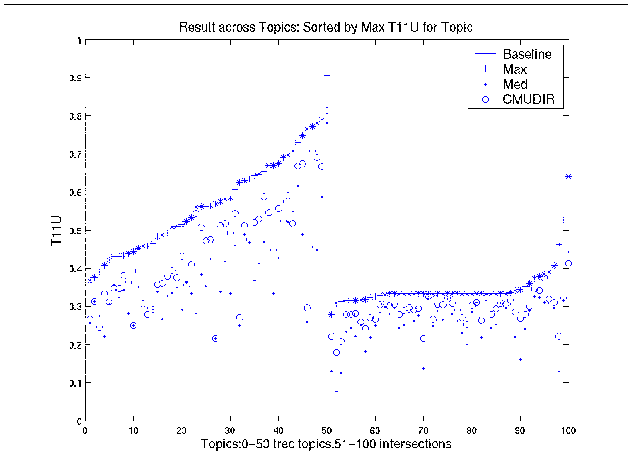

Figure 1: Compare our result (CMUDIR) with the best performance (Max) and the Median Performance (Med) for each Topic.

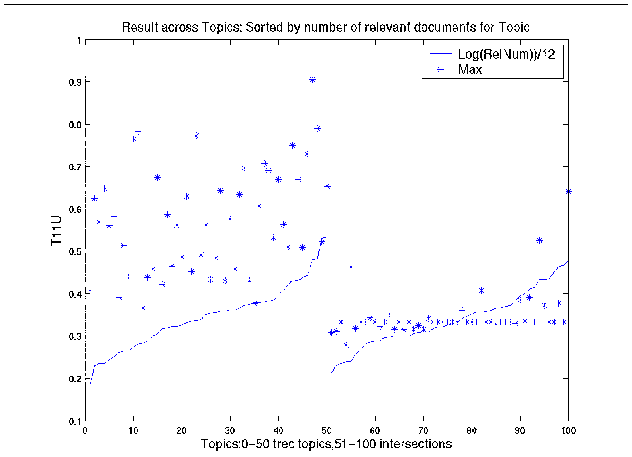

Figure 2: TREC annotators create the first 50 topics, and the last 50 topics are created by intersection of Reuter's categories.

3. Named-Page Finding

The language modeling approach to information retrieval typically makes the assumption that the query is representative of the relevant documents. In most previous research, language modeling places equal weight on all parts of the document. This may work well for ad-hoc document retrieval on newspaper corpora, but we do not feel that this necessarily makes the best use of information present in the document. For example, it does not leverage document structure, such as markup present in HTML documents. For named-page finding we hypothesize that the user's query is what the user believes to be a reasonable estimate of the "name" of the page she is seeking. Therefore, when estimating a language model, we do not want to estimate the language model of the entire document, but instead we want to estimate a model for the page's "name". Given some document structure, we can form a variety of document representations. We can weight these representations according to how characteristic they are of the page's "name". For example, we may want to weight the title of a document more than the rest of the text. Or in a hypertext environment, such as the Internet, we may want to incorporate the text of the links pointing to the page for which we are estimating a language model.

Language modeling suggests that we try to estimate a new language model from language models created from the document representations. This new language model for a document should be designed so that it closely models what we would expect a user to write as a query when requesting the document. For named-page finding, we would like to estimate a language model for the "name" of the page from language models produced by various document representations. We explore creating this new language model by taking a linear interpolation of the other models. Note that this is different from doing a linear combination of scores from different systems; we directly estimate the probability of a word given the differing language models. This is also different from directly weighting the term frequencies. Isolating the fields into different language models allows for smoothing each representation with a collection language model based on that representation only. This explicitly models the fact that the language usage in different document representations or fields is different, and adjusts the probabilities accordingly.

As mentioned above, we form language models from different document representations of the document. For example, one representation of a document may be its title. Another document representation may consist of the text contained in larger fonts. These representations do not need to be partitions, or even non-overlapping. We can still use the entire content of a document as a representation, while including other representations such as the document's title. From these different representations, we form a language model, using like representations from other documents for the backoff language model. We combine the language models from different representations using linear interpolation to form a new language model. The representations used and the linear interpolation weights chosen can be fine-tuned to a specific task. For example, we may expect that the document's title is more important for named-page finding than it is for a general ad-hoc relevance task. If so, then for named-page finding we would use a higher weight for the language model formed from the title than when performing an ad-hoc relevance search.

In this report, we investigate combining document representations to improve retrieval performance for named-page finding. We also explore how much information is needed to achieve performance that is similar to using the full document and in-link text available.

3.1 Language Modeling for Named-Page Finding

Two primary models for ranking documents are used in Information Retrieval: Kullback-Leibler divergence [15] and the generative language model [10]. Under common assumptions we make for this task, these approaches are equivalent [9]. So we arbitrarily choose to use a generative language model.

$$\mathrm{P}(Q|D) = \mathrm{P}(Q|\theta_D) = \prod_{w \in Q} P(w|\theta_D)$$

In the above equation, Q is a query (a sequence of words), D is a document, and θ is a language model. For the generative model, the query is treated as a sample from the document's language model, and we must then compute the probability that the document's language model produced the query sequence. Note that the probability of the term is in the product as many times as the term occurs. That is, we represent the query as a sequence of words.

To complete the specification of the retrieval method described above, we need to estimate the language models. The methods we use here are discussed and compared in [16]. The simplest way to estimate a unigram language model given a chunk of text is to use a maximum likelihood estimate:

$$P_{MLE}(w|\theta_T) = \frac{count(w;T)}{|T|}$$

Here, T denotes the text we are using to estimate the language model. The probability of the word given the text's model is simply the number of times the word occurs in the text divided by the length of the text. This has the advantage of being easy

to compute, but it has the problem that many words have zero probability or are poorly estimated if the length of the text is small. This technique is often used to estimate a background language model, such as the probability of a word given the entire collection. The collection is typically large enough to give estimates. To address the problem zero counts, linear interpolation is often used.

$$P_{LIN}(w|\theta) = \sum_{i=1}^{k} \lambda_i P(w|\theta_i)$$

Where k is the number of language models we are combining, and λ_i is the weight on the model θ_i. To ensure that this is a valid probability distribution, we must place these constraints on the lambdas:

$$\sum_{i=1}^{k} \lambda_i = 1 \quad \text{and for } 1 \le i \le k,\ \lambda_i \ge 0$$

We often use a linear interpolation of a text model and a collection model. One specific form of this is to use Dirichlet prior smoothing. This technique has worked quite well in ad-hoc retrieval experiments [16][9][14]. Dirichlet prior smoothing has one parameter, μ, which is typically chosen close to the average length of the text chunks being estimated. For this smoothing method, we have θ_1 as the text model, θ_2 as the collection language model, $\lambda_1 = |T|/(|T| + \mu)$, and $\lambda_2 = \mu/(|T| + \mu)$.

A final technique we will use to estimate a language model is a character based n-gram model. An n-gram model considers the context of the token. Specifically, it estimates the probability of the token given the previous n - 1 tokens.

$$P(c_i | c_{i-n+1} c_{i-n+2} \ldots c_{i-1}, \theta_T) = \frac{P(c_{i-n+1} c_{i-n+2} \ldots c_{i-1} c_i | \theta_T)}{P(c_{i-n+1} c_{i-n+2} \ldots c_{i-1} | \theta_T)}$$

The n-gram model does not specify how the probabilities on the right hand side of the above equation should be estimated, but they can be estimated using any of the previously described techniques. Using backoff where the background models are based on shorter sequences of tokens is a common method for estimating these probabilities.

3.2 System Specifics

We use the Lemur toolkit [7] for document indexing and retrieval. For document tokenization we used Inquery's stopword list and the Porter stemmer. The URLs were tokenized on punctuation (., /) and were not stemmed. A shorter stopword list was used for URLs ("http", "www", "com", "gov", "html", etc.). Each document had as many as seven document representations, outlined in Table 2. For every representation except the URL, we formed language models using a backoff model with Dirichlet prior smoothing. The Dirichlet prior parameter was chosen to be close to twice the average length of the representation. The probability of a word given the document's URL was computed treating the URL and word as a character sequence, then computing a character-based trigram generative probability. The numerator and denominator probabilities in the trigram expansion were estimated using a linear interpolation with the collection model (all URLs in the corpus).

The linear interpolation parameters for the .GOV corpus were trained using 80 queries we created locally for the named-page finding task. We trained the lambda parameters by performing retrieval separately on each of the representations. The weights were then taken as the scaled mean reciprocal rank of the system. The normalization was performed to ensure that the weights summed to one. This training procedure did not yield better results than assigning equal weight to each representation.

3.3 Named-Page Finding Results

We briefly describe experiments on the WT10G testbed (Table 5). Without the use of document priors, our system has respectable performance. This technique is as strong as any reported in TREC10 that did not use a form of document prior [3]. The best performing single document representation, document in-link text, had a MRR of 0.515, so combining the document representations significantly improves performance. Additionally, we tried using the document URL priors described in [4] as a re-ranking strategy for the top 1000 documents. The use of document priors did improve performance for all evaluation measures used for the task.

Configuration	MRR	% TOP 10	% FAIL
Equal lambdas	.676	83.4	5.5
Equal lambdas + URL length prior	.799	91.7	3.4

Table 5: Results of the homepage finding task on the WT10G testbed

Representation	Description	MRR	% TOP 10	% FAIL
Alt	Image alternate text	.194	28.0	66.7
Font	Changed font sizes and headings	.191	25.3	68.0
Full	Full document text	.469	66.7	16.7
Link	In-link text	.455	58.0	32.0
Meta	Meta tags (keyword, description)	.144	21.3	75.3
Title	Document title	.407	56.0	35.3
URL	Character trigram on URL	.131	19.3	68.7

Table 6: Performance of individual document representations on the named-page finding task

Run	Configuration	MRR	% TOP 10	% FAIL
LmrAllEq	Alt+font+full+link+meta+title+url (equal parameters)	.676	88.0	3.3
LmrAllEst	Alt+font+full+link+meta+title+url (trained parameters)	.667	86.7	3.3
LmrSmall	Link+font+title	.589	73.3	16.7
LmrDocStruct	Alt+font+full+meta+title	.567	72.7	15.3
LmrNoStruct	Full+link	.611	84.0	8.7

Table 7: Official results of the named-page finding task on the .GOV testbed

For this year's TREC, we wished to investigate two main questions: whether we get good performance gains by combining document representations, and what performance we can get when operating under a variety of system constraints. First we look at the performance of the individual document representations. This is in Table 6. The full text, in-link text, and title text were the best document representations for the named-page finding task; full text yielded the greatest performance with a mean-reciprocal rank of .469.

Our official results are summarized in Table 7. Two of our official submissions combined all of the representations: LmrAllEq and LmrAllEst, which respectively had mean-reciprocal ranks of .676 and .667. LmrAllEq used equal weighting parameters and LmrAllEst used parameters from our naïve training procedure evaluated on our 80 query training set. Both runs had much better performance on all measures than any of the individual methods. LmrAllEst performed slightly worse than LmrAllEq, but we are not sure whether this difference is significant.

The other three runs investigated combining document representations under different scenarios. LmrNoStruct used only the full text and the link text. The name is a little misleading, as it did keep the full text and link text separate as different language models. Combining these two best representations yielded a MRR of .611, which is not quite as good as combining all of the document representations. LmrSmall was an attempt to estimate what level of performance can be maintained while indexing only small amounts of text. Only the font, title, and link representations were used for this run. This resulted in indexing only 43 million terms, where the full text index contains around 945 million terms. The MRR of .589 for this run was not bad, but the failure rate at 50 documents was quite high, and the number of topics with the right answer in the top 10 was also lower than for other runs. The other run LmrDocStruct, looked at document representations from the document only, ignoring the URL and the in-link text. Incremental indexing of in-link text can be a complicated operation, and it may be too expensive to scan the entire corpus on a regular basis to compute each document's in-link text. This run was an attempt to measure how well a system could perform without the use of this information. The MRR for this run was .567, which suggests that in-link text is an important document representation for named-page finding.

3.5 Conclusions

We explored the use of document structure in the named-page finding task and the homepage finding task. We found that combining different document representations worked very well within the language modeling framework. We feel that the use of language modeling provides an effective mechanism for combining information from different document representations. We found that document in-link text is important for named-page finding, confirming previous results in homepage finding. We also demonstrated that good MRR performance can be achieved with a very small index, but that in order to get a high percentage of documents found in the top ten answers and to preserve a low failure rate, the full text of the

document is needed. We also showed that by adding more representations, we can improve performance, even though the content of the new representations may overlap with other existing representations.

The largest issue that we failed to adequately address was the training of the linear interpolation weights for the combination of document representation language models. The training method we used did not seem to provide any gain. We would like to explore more sophisticated techniques for training the parameters in the future.

4. Novelty Track: Finding Relevant and Redundant Sentences

The problems of finding relevant and redundant sentences are related to several other well-studied IR problems, with important differences. Finding specific relevant sentences is similar to some aspects of open-domain question-answering, in that we are looking for specific statements, and not just entire documents, that satisfy the query. We therefore include more surface features of sentences, such as punctuation and named entity types, in our analysis. However, the nature of the answers is much less specific than that typically seen in open-domain QA applications. Another similar problem is that of topic-level novelty and redundancy detection, as described by Zhang et al. [20], in which the authors use statistical models to perform adaptive filtering to find documents that are not only relevant, but also novel (or equivalently, not redundant). The novelty problem is also related to multi-document, query-specific summarization, in that we seek to find a set of representative, maximally informative sentences. However, the criteria for "maximally informative" are different for each problem: summarization seeks to obtain good coverage of the various aspects of the relevant information while keeping within a size/length constraint. For our problem, we have no length constraint and the definition of "novel" is extended to allow more subtle differences between sentences.

Our general approach is to view both relevant and redundant sentences as simple statistical translations of the query. For performance reasons, we currently only apply this type of model to the redundant sentence computation, and use a tf.idf-based approach to obtain relevant sentences.

4.1 Finding Relevant Sentences

We examined the performance of tf.idf-based retrieval using sentences as "documents" and found that, with pseudo-relevance feedback, using all sentences with a non-zero score gave high recall of relevant sentences (typically 70-80%). However, not surprisingly, the precision was extremely low (usually 10% or less), so this led to the following method.

1. Retrieve a set of candidate sentences using straightforward tf.idf-based retrieval with query expansion, based on a query constructed from the TREC description
2. Extract a set of features from the resulting candidate sentences
3. Use these features to classify each candidate sentence and remove those that are more likely to be non-relevant.

The classification in step 3 can be based on one of several different methods. In this study, we examined these three:

Simple_Threshold: use the tf.idf score as the only feature, and apply a threshold;
Decision_Tree (DT): extract a much wider set of features and build a decision tree; and
Proximity: simple model using proximity to highly relevant sentences as the main criterion.

We treated each sentence as a separate document, and indexed all the sentences from all relevant documents using Lemur [7]. We created a query from the title, description, and narrative fields in the topic and use this to score each sentence using tf.idf weighting. We performed query expansion using pseudo-relevance feedback, using the top 10 terms from the top 20 sentences. This produced an initial "candidate list" of sentences.

Simple_Threshold Rule

The tf.idf scores were normalized so that the highest scoring sentence received a score of 1.0. This rule was mostly useful as a baseline, and as a way for finding highly relevant sentences for the Proximity rule.

Decision Tree Rule

In this approach, we extracted a "base set" of about 15 to 20 surface and semantic features from a document sentence. This base set was extended to include features based on a small history window of previous sentences and "difference" features based on the query sentences. All of these were then used to build a decision tree using C4.5 [11] to predict non-relevant sentences. The intent was to find the most highly discriminative features, and hopefully increase precision by removing sentences from the candidate list which were likely to non-relevant based on the decision of the classifier.

Proximity Rule

This method is a simple form of clustering, where we assume that most relevant sentences occur in close proximity to "highly" relevant sentences. Based on our examination of the data, we found that a high proportion of sentences with a high tf.idf score were relevant, but there was very little overall correlation between tf.idf score and relevancy. On the other hand, there was a high correlation between a sentence's relative distance to a highly relevant sentence and its relevancy. This rule uses the tf.idf results, but unlike the other two methods is not restricted to that list when looking for sentences. We scan all the sentences in a document, using the tf.idf scores and relative distances as input. The proximity model can indicate that a sentence is relevant even if it received a zero score in the tf.idf model.

Here is one example of a simple proximity model which calculates a relevance score $R(i)$ for the i-th sentence in a document. It uses a window of nearby sentences with indexes $\{i-L, \ldots, i+K\}$, and scores $\{S(i-L), \ldots, S(i+K)\}$, where $S(i)$ is the tf.idf score for sentence i, and K and L are small positive integers (typically 2 or 3).

$$R(i) = f(S(i), i) \cdot \sum_{j=i-L}^{i+K} g_j(S(i), S(j), i, j)$$

Here, the function f is used to weight the contribution of sentence i based on its score and, optionally, its position in the document. The function g_j models the pair-wise relationship between sentences within the window, based on their scores and relative distances.

4.2 Finding Redundant Sentences

In this evaluation we focused on comparing *pairs* of sentences rather than more general *sets* of sentences. There are two main reasons for this. First, dealing with pairs is simpler and gives a good starting point before looking at the more general case. Second, when asked to find redundant facts in lists of sentences, most human assessors [20] (and we suppose, users) tend to focus on sentence pairs instead of complex subsets.

We view one redundant sentence as being a statistical translation of another. If we can build a good translation model in the language then we should be able to detect when two sentences are translations of the same thing. The task is simplified a little by the fact that the source and target sentences are in the same language. The methods we adopted for TREC use WordNet to estimate similarity for words and short phrases, and shallow parsing to help extract and compare sentence structures.

Given two sentences to compare, the algorithm has the following stages. First, we obtain parse trees for each sentence. For the TREC evaluation these were all pre-computed since parsing can be a time-consuming process. Second, we convert each parse tree into a graph that describes the modification structure of the terms. Third, we perform a simple graph matching algorithm that compares the terms from each sentence, weighted by their possible importance. The end result is a similarity measure that estimates how much one sentence is a translation of the other in the same language.

4.3 A Statistical Method for Estimating Word Semantic Similarity

Before looking at sentence structure, we estimate a similarity score for each word or short phrase. We do this by comparing the contexts in which they occur. We use a specialized set of contexts: those derived from each of the basic relation types available for a word or phrase in WordNet.

Given two words or short phrases to compare, we first construct a unigram model of the context for each word. Each unigram model is a linear combination of unigram sub-models. There is one sub-model for each relation we used from WordNet, which were synonyms, hyponyms, hypernyms, and coordinate terms. For some words, some of these submodels will be empty if no relation exists. Each submodel is built from the terms appearing in the word lists and, optionally, the glossary entries for that relation. A set of mixture weights is used to combine the probabilities from the submodels to calculate the final unigram model. The weights are trained from a training set of redundant, semi-redundant, and unrelated sentences.

If we visualize the enormous graph of words that comprise WordNet, each word has a set of synonyms and other related words. These together form a subgraph associated with that word. To compare two words we are essentially measuring the weighted overlap between their two subgraphs.

We compute the distributional similarity of the two overall unigram models using skew divergence. Skew divergence is described by Lee [6] and has the advantage of competitive predictive performance on statistical language tasks similar to ours, without requiring sophisticated smoothing. If $D(r \,||\, q)$ represents the KL divergence between distributions r and q, the skew divergence is:

$$s_\alpha(q, r) = D(r \,||\, \alpha q + (1-\alpha) r)$$

where α is a smoothing parameter and is set to α=0.99 in our application.

Once the skew divergence is calculated, the score must be normalized. This is done by calculating a similarity score relative to a small fixed set of "unfamiliar" words that are extremely unlikely to all be similar to the target word. The final score is a real number between zero (identical match) and an arbitrary upper bound of 500 (maximum dissimilarity). Table 8 shows scores for the word "astronaut" compared to various words. Note that words like "orbit" share similar co-occurrence distributions with "astronaut" but, correctly, do *not* get low translation distance scores.

astronaut	astronaut	0.000
astronaut	cosmonaut	0.002
astronaut	man	9.051
astronaut	explorer	14.372
astronaut	commander	20.877
astronaut	pilot	33.503
astronaut	traveler	49.312
astronaut	watermelon	153.548
astronaut	orbit	283.162
astronaut	rocket	294.722
astronaut	committee	302.601

Table 8: Semantic similarity "distances" of various words from the word "astronaut", as measured by normalized skew divergence of Wordnet-based unigram mixture models.

Using distributional similarity may be seen as a type of query expansion. Unlike typical scenarios for query expansion, the terms being compared are coming from documents already deemed relevant, so the same word found in two different sentences is less likely to be used with two very different senses, making sense disambiguation less of a problem. There are other methods described to estimate the substitutability of words, e.g. the confusion probability [2]. We do not explore those here; Lee does a comparison of some of these in a recent paper [5].

4.4 Sentence Parsing and Modifier Graphs

After obtaining word translation probabilities, we analyze sentence structure by obtaining a parse of all sentences to be examined either in the document set or in the TREC description and narrative fields. There are currently two purposes for this parse. First, we pre-process surface features to be used by the relevancy classifier, and second, we derive a dependency graph from the parse tree to estimate headwords and modifier relations, and the relative "importance" of terms in a sentence, both of are used our simple translation model. For the actual parsing we used the Apple Pie Parser, an easy-to-use corpus-based probabilistic parser written in C and developed by S. Sekine [13].

The algorithm for converting a parse tree to a dependency graph can be defined recursively, starting at the leaves of the tree:

1. A terminal node (leaf) depends only on itself and has itself as a headword.
2. Each possible type of non-terminal node has a rule to decide on the headword for that constituent, given the headwords derived from its leaves. The "winning" headword then becomes a new node in the dependency graph, with the "losing" headwords pointing to the new node and thus becoming dependants of the new node.
   For example, for the noun phrase NP → n_1 n_2 n_3 , we will choose n_3 (the last noun) as the headword, creating a new node for it, and creating edges pointing from n_1 and n_2 to n_3.
3. This process is continued up the tree until the root is reached.

With this dependency graph and the word similarity scores, we can compute the final translation probability of two sentences.

4.5 Graph Matching

We use a very simple graph matching step to model the fact that not all words in a sentence are equally "central". Some words or short phrases express the core ideas in a sentence, and other words act to modify them. Therefore, it seems reasonable to weight any matches between the core ideas in two sentences more highly than matches between other words. We currently define the graph weight of a node as its in-degree.

Our graph matching is currently "greedy": for each word W_i with graph weight A_i in the source sentence, we select the word $V(i)$ with graph weight $B_{V(i)}$ with lowest similarity distance $S_{i,V(i)}$ in the target sentence. This is constrained by a limit of at most K matches to any target word, where $K = 2$ in our implementation. Once a target word has reached its match limit, the

word with the next-lowest distance is used instead. The weighting factor for the i-th observation is $A_i \cdot B_i$, and so the matching score between the sentences is:

$$M(A,B) = \sum_{i=0}^{N} A_i B_{V(i)} S_{i,V(i)}$$

where N is an adjustable parameter (typically between 6 and 10) to be used when the source words are sorted in descending order of influence.

We show an example below for two sentences A and B taken from the TREC sample documents. The similarity score of Sentence A against Sentence B is the weighted mean described above: 14.821. Since this is below our threshold of 15, these sentences would be considered redundant.

Sentence A:

Some of the best shots, **released** this month by the US space agency **Nasa, show** parts of the universe billions of light years away - and therefore **billions** of **years** in the **past.**

Sentence B:

The images sent back this **year**, after astronauts repaired the telescope's defective mirror, **show** a **myriad** of astronomical objects too distant to be seen with the most **powerful** Earth-bound observatories.

Sentence A	**Graph weight A_i**	**Sentence B (Most similar word)**	**Graph weight B_i**	**Similarity Distance S_i**
past	6	year	4	2.456
years	3	year	4	0.0258
released	4	show	2	53.631
Nasa	5	powerful	1	68.240
show	2	show	2	0.000
billions	3	myriad	1	0.152
Weighted mean: 14.821				

Table 9: Comparison of word pairs from Sentence A and B, in order of influence. Only the top six word pairs, as sorted by Sentence A graph weight, are used for this example. Each word from Sentence A is paired with the word from Sentence B with the lowest similarity distance.

There is still plenty of work to do on the best features to represent in the graph, and the most appropriate, theoretically justified matching algorithm.

4.6 Novelty Track Results

We now give a brief summary of our official results, for both relevance and novelty, for the five runs we submitted. The best-performing runs are shown in boldface.

The runs are labeled as 'relevance algorithm + novelty algorithm'. The relevance algorithms Simple_Threshold, Decision_Tree (DT), and Proximity are those described previously. The novelty algorithms LowSameDoc and HighDiffDoc use the simple statistical translation approach. These labels refer to, respectively, a low threshold (sentences must be very similar in meaning) comparing sentences only within the same document, or a high threshold (more relaxed redundancy definition) comparing sentences only across different documents.

The official TREC scores we achieved for each run are shown in Table 10, which gives average precision (Ave P), average recall (Ave R), and average P·R scores. Our best average P·R score for relevance (0.058) was achieved with a simple tf.idf threshold on the ranked list of sentences. Our best average P·R novelty score (0.047) was achieved by selecting highly relevant sentences with the Proximity rule and then accepting all such candidates as novel, with our statistical approach performing marginally worse when applied to all sentences within the same document.

Run	Relevance			Novelty		
	Ave P	**Ave R**	**Ave P·R**	**Ave P**	**Ave R**	**Ave P·R**
Proximity + LowSameDoc	0.13	0.31	0.052	0.12	0.30	0.046
Proximity + HighDiffDoc	0.13	0.31	0.052	0.12	0.16	0.025
Proximity + All Novel	0.13	0.31	0.052	0.12	0.31	**0.047**
DT + LowSameDoc	0.10	0.13	0.019	0.10	0.13	0.018
Simple_Threshold + HighAllDoc	0.17	0.23	**0.058**	0.16	0.18	0.043

Table 10: Official Novelty Track Results by Run.

Comparative results, relative to the median P·R across all systems, are given in Table 11. The results are given as a fraction of the total number of queries (49). Because the scoring scale is continuous, we label as "at the median" any P·R score within ±0.01 of the median P·R score. Three of the runs used the same relevance algorithm (Proximity) and so these are collapsed into one entry. Overall, 4 out of 5 of our novelty runs had more than 50% of the scores at or above the median. Our best run, Proximity + All Novel, had 42.9% of scores above the median, and 91.8% of scores at or above the median.

Run (relevance + novelty algorithm shown)	**Below Median**	**At Median**	**Above Median**	**At or Above Median**
Relevance				
Simple_Threshold	0.367	0.347	**0.286**	0.633
Decision_Tree (DT)	0.694	0.265	0.041	0.306
Proximity	0.204	0.510	**0.286**	**0.796**
Novelty				
Simple_Threshold + HighAllDoc	0.286	0.388	0.327	0.714
Proximity + LowSameDoc	0.796	0.204	0	0.204
Proximity + HighDiffDoc	0.347	0.490	0.163	0.653
Proximity + All Novel	0.082	**0.490**	**0.429**	**0.918**
DT + LowSameDoc	0.469	0.408	0.122	0.531

Table 11: Comparative Novelty Track Results for Ave P·R scores, as a fraction of the total number of queries
Runs labeled "at median" have an average P·R score within 0.01 of the median.

4.7 Conclusions

The problem of finding *specific* relevant sentences seems quite difficult. Easier and perhaps more helpful would be to find precise *zones* of relevance which include more context. In any case, our simple proximity model gave better performance than a more sophisticated decision tree method. The decision tree did not include the same proximity model, so there is some chance that combining the two methods might give better performance: the representation may have made the difference here. The very high proportion of sentences judged as novel made it easy for the trivial "accept everything as novel" algorithm to do well. As a result, overall system performance on this track was dominated by the ability to find relevant sentences.

We described a word semantic similarity measure based on comparing word contexts from WordNet. Other query-expansion-type techniques, such as LSI might work as well or better. Wordnet is interesting because it allows some flexibility in how different similarity "features", such as synonyms, hyponyms, coordinate terms, and so on, are combined. Unfortunately, calling Wordnet and building language models is very slow, so pre-computing the LSI matrix might be a good compromise.

The greedy graph-matching approach is a first step in a direction we think is promising. It's clear that using only the in-degree of word nodes is not a sufficient indication of their importance in many cases. The current algorithm does tend to find good, similar sentences, but is still too tolerant of differences in the lesser-weighted areas of the graph. Among other things, named entities could use special treatment. With more work we think it should be possible to create a much more accurate alignment model for redundant sentences and passages.

5. Acknowledgements

This material is based on work supported by NSF grants IIS-0096139 and EIA-9983253. Any opinions, findings, conclusions, or recommendations expressed in this material are the authors', and do not necessarily reflect those of the sponsor.

6. References

[1] A. Arampatzis, J. Beney, C.H.A. Koster, and T.P. van der Weide. KUN on the TREC-9 Filtering Track: Incrementality, decay, and threshold optimization for adaptive filtering systems. In *Proceedings of Ninth Text REtrieval Conference (TREC-9)*, 2001.

[2] R. Grishman and John Sterling. Generalizing automatically generated selectional patterns. In the *Proceedings of the 15th. International Conference on Computational Linguistics*, Kyoto, Japan. Vol. II, 1994, pages 742-747.

[3] D. Hawking and N. Craswell. Overview of the TREC-2001 Web Track. In *Proceedings of the Tenth Text REtrieval Conference (TREC-10)*, 2002, pages 61-67.

[4] W. Kraaij, T. Westerveld, and D. Hiemstra. The Importance of Prior Probabilities for Entry Page Search. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2002)*, pages 27-34.

[5] L. Lee. On the Effectiveness of the Skew Divergence for Statistical Language Analysis. In *Artificial Intelligence and Statistics 2001*, pages 65-72.

[6] L. Lee. Measures of Distributional Similarity. In the *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, 1999, pages 25-32.

[7] The Lemur toolkit for language modeling in information retrieval. http://www.cs.cmu.edu/~lemur

[8] T. Minka. Bayesian Analysis of a Threshold Classifier, 2001. http://www.stat.cmu.edu/~minka/papers/minka-threshold.ps.gz

[9] P. Ogilvie and J. Callan. Experiments Using the Lemur Toolkit. In *Proceedings of the Tenth Text Retrieval Conference, (TREC-10)*, 2002, pages 103-108.

[10] J. Ponte and W. B. Croft. A Language Modeling Approach to Information Retrieval. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 1998)*, pages 275-281.

[11] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[12] S.E. Robertson. Threshold setting in adaptive filtering. In *Journal of Documentation*. In press.

[13] S. Sekine. The Apple Pie Parser. http://www.cs.nyu.edu/cs/projects/proteus/app/

[14] T. Westerveld, W. Kraaij, and D. Hiemstra. Retrieving Web Pages Using Content, Links, URLs, and Anchors. In the *Proceedings of the Tenth Text REtrieval Conference (TREC-10)*, 2002, pages 663-672.

[15] J. Xu and W. Croft. Cluster-based language models for distributed retrieval. In *Proceedings of the 22nd Annual International Conference ACM SIGIR on Research and Development in Information Retrieval (SIGIR 1999)*, pages 254-261.

[16] C. Zhai and J. Lafferty. A Study of Smoothing Methods for Language Models Applied to Ad Hoc Information Retrieval. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2001)*, pages 334-342.

[17] Y. Zhang, and J. Callan. YFilter at TREC9. In *Proceeding of Ninth Text REtrieval Conference (TREC-9)*, 2001

[18] Y. Zhang, and J. Callan. YFilter at TREC10. In *Proceeding of Tenth Text REtrieval Conference (TREC-10)*, 2002.

[19] Y. Zhang, J. Callan. Maximum Likelihood Estimation for Filtering Thresholds. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2001)*.

[20] Y. Zhang, J. Callan, and T. Minka. Novelty and Redundancy Detection in Adaptive Filtering. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2002)*, pages 81-88.

[21] Y. Zhang, W. Xu, J. Callan. Exact Maximum Likelihood Estimation for Word Mixtures. In *Text Learning Workshop in International Conference on Machine Learning (ICML2002)*.