

Automatic Discovery of Query-Class-Dependent Models for Multimodal Search

Lyndon S. Kennedy
Dept. of Electrical Engineering
Columbia University
New York, NY 10027
lyndon@ee.columbia.edu

Apostol (Paul) Natsev
IBM Thomas J. Watson
Research Center
Hawthorne, NY 10532
natsev@us.ibm.com

Shih-Fu Chang
Dept. of Electrical Engineering
Columbia University
New York, NY 10027
sfchang@ee.columbia.edu

ABSTRACT

We develop a framework for the automatic discovery of query classes for query-class-dependent search models in multimodal retrieval. The framework automatically discovers useful query classes by clustering queries in a training set according to the performance of various unimodal search methods, yielding classes of queries which have similar fusion strategies for the combination of unimodal components for multimodal search. We further combine these performance features with the semantic features of the queries during clustering in order to make discovered classes meaningful. The inclusion of the semantic space also makes it possible to choose the correct class for new, unseen queries, which have unknown performance space features. We evaluate the system against the TRECVID 2004 automatic video search task and find that the automatically discovered query classes give an improvement of 18% in MAP over hand-defined query classes used in previous works. We also find that some hand-defined query classes, such as “Named Person” and “Sports” do, indeed, have similarities in search method performance and are useful for query-class-dependent multimodal search, while other hand-defined classes, such as “Named Object” and “General Object” do not have consistent search method performance and should be split apart or replaced with other classes. The proposed framework is general and can be applied to any new domain without expert domain knowledge.

Categories and Subject Descriptors

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

General Terms

Algorithms, Performance, Experimentation

Keywords

Video Search, Multimodal Fusion, Query-Class-Dependent Models

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.

MM'05, November 6–11, 2005, Singapore.

Copyright 2005 ACM 1-59593-044-2/05/0011 ...\$5.00.

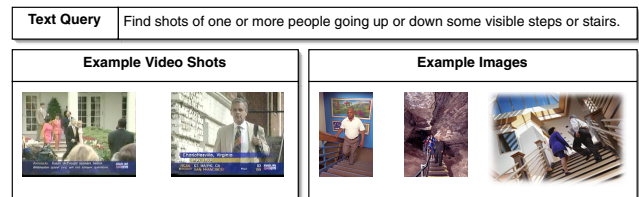


Figure 1: An example multimodal query for a video database, including text, example video shots, and images.

1. INTRODUCTION

In multimodal search applications, we are dealing with large sets of documents which contain information and cues in a number of different modalities. On the Web, we can think of the documents as being individual pages on the web, which are composed of information from several different modalities, such as the text they contain, the structures (such as titles, sections, and metadata) associated with that text, as well as the count and quality of other pages linking to them. In personal image collections, we can think of the documents as the low-level visual qualities of the images as well as metadata (such as keywords, dates, and sizes) associated with them. And in video databases, we can think of shots or segments of video as the documents, which are composed of information from a number of different modalities, such as low-level visual content of the images in the video stream, the qualities of the audio stream, and the transcript of the words being spoken (captured via automatic speech recognition (ASR) or closed captions).

In each of these situations in multimodal search, we are tasked with finding the most relevant documents in the index according to some query which is itself multimodal. In particular, in the case of multimodal search over a video database, we may want to issue a query with a set of keywords and a visual example (such as a shot or an image). An example of such a query is shown in Figure 1. A straightforward strategy for finding relevant shots for this document might be to issue two unimodal queries. The first would be a text search over the ASR transcript of the videos and the second would be a content-based, query-by-example image search over the images contained in the video stream. We would then need to come up with a fusion strategy for merging the results from unimodal searches into a multimodal ranking of relevance. We would quickly discover that there are limitations to fusing the unimodal searches using

a *query-independent* model, where the same fusion strategy is used for every query in order to optimize the average performance. The best fusion strategy for each given query is not necessarily the best fusion strategy for other queries and in certain cases the text search may be dominant while the image search only degrades the results (or vice versa). So, we would then benefit from developing a model wherein we choose the fusion strategy individually for each query to optimize the performance for that query alone: a *query-dependent* model. Of course, it's not feasible to know the proper fusion strategy for every possible query in advance and training for particular queries is counter to the untrained nature of the search task. To address this need, recent work has led to the development of *query-class-dependent* models, which classify queries into one of several classes. Search strategies are then tuned to optimize average performance for queries within the same class. So, each query within the same class uses the same strategy, but different classes of queries may have different strategies.

Query-class-dependent modeling can give significant improvement over query-independent modeling, but there is still the unanswered fundamental question of how, exactly, to define the classes of queries. In previous work [15, 3], the classes used have been human-defined, typically by examining some set of queries and identifying trends of semantic similarity that seem to be present. The classes discovered by these methods range from "Named Persons" to "General Objects" to "Financial" to "Sports." The results have confirmed the general benefits of query-class-dependent modeling over query-independent modeling, but they have not shown whether the hand-defined query classes are truly meaningful or optimal. It is unclear if they provide classes which have queries with similar optimal search strategies.

If the end goal of query-class-dependent models is to find query-classes wherein the optimal search fusion strategies for the queries within the class is consistent, then it seems that the definitions of query classes should have the constraint that each of the queries within a class has similar performances in the various unimodal search methods. To address this constraint, we propose a data-driven method for the discovery of query classes, in which we rely on a set of example queries with labeled ground truth relevance. We discover query classes by clustering in a "Performance Space," which is defined by the performance of the query in various unimodal searches, as well as a "Semantic Space," which is defined by the semantic content of the query. Once query classes are discovered, fusion strategies for each class are found and unseen queries can be mapped in semantic space to the class with the best fusion strategy.

We implement the model for the TRECVID 2004 video search task and find that we can increase performance from a MAP of 0.0605 (using hand-defined query classes) to a MAP of 0.0711 (using automatically discovered performance-based query classes). The results confirm the usefulness of query-class-dependent models and show that performance-based query classes outperform hand-defined query classes. The results also suggest that some hand-defined classes, such as "Named Person" and "Sports," do have consistency in performance, while other hand-defined classes, such as "Named Object" and "General" should either be split apart into subclasses or removed altogether and replaced with some other class, such as "Named Location," "Vehicles," or "Animals."

The unique contribution of this work is a framework for

automatically discovering query classes which have consistent performance across various unimodal search methods and, thus, consistent search method fusion strategies. The method is the first approach to use a performance-based strategy to automatically discover query classes. By including the semantic space, the framework can handle unseen queries, with no known performance features, and map them to performance-based classes. The framework can be deployed over any new task with little expert knowledge and can alleviate the errors in class selection caused by human factors. We have uncovered query classes which have not been considered in previous work and can motivate future research in video search.

In Section 2, we will discuss previous and our proposed framework. Our experiments will be discussed in Section 3. In Sections 4 and 5, we will discuss the results of the experiments and the conclusions that we can draw.

2. QUERY CLASS DISCOVERY

Having seen the limitations of query-independent fusion in multimodal search, some recent efforts in the TRECVID video search task, have implemented query-class-dependent models and have seen significant improvements.

A group from Carnegie Mellon (CMU) [15] defined five classes by hand: "Named Persons" "Named Object," "General Object," "Scene," and "Sports."

Another group from the National University of Singapore (NUS) [3] also developed a query-class-dependent model. Their model had six hand-defined classes: "Person," "Sports," "Finance," "Weather," "Disaster," and "General."

In both cases, fusion strategies are determined for each class (either by learning over a training set, or hand-tuning). The test queries could be automatically mapped to classes using some light natural language processing on the text of the query (since the classes of queries tend to have similar linguistic properties). The findings both confirm that the use of these query classes in a query-class-dependent model provides significant improvement over a query-independent approach. The work from both groups gives good reason to believe that there exists some way to classify queries where intra-class queries have very similar fusion strategies, so much so that the query-class-dependent model can approximate the performance of a query-dependent model.

There is an open question, however, of how to optimally choose these query classes. Inspection of the query classes provided by CMU and NUS shows that there is a significant human factor to the classes discovered. Only two classes, "Named Person" and "Sports," are shared between the two sets. Some classes, such as "Finance" and "Weather" from NUS, are not representative of the types of queries typically used: there are virtually no such queries in the TRECVID search task. And some classes, such as the "General" class from NUS, are too broad: approximately half of all TRECVID queries fall into this category. And finally, the classes given by NUS and CMU seem to be chosen mostly by the semantic similarity between the queries they contain. The end goal should be the discovery of classes that contain queries that are best served by similar unimodal fusion strategies, but the semantics-based methods employed by previous efforts do not guarantee that this will be the case.

2.1 Performance Space

In our work, we propose a framework for automatically

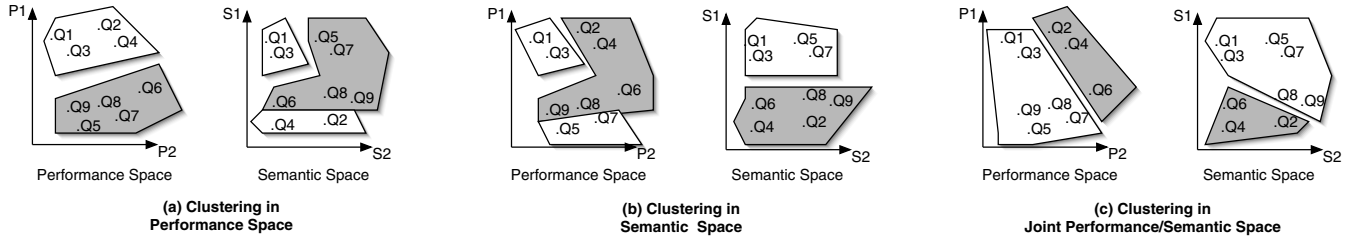


Figure 2: Conceptual view of the mapping of queries and clusters into performance, semantic, and joint performance/semantic spaces. Performance space (P_1, P_2) is defined in terms of the performance of two example unimodal search methods. Semantic space (S_1, S_2) is defined in terms of some features which can be extracted from queries in the absence of performance information. The Queries (Q) are shown mapped to their respective locations in each space. Clusters found in the various spaces are shown in white and gray boxes.

discovering classes of queries which have consistent performance in various unimodal searches and are thus more likely to have consistent intra-class fusion strategies. To achieve this class-consistency in performance, we discover the query classes by forming clusters of queries in *Performance Space*, which is defined for each query by the performance of various unimodal search methods on that query. Namely, each query, Q , is represented by the performance of all unimodal search methods, $P = \{P_1, \dots, P_N\}$, where P_i is the performance measure (e.g. average precision) of the search method i in answering query Q against some training corpus with ground truth. Queries near each other in performance space will have similar performances in the various unimodal search methods, while queries far away from each other will have very different performances. For example, “Named Person” queries tend to have high performance in text searches and low performance in content-based image searches and could be discovered as a useful query class using this method. On the other hand, “General” queries have highly variant performance across all unimodal search methods and would most likely not be discovered as a useful query class using this method. In comparison to hand-defined query classes, such as those defined by NUS and CMU, the query classes discovered by clustering in performance space should have more consistent within-class fusion strategies, be more representative of the query classes needed for the task, and less susceptible to errors induced by human factors. The method can then also be deployed in new domains without requiring experts to define new classes of queries.

2.2 Joint Performance/Semantic Space

In later experiments, we will find that while performance space clustering can discover classes of queries which can improve the performance of query-class-dependent models, the amount of improvement is limited by our ability to select the correct query class for each new query. New queries coming into a system do not have any known ground truth relevance information and it’s impossible to predict the location of the query in performance space. We therefore need to introduce a space to describe incoming queries.

Typically, the only information available at query time (in the TRECVID task in particular) is a natural language textual statement of the query and some example images. We can therefore describe queries in terms of a *Semantic Space*, which can be composed of the characteristics of the query such as the occurrence of named entities, parts of speech, and senses of words in the text. Namely, each query, Q is represented by a semantic description vector, $S = \{S_1, \dots, S_N\}$, where S_j are the linguistic features ex-

tracted from Q . In fact, we can think of the hand-defined queries from previous work as being defined in this semantic space, without knowledge of the performance space.

The semantic information about the queries can be leveraged in a number of ways. A first attempt might be to discover query classes by clustering in performance space and then attempt to learn a mapping between the performance space clusters and the semantic space. A more useful solution might be to temper the performance space clustering with constraints for semantic similarity as well as performance similarity during clustering. This can be achieved by forming a *Joint Performance/Semantic Space*. The joint performance/semantic space is simply composed of the combination of the unimodal search method performances (the performance space) with the query term features (the semantic space). The distances in the performance and semantic spaces can be measured using different metrics and can be combined through weighted summation to arrive at the joint performance/semantic space.

Figure 2 conceptually shows the positioning of queries and discovered clusters in semantic, performance, and joint performance/semantic spaces. We see that queries are mapped differently in performance and semantic spaces. Discovering query classes through clustering in performance space alone can cause the classes to be ill-formed in semantic space, making it difficult to map new queries to the correct class. Defining query classes through semantic space alone (like in hand-defined schemes) can lead to clusters which are ill-formed in performance space, causing difficulties when trying to choose the best fusion strategy for each class. Discovering the query classes through clustering in joint performance/semantic space can lead to classes with consistency in both performance and semantic space, allowing for clear choice of fusion strategy and easy mapping of new queries.

2.3 System Architecture

Figure 3 shows an overview of the system architecture for our framework. The system consists of two major components, one for training and another for testing.

During training, we rely on a collection of queries with ground truth relevance labels. We use any number of unimodal search methods, run them against the queries, and evaluate the results against the ground truth to determine the performance space for each query. We then run some light natural language processing analysis on the queries to extract their semantic space features. The joint performance/semantic space is then used in clustering to discover the query classes. Optimal unimodal fusion strategies are then learned for each class of queries. The discovered query

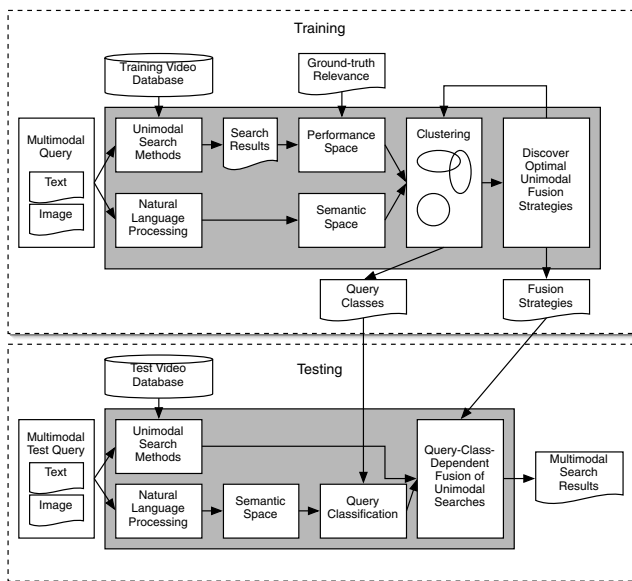


Figure 3: Overview of framework for query class discovery by clustering in joint performance/semantic space (in the “Training” phase) and the application of the discovered clusters to a query-class-dependent model (in the “Testing” phase).

classes (along with their semantic-space characteristics) and the optimal fusion strategy for each class are then passed along to the testing component.

During testing, we take new queries, which have unknown ground truth and apply the query classes and fusion strategies learned in training to score the relevance of video shots in the test database. Each new query is processed to extract its location in semantic space, which is then used to select the proper class. The unimodal search methods are run, the results are fused according to the rules for the given class, and the combined multimodal result is returned.

3. COMPONENTS AND EXPERIMENTS

We conduct experiments to verify the utility and performance of our query-class-dependent model and query class discovery framework using the NIST TRECVID 2003/2004 corpus, which consists of more than 150 hours of news video from a series of 30-minute broadcasts from ABC and CNN collected throughout 1998. The corpus is divided into three major sections: the development and test sets from the 2003 evaluation and the test set from the 2004 evaluation. A reference automatic shot segmentation (with over 100,000 shots in total) is given along with the output from an automatic speech recognition system [5].

We evaluate the system against the TRECVID 2004 automatic video search task. In this task, we are given a set of 23 multimodal queries containing textual natural language statements of information need as well as example video shots and static images showing samples of the desired type of shot. Figure 1 is a real example query from the TRECVID 2004 search task. In the automatic video search task, the system must parse this query without any interaction from a human and rank shots in order of their relevance to the query. To train the system we develop against the test data from TRECVID 2003 using the 25 multimodal queries defined from that year’s video search task.

To bolster our set of example queries, we define an additional 143 queries with natural language text queries and example shots. We define these queries by studying a log of over 13,000 actual requests for footage from a video archive at the BBC in 1998. We filter through the queries by hand and choose only the queries which would be appropriate for an archive of news footage such as the TRECVID 2003 search set. We also make an effort to choose queries in the style of the TRECVID queries, which tend toward searches for visual concepts in the video stream rather than news topics specified by the speech of the reporter.

Once we have established the set of queries, we find visual examples for each query by browsing the TRECVID 2003 development set to find example video shots for each query. We then generate “pooled” ground truth relevance labels for each of the queries by running some search methods for each query against the set and evaluating only the top-ranked results. Shots which are human-evaluated to be relevant are counted as relevant shots. All other shots (human-evaluated to be irrelevant or not human-evaluated at all) are counted to be irrelevant. NIST uses this method to generate the ground truth when evaluating the search task. In total, we evaluate approximately 3000 shots for each query. We then drop queries which have no example images found in the development set or fewer than 5 total relevant shots in the search set. We are left with 89 new queries that we’ve defined with pooled ground truth. We merge this set with the 23 usable queries from TRECVID 2003, giving us a set of 112 total queries, which we use to discover query classes and train our query-class dependent model.

The validity of the learned classes is certainly sensitive to the types of queries found in the training set. To be sure that the training queries are representative of the testing queries, we use a very similar method to NIST for determining worthwhile queries (browsing the BBC log). The fact that the queries come from a real-world application also ensures that the classes learned are, indeed, useful for real-world video search. If, in other deployments of the framework, we were to discover that the query-class-dependent modeling was not outperforming the query-independent model, we could back off to the query-independent model, or re-tune the system to accommodate the types of queries that it is seeing.

3.1 Search Methods

The query class discovery framework relies on a set of unimodal search methods which we use to define the semantic space. In principal, these component search methods can be any reasonable ranking of document relevancy with respect to some aspect of the query. They need not be unimodal and could even be composed of various combinations of simpler search methods. In this work, we keep the search methods as fundamental as possible. This is an initial investigation and we are limited in the number of methods that we can reasonably use (we have only 112 queries to cluster and must keep the dimensionality of the performance space small).

We include six search methods in our performance space: four are variants on text search against the ASR transcript, one is a content-based image retrieval method comparing query images against keyframes of shots from within the search set, and the last is a specialized named person search method, which incorporates text search information as well as time distribution and face detection information. All have been shown to be useful for searching news video [15, 2, 3].

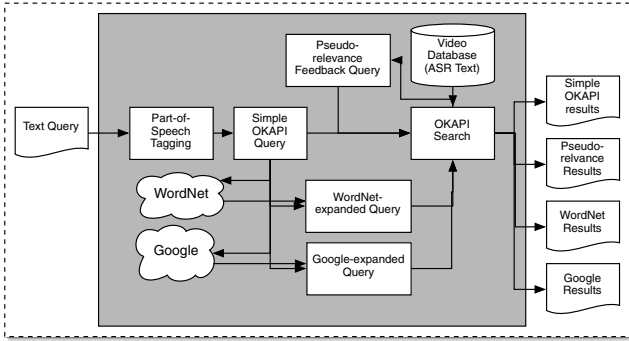


Figure 4: Overview of automatic text query processing. The natural language query is processed and expanded to give four unique sets of search results.

3.1.1 Text Retrieval

In text retrieval, we conduct text searches against the ASR transcript of the broadcasts using key words extracted from the natural language text queries. The text associated with each shot is the entire text of the story in which it is contained. A “story” is defined as a segment of the news broadcast with a coherent news focus and at least two independent, declarative clauses. It has been shown that stories are good segments for semantic associations between words and visual concepts [2]. For training, we use ground truth story boundaries as provided by the LDC. For testing, we use automatically-detected story boundaries. We compare the retrieval performance achieved using these automatic boundaries with the retrieval performance found using ground truth story boundaries. We notice only a small decrease in performance. So, automatic story segmentation is imperfect, but still usable for retrieval purposes.

The text is then stemmed using Porter’s algorithm [10] and stop words are removed. Retrieval is done using the OKAPI BM-25 formula [12] with key words automatically extracted from the natural language text queries. The manner in which the key terms are extracted from the text queries is the main point of difference between the four text search methods used. The query keyword extraction process is summarized below and in Figure 4. The keywords, extracted using the various methods for a few example queries, are shown in Table 1. We see that various queries have varying qualities of keywords found by each method, which has a direct impact on the performance of the various methods for search and indicates a need for a query-class-dependent model for properly choosing when to use each method.

3.1.1.1 Simple OKAPI.

In simple OKAPI retrieval, we just use keywords which we automatically extract from the text query to search against the ASR text associated with the videos. The text query is analyzed with a part-of-speech tagger [8]. Standard stop words are removed. Nouns are taken to be the query keywords. If there are no nouns in the query, then we back off to the main verb [3, 15]. The list of keywords is then issued as a query against the ASR index using the OKAPI formula. This simple keyword extraction scheme works well for queries which have well-pointed keywords already in the text query, such as queries for named entities.

3.1.1.2 Pseudo-Relevance Feedback.

In pseudo-relevance feedback, we operate under the assumption that the top-returned documents from a text search are mostly relevant and we can then just take the most frequent terms in those top documents and feed those terms back in to another search. This approach can be useful for identifying additional keywords for general object queries where some of the words in the original query can return some relevant documents, which contain clues on additional related words to search to give better results [3].

We begin by issuing the same query we used in simple OKAPI retrieval. We then analyze the top M returned documents to find the top N most frequent terms that appear in those documents. The discovered terms are then added to the query and search is repeated. M and N are chosen by an exhaustive search for the single combination that yields the best retrieval results on the training set in terms of mean average precision (MAP), which is defined in Section 4.

3.1.1.3 Query Expansion.

In query expansion, we look to add additional terms to the query which are related to the desired search term and give improved precision and recall. The strategy for query expansion involves using the simple OKAPI query against some knowledge resources to discover related terms. Query-expansion is a long-standing technique for enhancing information retrieval results which has been shown to be useful in video retrieval [3], particularly in cases where the exact terms contained in the query are lexically or semantically (but not exactly) related to the terms contained in relevant documents. We implement two query expansion systems using two knowledge sources: WordNet [9] and Google.

We use WordNet for query expansion by extracting all the hypernyms (general terms which also denote the specific term, for example, “vehicle” is a hypernym of “train”) and synonyms (specific terms which denote other specific terms, for example, “locomotive” is a synonym of “train”) for each of the terms from our Simple OKAPI query from the database. We add N terms to the query by constructing lists of related terms for each query term and iterating through each query term, adding the top related term to the query. N is set through experiments to maximize retrieval performance on the training set.

We use Google for query expansion by, again, issuing our simple OKAPI query to Google. Google then returns a list of ranked documents and we can examine the top M documents to discover the top N most frequent terms in those documents. The documents are part-of-speech tagged and only nouns are retained. Stop words are removed and only the remaining terms are analyzed for frequency. Those terms are then added to the expanded query. M and N are chosen to maximize performance on the training set.

3.1.2 Content-Based Image Retrieval

In content-based image retrieval (CBIR), we use the example shots and images from the query to find shots from within the search set which have similar visual characteristics in some low-level feature space. CBIR is a common approach for finding visually similar images in image databases. It works particularly well when the images being sought in the database have consistent visual features, compared with each other and the query image, which are distinct from other images in the database and can be very useful for ad-

Original Query	Simple OKAPI	PRFB	WordNet	Google
Find shots of Osama bin Laden.	osama bin laden	osama bin laden afghanistan taliban	osama bin laden container bank	osama bin laden usama fbi wanted
Find shots of pills.	pills	pills viagra pfizer	pills lozenge tab dose	pills prescription drug
Find shots with a locomotive (and attached railroad cars if any) approaching the viewer.	locomotive railroad car viewer	locomotive railroad car viewer germany crash wreckage	locomotive railroad car viewer engine railway vehicle track machine spectator	locomotive railroad car viewer steam engine power place locomotion

Table 1: Keywords automatically extracted from various queries using part-of-speech tagging (simple OKAPI), pseudo-relevance feedback (PRFB), query expansion via WordNet and Google.

dressings some of the queries in video search tasks. Shots in the search set and the query are represented by single keyframes from the center of the shot. Each still image is then represented in LAB color space and segmented into a 5x5 grid. The features representing each image are the first three moments of the distribution in each channel in each grid. Each query shot is then matched against each shot in the search set using the Euclidean distance. The score for each shot in the search set is then just the minimum distance between the shot and any one of the query images.

3.1.3 Person-X Retrieval

In previous works in the TRECVID search task, it has been shown that the retrieval of named persons has been the most important source of performance for a system, and so it has been common practice to build search methods specifically geared toward retrieval of named persons. Person-X search leverages text searches for the person’s name along with knowledge of the distribution of the time delay between the appearance of a person’s name in the ASR and the actual appearance of their face in the video stream. Detection of anchors and faces are also incorporated. The score for Person-X retrieval is then given as

$$P_X = \lambda P_T + (1 - \lambda)(\alpha P_t + \beta P_f + (1 - \alpha - \beta)P_{\bar{a}}) \quad (1)$$

where P_f is the probability of a face detected in the shot, $P_{\bar{a}}$ is the probability of the absence of an anchor detected in the shot, P_T is the probability of a text match existing between the person’s name and the text in the ASR transcript, and P_t is the probability of the shot being relevant, given its time distance from the nearest appearance of the person’s name in the ASR text. λ , α , and β are weighting factors, where λ and $(\alpha + \beta)$ are constrained to be between 0 and 1. The probability distribution for P_t , as well as the weighting factors, λ , α , and β , are learned over a training set. P_X is the final combined Person-X score.

Previous work has also included specific-person face detection, where example shots of faces are provided by a user or mined from the Web and eigenfaces is employed to rate the appearance of a specific person’s face. At this time we are only providing baseline measurements so we do not fully implement the specific face detection component.

3.2 Query Clustering

With the pool of training queries and the set of search methods, we set out to automatically discover meaningful query classes. The approach we take is to compute pairwise distances between each of the training queries, measured in performance and semantic spaces, and apply a clustering algorithm to discover the classes.

3.2.1 Performance Space

The performance space for each query is represented as a vector of performances in the six search methods defined in Section 3.1. Each of the search methods is run for each of the queries. The performance is measured in terms of non-interpolated average precision at 1000 returned shots, a metric used by NIST in the TRECVID evaluations, which approximates the area underneath the Precision-Recall curve. The various search methods used are predisposed to having different performances: text search is usually high and content-based image retrieval is usually low. To avoid having one performance dimension dominate the clustering process, we normalize the results from the various search engines to have similar dynamic ranges by subtracting the mean from each dimension and dividing by the variance. The performance vector for each query is then L1-normalized, which makes the performance metric in each dimension a measure of the relative usefulness of each search method for each query, rather than a measure of absolute performance. Experimentation shows that these normalization steps lead to better performance. Euclidean distances in performance space are then calculated pairwise between all queries.

3.2.2 Semantic Space

In our experiments, we will see that while performance-space clustering alone has the potential to improve performance, it is hindered by our inability to map incoming queries into the correct performance space clusters. To address this issue, we need to constrain the clusters to be consistent in the semantic space, which we can measure at query time without need for explicit performance space knowledge. We develop two methods for measuring distances in the semantic space. The first method, *Query Term Features*, calculates features based on a light natural language processing of the text query. The second method, *WordNet Semantic Distance*, uses WordNet to estimate a measure of semantic similarity between the terms in two queries.

With query term features, we calculate a 5-dimensional representation of each query based on counts of nouns, verbs, and named entities appearing within the text query. We run a part-of-speech tagger as well as a named entity tagger [1] against each text query and count the noun phrases, verb phrases, named persons, named locations, and named organizations contained within each query, leaving out query stop words, such as “find” and “shot.” Differences in dynamic range between dimensions are normalized to have unit variance (without shifting the distribution mean). This normalization step is shown to improve results in the query-class-dependent model. Distances between query term features are calculated pairwise between all queries using cosine

distance, which is simply 1 - the cosine similarity:

$$\text{cosine distance} = (1 - \cos \theta) = (1 - \frac{\mathbf{q} \cdot \mathbf{q}'}{\|\mathbf{q}\| \|\mathbf{q}'\|}) \quad (2)$$

where \mathbf{q} and \mathbf{q}' are the vectors of two queries in query term feature space and cosine distance is the pairwise distance between them. \mathbf{q} and \mathbf{q}' have zero cosine distance if the proportional distributions among dimensions are equal.

For WordNet semantic distance, we use the semantic similarity metric defined by Resnick [11], which fuses semantic similarity in WordNet with probabilistic corpus-based semantic similarity. The similarity between two terms can be determined via corpus-based techniques by counting co-occurrences for the pairs of words in some corpus. This approach leads to problems since incredibly large corpora are needed to gather statistically meaningful counts for all possible word pairs. Resnick’s technique overcomes this challenge by counting cooccurrences of hypernyms of words. So, in the case where pairwise counts for two terms are unknown, we can back off up the WordNet hierarchy and examine co-occurrences of hypernyms of terms. Specifically, the Resnick similarity between two terms is the information content of the lowest super-ordinate (lso, or lowest hypernym pair) for the terms which can be counted from the corpus:

$$\text{sim}_{\text{Resnick}}(t_1, t_2) = \log P(\text{lso}(t_1, t_2)) \quad (3)$$

where t_i are the terms and $P(\text{lso}(t_1, t_2))$ is the probability of observing the lowest super-ordinate of the two terms.

Since Resnick’s measure is a similarity, we use its inverse to measure the WordNet semantic distance. We measure the pairwise WordNet semantic distance for each individual term in the queries against all other terms in the queries. We can then measure the pairwise WordNet semantic distances between queries with the average of the word-level distances.

3.2.3 Joint Performance/Semantic Space

To leverage the power of both the performance and semantic spaces, we combine them into a joint space. The pairwise distances between queries in joint performance/semantic space can easily be calculated as a weighted sum of the pairwise distances in each of the performance and semantic spaces:

$$D = \alpha D_P + (1 - \alpha)(\lambda D_{QT} + (1 - \lambda) D_{WN}) \quad (4)$$

where D is the joint performance/semantic distance matrix, and D_P , D_{QT} , and D_{WN} are the distance matrices in performance space, query term space, and WordNet, respectively. α and λ are weights which are either set to 0, 0.5, or 1. In other words, we generate a number of different joint performance/semantic distance matrices in which particular dimensions are either entirely on, entirely off, or evenly averaged. Any weighting can be used, but for simplicity, we only explore these few cases.

3.2.4 Clustering

Given our pairwise distance matrices between queries, we can cluster the queries using any number of clustering algorithms (from K-means, to Normalized Cuts, to Hierarchical Clustering). We avoid normalized cuts since the clusters discovered by normalized cuts are not necessarily of the form that we would like to find. Namely, the clusters can have highly irregular shapes, and since we are looking for compact clusters with consistent performance across various search methods, the Normalized Cut method may hurt

more than it helps. K-means and Hierarchical Clustering are both well-suited for finding the sorts of tight clusters that we’re looking for. We choose Hierarchical Clustering over K-means, since Hierarchical Clustering gives us a better method for choosing the number of clusters. Varying the number of clusters in K-means can lead to wildly different clusters, while varying the number of clusters in Hierarchical Clustering leads only to splitting or merging various clusters. The Hierarchical Clustering is performed by forming a linkage tree by iteratively joining queries with their nearest neighboring queries (or clusters of queries). We can form clusters from the linkage tree by taking a horizontal cut across the linkage tree, adjusting the height of the cut in order to produce a particular number of subtrees. Each subtree forms a cluster consisting of all the leaf nodes which are children of that subtree. We choose the number of clusters by trying all numbers of clusters and picking the number which maximizes retrieval performance (in terms of mean average precision) on a held-out validation set.

3.3 Search Method Fusion

After query classes are discovered, we need to find optimal fusion strategies for combining the various search methods in each class. For this initial study, we use linear weighting of the scores from each of the unimodal search methods:

$$\text{Score}(c)_M = \sum_i \lambda_i(c) \text{Score}_i \quad (5)$$

where Score_i is the score of each individual search engine, λ_i is the weight for the individual search engine, and $\text{Score}(c)_M$ is the combined multimodal score. The weights, λ_i , are conditioned on c , the class of the query.

The weights, λ_i are found through a full grid search of all possible combinations, where each weight is quantized to 11 values between 0 and 1 (0, 0.1, 0.2, etc) and it is constrained that all weights sum to one. The combination of weights for each query class which gives the best performance for that class in the training set is retained and used in the test application. The scores, Score_i , can be estimated from the raw scores, Score_{r_i} (the outputs from each unimodal search method) using rank-based normalization.

For rank-based normalization, we essentially ignore the raw scores for each shot given by each of the search methods and only evaluate based on the position of each shot in the ranked list of scores [15]. This eliminates the need to scale disparate scoring metrics and helps smooth large quantizations that happen when using story-based grouping of shots, like we do with our text search. (With story text search, all of the shots in a story get the same raw score and the difference in scores between stories may be large, making it difficult to gain anything from content-based image search. Rank does not allow shots to have the same score and smoothes out the large gaps between stories). The rank-normalized score is calculated as $\text{Score}_{ij} = 1 - R_{ij}/N$, where R_{ij} is the location of the shot j in the ranked list returned from search method i . N is the total number of shots.

3.4 Query Classification

At query time in an application situation, we need to properly choose the best class for each incoming query and employ the optimal fusion strategy for that query. With these incoming queries, we have no advance knowledge of search engine performance and we only have the semantic space

	Method	Baseline O	Classification SD	SVM
Query-Independent	Text	0.0595	-	-
	Multimodal	0.0595	-	-
Hand-defined Classes	CMU	0.0605	-	-
	NUS	0.0598	-	-
Automatically Discovered Classes	P+Q+WN	0.0748	0.0673	0.0359
	P+Q	0.0749	0.0711	0.0478
	P+WN	0.0745	0.0645	0.0421
	P	0.0769	0.0172	0.0320
	Q+WN	0.0609	0.0598	0.0465
	Q	0.0615	0.0602	0.0533
	WN	0.0599	0.0590	0.0544

Table 2: Summary of performance (in Mean Average Precision) on the TRECVID 2004 search test set using various query-class-dependent models. Baselines are query-independent models using text-only and multimodal searches. Automatically discovered classes are shown with performance in three classification scenarios: Oracle (another baseline: the best class is known ahead of time, denoted by “O”), Shortest Distance (“SD”), and Support Vector Machine (“SVM”). Classes are automatically discovered in variants of joint performance/semantic space, in the presence or absence of the three subspaces: P (performance space), Q (query term space), and WN (WordNet space).

coordinates of each query to aide us in our decision. In this work, we evaluate two methods to make this decision: shortest distance in semantic space and a support vector machine (SVM) using the semantic space kernel.

3.4.1 Shortest Distance

Using shortest distance in semantic space is straight-forward and quite powerful in our case, since the classes we find are constrained to form reasonable clusters in semantic space. To measure the distance between an incoming query and a class of queries, we take the average distance in semantic space between the query and each of the queries within the class. We then choose the class of the query to be the class with the shortest distance in semantic space.

3.4.2 SVM

A potentially powerful method for classification is the support vector machine [14]. SVMs learn optimal hyperplanes for binary decisions in some high-dimensional space given only a kernel matrix for that space. Traditionally, kernels such as linear, polynomial, and radial basis functions have been used as SVM kernels. Recent research, however, has shown that any kernel matrix can be used in SVMs, as long as the matrix is positive semi-definite and represents a plausible distance between examples [6, 7]. We can use our semantic space distance matrix, given by the combination of the distances in query terms space and WordNet, and simply plug it into an SVM. Since SVMs are binary discriminators, and our application requires a multiclass classifier, we need to extend the SVM to a multiclass tool [13], which can be done using error-correcting output codes [4]. In our analysis, we find that the SVM does not perform as well as simply using the shortest distance. This is most likely due to the small size of our training set (only 112 examples).

4. ANALYSIS OF RESULTS

We perform the query class discovery over our training set of queries and apply the learned clusters and fusion strate-

gies to the TRECVID 2004 search task. The results, expressed as Mean Average Precisions (“MAP,” or the mean of the average precisions for each of the 23 queries in the evaluation set), are summarized in Table 2. The results confirm that query-class-dependent models improve upon query-independent models and that classes discovered by clustering in joint performance/semantic space can improve upon classes defined by hand. The results also show that the choice of space for query clustering has significant impact on our ability to map to the correct class for test queries. The performance is discussed in more detail in Section 4.1.

Analysis of the classes discovered by clustering in joint performance/semantic space confirms that some of the query classes that have been defined by hand in previous efforts, such as “Named Person” and “Sports” are, indeed, useful classes, while other classes, such as “Named Object,” can be split into subclasses and other classes, which have not been used before may be helpful. The qualities of the classes discovered are discussed in more detail in Section 4.2.

We have experimented with number of clusters. In these analyses, we only discuss the best case, which uses eight.

4.1 Performance

We use two query-independent fusion strategies as baselines: text-only (using only simple OKAPI search) and query-independent multimodal fusion (“Text” (MAP: .0595) and “Multimodal” (MAP: .0595) in Table 2, respectively). We find that query-independent multimodal fusion cannot outperform text-only queries, which confirms the need for query-class-dependent models to fully leverage the power of content-based image searches and textual query expansion.

We then test our fusion strategies against the hand-defined classes from the prior work of CMU and NUS (marked “CMU” (MAP: .0605) and “NUS” (MAP: .0605)) in Table 2. We learn optimal fusion strategies for each of query class in our training set and apply them in the test case. Both groups present rule-based algorithms, which can classify the incoming queries nearly perfectly, so we assume errorless classification. We see that these query-class-dependent models give some improvement over query-independent models.

We then apply our query class discovery framework over the training set to discover query classes. We experiment with several variations on the joint performance/semantic space for clustering queries and also with several methods for classifying incoming queries. The joint performance/semantic space is formed using all possible combinations of the performance space, “P,” the query term space, “Q,” and the WordNet space, “WN.” For each space, we also experiment with different classification methods for assigning classes to incoming queries. The “O” classification method is the “Oracle” method, which is not a real classification method at all, but a baseline method in which we explore the limits of the discovered classes by assuming that we can automatically pick the best query class for each incoming query. The “SD” (Shortest Distance) and “SVM” (Support Vector Machine) classification methods are real classifiers, which were discussed in Section 3.4

We see that using the performance space to conduct query class discovery provides the best potential for overall performance, as evidenced by the “P” method with Oracle classification (MAP: 0.769). We see, however, that it is impossible to use real classifiers to recover the proper class for incoming queries, unless we incorporate the semantic space into

our query class discovery framework: the classes discovered in joint performance/semantic space have only slightly degraded potential for performance and can be better recovered at query time, with a maximum MAP of .0711. Clustering in semantic-space alone gives performance similar to the hand-defined clusters found by CMU and NUS, with MAPs ranging from .0590 to .0602, which makes sense since those clusters were essentially found by human inspection of queries, looking for common semantic groupings.

The quality of the classification schemes is difficult to evaluate empirically: it’s unclear what the “true” class membership of each incoming query is. Various classes can still have similar fusion strategies, so it’s unclear what penalties a “misclassification” of an incoming query will actually have on the end performance. We can, however, look at the best-case performance from classification using the Oracle baseline. We see that in joint performance/semantic clusters, like “P+Q,” we are able to map to classes which are close to the best-performing class by using the shortest distance classification method. In performance-only clustering, “P,” we are unable to map to useful classes in any meaningful way at query time. This is due to the lack of cohesion in semantic space found in the pure performance clusters.

In almost all cases, the SVM classification fails to map to useful classes, which is most likely due to the small number of training examples available. Interestingly, the SVM classification provides significant improvement over shortest distance in performance-only clustering (MAP: .0320 vs. MAP: .0172). This is probably since the divisions between classes in semantic space in this clustering approach is highly non-linear (since there is no constraint on semantic space similarity in performance space clustering) and SVMs are particularly adept at learning such complicated decision boundaries. This also gives hope that, given enough training examples, we may be able to exploit the power of SVMs for classification, uncovering some nonlinear mapping between the performance space and the semantic space, which would allow us to discover clusters in pure performance space, leaving the decision on semantic mappings up to the classifier.

Figure 5 shows the mean average precision of each of the methods discussed on the TRECVID 2004 video search task using shortest distance classification (in red) against all official TRECVID 2004 runs (as scored by NIST). We see that while we are able to show improvement by automatically selecting the query classes, we are still inhibited by the quality and number of our component search methods as well as our search method fusion strategy. In this work, we focus on the evaluation of the effects of automatic query class discovery, and we leave the tasks of exploring better search methods as well as better fusion strategies to future work.

4.2 Discovered Classes

Figure 6 shows some sample queries from the classes automatically discovered by clustering in joint performance/semantic space. The queries are segmented into clusters and each cluster shows only a few example queries. (There are 112 training queries in total). The blocks to the left of each query show the performance space mappings for the query. Each block gives the relative performance for the query in each independent search method. The bright blocks indicate high performance, while the dark blocks indicate low performance. The performances are normalized for each query, so the performances indicate which methods are most helpful

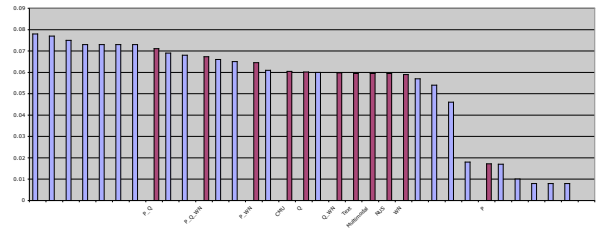


Figure 5: Performance (MAP) of our methods (red) and all official TRECVID 2004 automatic search task submissions (purple).

and least helpful on a per-query basis. The figure shows the six classes (out of the total eight), which have interesting interpretations and consistency in performance space. The remaining two classes have no easy interpretation and are quite similar to the “General” class described by NUS.

The first cluster contains virtually all of the **Named Persons** in the training set. The queries all also seem to have consistently high performance across the various search methods, except for content-based image retrieval. Simple OKAPI and person-X searches are most helpful, pseudo-relevance feedback and query expansion seem to slightly degrade performance, and content-based image retrieval does not help at all. This cluster seems to confirm the usefulness of the “Named Person” classes defined both by CMU and NUS, since these classes are tightly-clustered in performance space.

The second and third clusters both contain many **Named Objects**. There is a distinction between the two classes, however, in performance space. In the second cluster, text-based searches seem to be the most useful, while content-based image retrieval doesn’t help at all. In the third cluster, the case seems to be quite the opposite: content-based image retrieval is much more accurate than text searches. Examining the semantics of the queries seems to show that the queries in the third cluster are more graphical (such as the Dow Jones financial screen or the Siemens logo), while the queries in the second cluster seem to be more or less specific scenes or locations (such as the Sphinx or the New York City skyline). These two clusters seem to confirm the need for a “Named Object” class, as suggested by CMU, but they also seem to indicate that this class could be split into two subclasses (**Named Object** and **Named Location**) to give more cohesive classes in performance space.

The fourth cluster presents an interesting group of queries with various meanings. Most interestingly, many of the queries related to **Sports** and **Vehicles** end up in this class. In performance space, these queries all seem to perform similarly in all search methods, which leads to the conclusion that for these queries, all search methods (including content-based image retrieval) are equally important. This cluster seems to support the validity of the “Sports” classes identified by both CMU and NUS, but also indicates that this class could be augmented to include other types of queries, such as “vehicle” queries, since they have similar profiles in performance space and benefit from similar fusion strategies.

The fifth cluster contains mostly **Animals** (along with some other queries) and gets the most performance from query expansion via **Google**. No such animal cluster has been identified by previous work, but it seems that queries for animals have a compact class cluster in performance space and would be well-served by similar fusion strategies.

The final cluster contains a number of queries which ben-

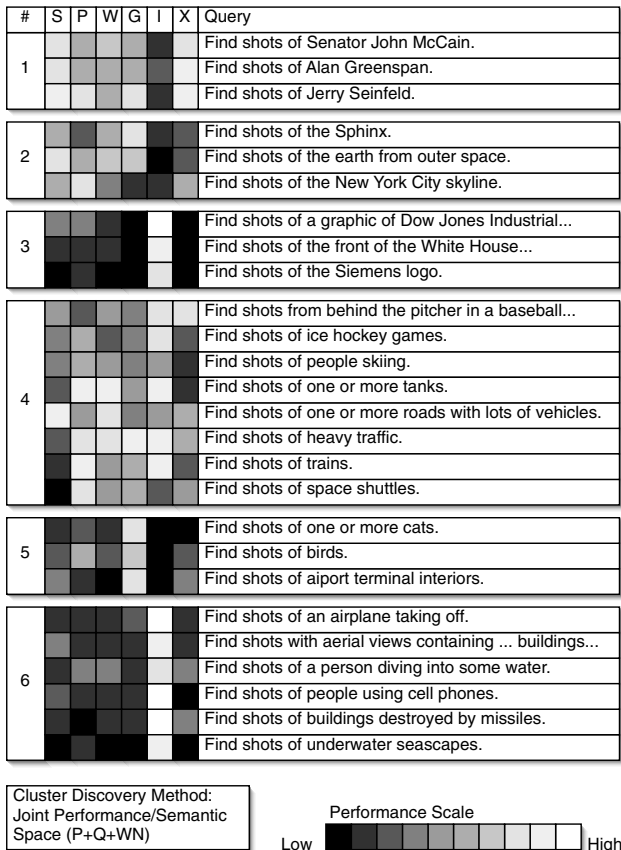


Figure 6: Sample queries from the clusters discovered in joint performance/semantic space. Class IDs are shown along the left side, with relative performance in each dimension in performance space to the right (“S” = Simple OKAPI, “P” = Pseudo-Relevance Feedback, “W” = Query Expansion via WordNet, “G” = Query expansion via Google, “I” = Content-based Image Retrieval, and “X” = Person-X), followed by the text of the query.

efit more from **content-based image retrieval** than any other search method. The queries contained within this cluster seem to be mostly scenes, but it is difficult to assign an understandable semantic label to this cluster. There is, however, some semantic relationship which can be recovered through semantic space classification, which demonstrates the power of clustering in joint performance/semantic space, particularly its ability to discover classes of queries which are useful for query-class-dependent retrieval.

5. CONCLUSIONS

We have developed a framework for automatically discovering query-class-dependent models for multimodal search by defining query classes through a clustering process according to search method performance and semantic features. We apply this framework to the TRECVID 2004 video search task. We confirm that query-class-dependent models can outperform query-independent models and find that automatically discovered performance-based classes can outperform the hand-defined query classes from previous works. The query classes that we discover indicate that some hand-defined classes from previous works are useful for query-class-dependent modeling, while others should be split into subclasses or replaced with different classes.

The unique contribution of this work is a system which can automatically discover classes of queries having consistent performance in various search methods, and therefore, similar optimal strategies for fusing those search methods. The discovered classes can also be constrained to have consistent semantic characteristics, allowing us to map incoming queries, with no performance information, into appropriate performance-based classes. The process can be used to develop a query-class-dependent model for any new domain without requiring expert knowledge, while also alleviating errors induced by human factors in query class definition.

While we have shown that automatic discovery of query classes improves over hand-defined query classes, we have also seen that much improvement can be gained through choosing better component search methods and employing better search fusion strategies. In future work, we will explore ways to discover the best search methods and fusion strategies, through similar performance-based metrics.

6. ACKNOWLEDGMENTS

This material is based upon work funded in whole [or part] by the U.S. Government. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the U.S. Government.

7. REFERENCES

- [1] LingPipe Named Entity Tagger. Available at: <http://www.alias-i.com/lingpipe/>. 2004.
- [2] J. Adcock, A. Girgensohn, M. Cooper, T. Liu, L. Wilcox, and E. Rieffel. FXPAL experiments for TRECVID 2004. In *TRECVID 2004 Workshop*, 2004.
- [3] T.-S. Chua, S.-Y. Neo, K.-Y. Li, G. Wang, R. Shi, M. Zhao, and H. Xu. TRECVID 2004 search and feature extraction task by NUS PRIS. In *TRECVID 2004 Workshop*, 2004.
- [4] T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.
- [5] J. L. Gauvain, L. Lamel, and G. Adda. The LIMSI broadcast news transcription system. *Speech Communication*, 37(1-2):89–102, 2002.
- [6] R. Kondor and T. Jebara. A kernel between sets of vectors. In *International Conference on Machine Learning*, 2003.
- [7] C. Leslie and R. Kuang. Fast kernels for inexact string matching. In *Conference on Learning Theory and Kernel Workshop*, 2003.
- [8] H. Liu. MontyLingua: An end-to-end natural language processor with common sense. Available at: <http://web.media.mit.edu/~hugo/montylingua>. 2004.
- [9] G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. J. Miller. Introduction to WordNet: An on-line lexical database. *International Journal of Lexicography*, 3(4):235–244, 1990.
- [10] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, July 1980.
- [11] P. Resnik. Using information content to evaluate semantic similarity. In *Conference on Artificial Intelligence*, 1995.
- [12] S. E. Robertson, S. Walker, M. Hancock-Beaulieu, A. Gull, and M. Lau. Okapi at TREC4. In *Text REtrieval Conference*, 1992.
- [13] A. Schwaighofer. SVM Toolbox, available at <http://www.igi.tugraz.at/aschwaig/software.html>. 2002.
- [14] V. Vapnik. *Statistical Learning Theory*. Wiley, New York, 1998.
- [15] R. Yan, J. Yang, and A. G. Hauptmann. Learning query-class dependent weights in automatic video retrieval. In *ACM Multimedia*, 2004.