# MSRA at NTCIR-10 1CLICK-2

Kazuya Narita*
Tohoku University, Sendai, Japan
narita@ecei.tohoku.ac.jp

Tetsuya Sakai
Microsoft Research Asia, Beijing, P.R.C
tetsuyasakai@acm.org

Zhicheng Dou
Microsoft Research Asia, Beijing, P.R.C
zhichdou@microsoft.com

Young-In Song
NHN Corporation, Korea
youngin.song@nhn.com

## ABSTRACT

We describe Microsoft Research Asia's approaches to the NTCIR-10 1CLICK-2 task. We construct the system based on some heuristic rules, and change the setting of our approaches to test the effectiveness of each setting. The evaluation results show the effectiveness of the query attributes.

## Team Name

MSRA

## Subtasks

Main task (Japanese)
Query Classification Subtask (Japanese)

## Keywords

Information extraction, attributes, query classification

## 1. INTRODUCTION

In this paper, we describe Microsoft Research Asia's approaches to the NTCIR-10 1CLICK-2 task. 1CLICK aims to gather and return relevant information directly to an user, to satisfy the user immediately after the click on the SEARCH button. In this task, the system is expected to present important information first and to minimize the amount of text which the user has to read [3, 7].

The NTCIR-10 1CLICK-2 task comprises the Main tasks (given a query, return a single textual output called $X$-string) and the Query Classification subtask (given a query, return the query type). In Main tasks, we can select language (English or Japanese), the length limitation of $X$-string (DESKTOP or MOBILE) and the source of $X$-string (Mandatory, Oracle or Open). We tackle the Query Classification subtask and the Main task of Japanese, DESKTOP and Mandatory, and construct the system based on some heuristic rules to output $X$-string. As a result, we submitted 1 run for the Query Classification subtask, and 4 runs (we change the setting of our approaches to test the effectiveness of each setting) for the Main task.

The reminder of this paper is organized as follows. Section 2 indicates the details of our system. Section 3 describes the evaluation results and section 4 discusses the error analysis. Section 5 concludes this paper.

*This work was done when the first author was an intern at Microsoft Research Asia.

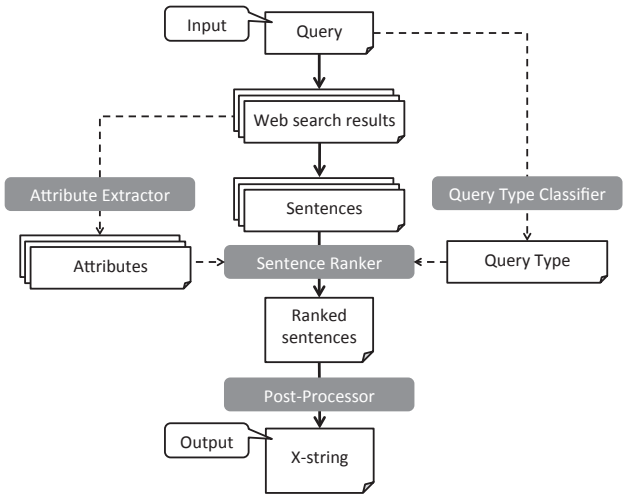


**Figure 1: System overview**

## 2. SYSTEM ARCHITECTURE

We construct the rule-based system (without machine learning techniques) to output $X$-string. Figure 1 shows an overview of our system. Our system consists of 4 modules: Query Type Classifier, Attribute Extractor, Sentence Ranker, Post-Processor.

Query Type Classifier first predicts the query type (ARTIST, ACTOR, POLITICIAN, ATHLETE, FACILITY, GEO, DEFINITION and QA) for a given query because information an user wants differ depending on the query type. Attribute Extractor then finds the query attributes from web search results to rank sentences. Next, Sentence Ranker ranks sentences from web search results according to relevance estimated by some heuristic scores. Finally, Post-Processor diversifies ranked sentences by deleting redundant information and outputs the head of diversified sentences as $X$-string.

In the following subsections, we describe in detail 4 modules of the system.

### 2.1 Query Type Classifier

This module predicts the query type (ARTIST, ACTOR, POLITICIAN, ATHLETE, FACILITY, GEO, DEFINITION and QA) for a given query.

Classification is based on some heuristic rules below:

1. If a query is longer than N, it will be classified into QA
2. If a query contains question particles, it will be classi-

fied into QA

3. If a query contains more than two functional words, it will be classified into QA

4. If

   (a) a query contains GEO-constraint clue words (e.g., “市”,“駅”, “区”, “町” ···) at its middle (not the end position) without non GEO constraint clue words (e.g., 市立) or contains blank (space) unit,

   (b) and it has GEO-specification clue words (e.g., “ビジネスホテル”, “歯医者” , “イタリアン”, “ランチ”, “外科”, “マッサージ”, “レンタカー”, · · · ) at its end,

   then it will be classified GEO

5. If

   (a) a query does not consist of only Katakana and

   (b) not contain a space and

   (c) is ended with FACILITY clue words (or contains for some special clue word, e.g., “園”,“館”, “学校”, “学”, “城”, “院”, “店”, “館”, “所”, “庁”, · · · ),

   it will be classified into FACILITY

6. If (a part of) a query is analyzed as an “ORGANIZATION” or “LOCATION” name by a NE tagger and its length is less than L, it is classified into FACILITY (in the case that it does not contain a space) or GEO (in the case that it contains a space)

7. If a wiki page(s) is/are retrieved by a query and the page contains clue words for artist/actor/politician/athlete (e.g., “小説家”, “漫画家”, “作曲家”, “選手”, “政治家”, “女優”, “俳優” ... ), the query will be classified into ARTIST/ACTOR/POLITICIAN/ATHLETE. The labels can be temporarily assigned together (due to ambiguity)

8. If a query can be found in the celebrity dictionary, it will be classified into the label in the dictionary

9. If a query is analyzed as a PERSON name (by NE tagger), it will be temporarily classified into CELEBRITY

10. If a query is ended with a Hiragana, it will be classified into QA

11. Else, it will be classified into DEFINITION (in the case that a query is short) or QA (in the case that a query is long)

Rules 7 and 9 temporarily classify the query into tentative labels due to ambiguity. We should determine tentative labels into one of eight types. We form a hypothesis that queries of the same type are more likely to co-occur than the queries of the other type (for example, an artist is more likely to co-occur with another artist than with a politician) and determine the label based on the number of hits on Bing search API[1].

In particular, we throw the combined query “<query> <type-known query>” to Bing search API. <query> is the indecisive query assigned tentative labels by rules. As <type-known query>, we made some queries on each query type. We obtain the hits of some combined queries, and assign the type of <type-known query> in the combined query which the hit is the biggest in all combined queries.

[1] http://datamarket.azure.com/dataset/bing/search

## 2.2 Attribute Extractor

This module finds the query attributes from web search results to rank sentences. Most Web queries contain one or more entities. Given a query $q$, and its top search results $R$, the query attributes extractor targets to extract attributes and their values for the entities contained in the query. These attributes and values are useful when users are seeking for basic information of entities. For example, if a user issues a query “tom cruise,” he/she is most likely looking for the famous actor Tom Cruise’s biography (e.g., birth date or performed movies). These are exactly “attributes” of entity “tom cruise.” This perfectly matches the motivation of 1-CLICK task, especially when the query is exactly comprised of an entity.

Suppose $L$ is a list of attributes we want to generated for query $q$. Each element $p \in L$ is a tuple $p =< k, V >$ where $k$ is the name of an attribute, and $V = \{W\}$ is a set of possible values of $k$ where $w =< v, w >$ is an value $v$ and its estimated importance.

Let $P = <k, v>$ is a list of key value pair. For each webpage $d$ in $R$, we extract key-value pairs using the following methods:

1. Extract all tables in $d$, and for each table $T$ which has at least two columns:

   (a) For each row $R \in T$, let $k$ be the first cell of $R$, $v$ be the second cell of $R$, and then add $<k,v>$ to $P$. This is used to extract tables like that shown in Table 1.

   (b) If the first row of $T$ is detected to be “header” (for example, it is TH tag), then for each row $\in T$ except the first row:

      i. For each cell $c$ in $R$, let $v$ be the corresponding cell in the first row (with the same column index), add $<c, v>$ to $P$.

   This is used to extract tables like that shown in Table 2.

2. Detect all blocks in $d$ using the method introduced in [2], and for each block $b$ whose text $t$ contains the character “:”:

   (a) Get all paragraphs $PS$

   (b) If the first sentence $s$ is comprised of $k$:$v$, then add $<k, v>$ to $P$.

We intent to extract key value pairs from the following example text using this algorithm:

**Born:** (1962-07-03) July 3, 1962 (age 50)

**Occupation:** Actor, producer, writer, director

**Height:** 5 ft 7 in (1.70 m)

**Religion:** Scientology

| | |
|---|---|
| **Born** | (1962-07-03) July 3, 1962 (age 50) |
| **Occupation** | Actor, producer, writer, director |
| **Height** | 5 ft 7 in (1.70 m) |
| **Religion** | Scientology |

**Table 1: An example table: the entire table is about an entity**

| Year | Title | Role |
|---|---|---|
| 1981 | Endless Love | Billy |
| 1981 | Taps | Cadet Captain David Shawn |
| 1983 | Outsiders, The Outsiders | Steve Randle |
| 1983 | Losin' It | Woody |

**Table 2: Another example table: each row includes an entity**

After we process all webpages in $R$, we already save all candidate key-value pairs into $P$. Then for each $p = <k, v>$ in $P$

1. if $L$ does contain the attribute $k$, create an empty $V$, and add <k, V> to $P$
2. Add $W =< v, 1 >$ to $P[k]$ if $P[k]$ does contains $v$; otherwise, update $P[k][v]$ to $P[k][v] + score$. $score$ is the weight of $p$, which is simply set to 1 (each occurrence of $p$ is equally important). Better weighting functions can be also used, such as, considering the rank of document $d$, the position of $p$ in document $d$, etc.

Now we already get a unique list of attributes, and their possible values with weights. We then apply some rules to filter out some noisy items, like the attribute whose length is larger than a specified value, or is found in a list of stop words.

## 2.3 Sentence Ranker

This module ranks sentences from web search results according to relevance estimated by some heuristic scores. We determined heuristic scores through repeating the test with 80 development queries, 10 queries per query type. As a result, we adopt 4 heuristic scores (scoring by content similarity, attributes, clue words/URLs and search rank) and obtain relevance by multiplying them.

In the following sections, we show heuristic scores for ranking sentences.

### 2.3.1 Content Similarity

Contents written in many web pages are likely to be more relevant. We thus apply the LexRank algorithm [5] to calculate the similarity between all sentences. The LexRank algorithm assigns a high score to a sentence that is similar to many other sentences. The algorithm detects a vector $\mathbf{p} = (p_1, p_2, ..., p_N)^T$, which represents the importance of each sentence ($N$ is the number of sentences, and the importance of a sentence $s_i$ corresponds to $p_i$), by the following recursive calculation (like the PageRank algorithm [4]):

$$\mathbf{p} = [d\mathbf{U} + (1-d)\mathbf{B}]^T\mathbf{p} \quad (1)$$

where $\mathbf{U}$ is a square matrix with all elements being equal to $1/N$, $\mathbf{B}$ is the similarity matrix (element $b_{ij}$ is the cosine similarity between sentences $s_i$ and $s_j$), and $d$ is a "damping factor". After applying the LexRank algorithm, we obtain $p_i$ as the score of sentence $s_i$ by the content similarity.

### 2.3.2 Attributes

This score is based on attributes extracted automatically in Attribute Extractor. The more attributes the sentence has, the higher the score of the sentence is. So we adapt attributes as positive clue words. To test the effectiveness of attributes, we prepare two scoring methods for attributes (binary weights and tf-idf weights) and a scoring method without attributes.

In Attribute Extractor, we obtain each attribute list for each query. Each attribute list is noisy, so we use not only the attribute list of the input query but also attribute lists of other queries to filter out. In detail, we use the type of each query given by Query Type Classifier. Some attributes are found in some attribute lists of the same query type, because it is expected that attributes of the same query type are similar. We thus only adopt the attributes found in some attribute lists of the same query type, to filter out some noisy attributes.

We assign higher score as the score of attributes when a sentence has more attributes. However, each attribute is supposed to have each level of importance. We thus prepare the weight for each attribute as importance. The weight of the attribute $a$ is decided by the attribute lists of the other query type like tf-idf:

$$w(a) = \frac{n}{N}\left(\log\frac{T}{T_a} + 1\right) \quad (2)$$

where $n$ is the number of attribute $a$ in attribute lists of the same query type, $N$ is the number of token in attribute lists of the same query type, $T$ is the number of query types ($T = 8$ in the 1CLICK-2 task), and $T_a$ is the number of query types which the attribute lists have attribute $a$ ($1 \leq T_a \leq T$). By this weight, we can give high priority to characteristic attributes. For comparison, we also test binary weights which is not considered each level of importance in the experiment.

### 2.3.3 Clue words and URLs

In addition to attributes extracted automatically, we manually prepare some clue words and URLs. For example of clue words, for a celebrity query (ARTIST, ACTOR, POLITICIAN and ATHLETE), we can expect that a sentence that contains the words "birthday" or "birth town" is probably relevant, while a sentence that contain the words "posted by" or "answerer" is probably not relevant. In the case of clue URLs, for a celebrity query, sentences from "talent.yahoo.co.jp" or "ja.wikipedia.org" are more important, while sentences from "amazon.co.jp" are less important for an user. We assign higher score when a sentence has positive clue words, while we assign lower score when a sentence has negative clue words.

### 2.3.4 Search Rank

Sentences in high rank pages in web search results are more important due to the confidence of the search engine. We thus construct the following linear score by the search rank:

$$S_{\text{srank}}(s) = 1 - \frac{\text{rank}_{\text{search}}(\text{s})}{N_{\text{search}}} \quad (3)$$

where $N_{\text{search}}$ is the number of pages in web search results, and $\text{rank}_{\text{search}}(s)$ is the search rank of the page including the sentence $s$ ($0 \leq \text{rank}_{\text{search}}(s) < N_{\text{search}}$). This score is high when the search rank of the page is high.

## 2.4 Post-Processor

This module diversifies ranked sentences and outputs the head of diversified sentences as $X$-string.

In Sentence Ranker, Sentences with high scores are similar to each other because we use content similarity for scoring. When we output sentences with high scores as $X$-string, we should diversify sentences in the output. We thus summarize the sentences by using the MMR algorithm [1]. The MMR algorithm selects sentences for the output by using the similarity as the penalty. An output sentence is decided by the following formula:

$$\text{MMR} = \arg\max_{s_i \in R \setminus S} \left[ \lambda \text{Score}(s_i) - (1-\lambda) \max_{s_j \in S} \text{Sim}(s_i, s_j) \right] \quad (4)$$

where $R$ is the ranked list of sentences, $S$ is the subset of sentences in $R$ that are selected as the output, $R \setminus S$ is the subset of as yet unselected sentences in $R$, $\text{Score}(s_i)$ is a score of a sentence $s_i$ by Sentence Ranker, $\text{Sim}(s_i, s_j)$ is the cosine similarity between sentence $s_i$ and $s_j$, and $\lambda$ is a controlling parameter. This formula first selects a sentence for the output and the selected sentence is added to $S$. Next, the formula is calculated again, and adds the next sentence to $S$. The algorithm repeats this processes until $S$ has enough sentences for the output.

The MMR algorithm is based on the similarity between a selected sentence and a yet unselected sentence. We don't consider only the similarity between a sentence and another sentence, but also the similarity between a set of selected sentences and a set of yet unselected sentences. So, when a selected sentence is added to the set of selected sentences in the MMR algorithm, we do not add the sentence, which is only composed of the content words in the set of selected sentences.

Finally, this module deletes unimportant strings occurring frequently (like "出典" (reference)), and outputs the head of ranked sentence list as $X$-string.

# 3. EVALUATION RESULTS

In this section, we report on our results in the NTCIR-10 1CLICK-2 task. For Main tasks, We submitted 4 runs to test the effectiveness of each settings, and for Query Classification subtask, we submitted 1 run.

## 3.1 Main tasks

We submitted 4 runs to test the effectiveness of each settings. Table 3 shows the settings of each run. The differences of each run are "Query type" and "Attribute". "Query type" is whether the scoring depends on the query type, i.e. whether scoring with/without manual clue words and URLs depending on the query type. "Attribute" is the scoring method for attributes (attributes with tf-idf weights, attributes with binary weights, or without attributes).

Table 4 shows the mean $S\sharp$-measure [3, 6] for each run. For comparison, the results of other mandatory runs is also shown. The MSRA runs are not statistically significant from each other according to the organizers [3]. Our runs and the other teams' runs are also statistically indistinguishable. The exceptions are

- ORG-J-D-MAND-1 with all our runs ($p<0.01$);
- ORG-J-D-MAND-2 with all our runs ($p<0.01$);
- MSRA-J-D-MAND-1 with TTOKU-J-M-MAND-3 ($p<0.05$)

Figure 2 shows the per-query $S\sharp$-measures (in case that the match type is "union") for each run[2].

| Run name | Query type | Attribute |
|---|---|---|
| MSRA-J-D-MAND-1 | dependent | tf-idf weights |
| MSRA-J-D-MAND-2 | dependent | binary weights |
| MSRA-J-D-MAND-3 | dependent | no |
| MSRA-J-D-MAND-4 | independent | no |

**Table 3: Settings of each run**

| Run name | Union | Intersection |
|---|---|---|
| MSRA-J-D-MAND-1 | **0.1552** | **0.1082** |
| MSRA-J-D-MAND-2 | 0.1191 | 0.0754 |
| MSRA-J-D-MAND-3 | 0.1230 | 0.0816 |
| MSRA-J-D-MAND-4 | 0.1127 | 0.0608 |
| ORG-J-D-MAND-1 | **0.2529** | 0.1971 |
| ORG-J-D-MAND-2 | 0.2398 | **0.2031** |
| KUIDL-J-D-MAND-1 | 0.1539 | 0.0915 |
| KUIDL-J-M-MAND-3 | 0.1519 | 0.0960 |
| TTOKU-J-M-MAND-3 | 0.0704 | 0.0395 |

**Table 4: Overall $S\sharp$-measure for mandatory runs**

## 3.2 Query Classification Subtask

The confusion matrix for Query Classification subtask is shown in Table 5. Overall, our classifier shows 83% accuracy $(83/100)$[3].

# 4. DISCUSSION

In this section, we describe the error analysis of Query Type Classification and Sentence Ranking. We discuss the effectiveness of the query attribute especially.

## 4.1 Query Type Classification

We show the list of misclassifications in Table 6. Our classifier performs well except FACILITY queries. For example, we find that clue words in a wiki page (rule 7) are very efficient for celebrity queries.

For FACILITY queries, the classifier predicts FACILITY by mainly suffix (rule 5c). 3 of 8 errors, which FACILITY queries are not predicted as FACILITY, are due to insufficient suffix. However, the remain errors are difficult because they are predicted by suffix. We find that the approach only by simple heuristic rules such as suffix has limitations. In order to correctly classify these queries, we need other knowledge. For instance, a wiki page is efficient for celebrity

[2]Although MSRA-J-D-MAND-3 and MSRA-J-D-MAND-4 are actually identical for DEFINITION queries, they received different S♯ scores as different assessors assessed these runs respectively. The differences are not statistically significant.

[3]This accuracy ranks seventh among eleven submitted runs.

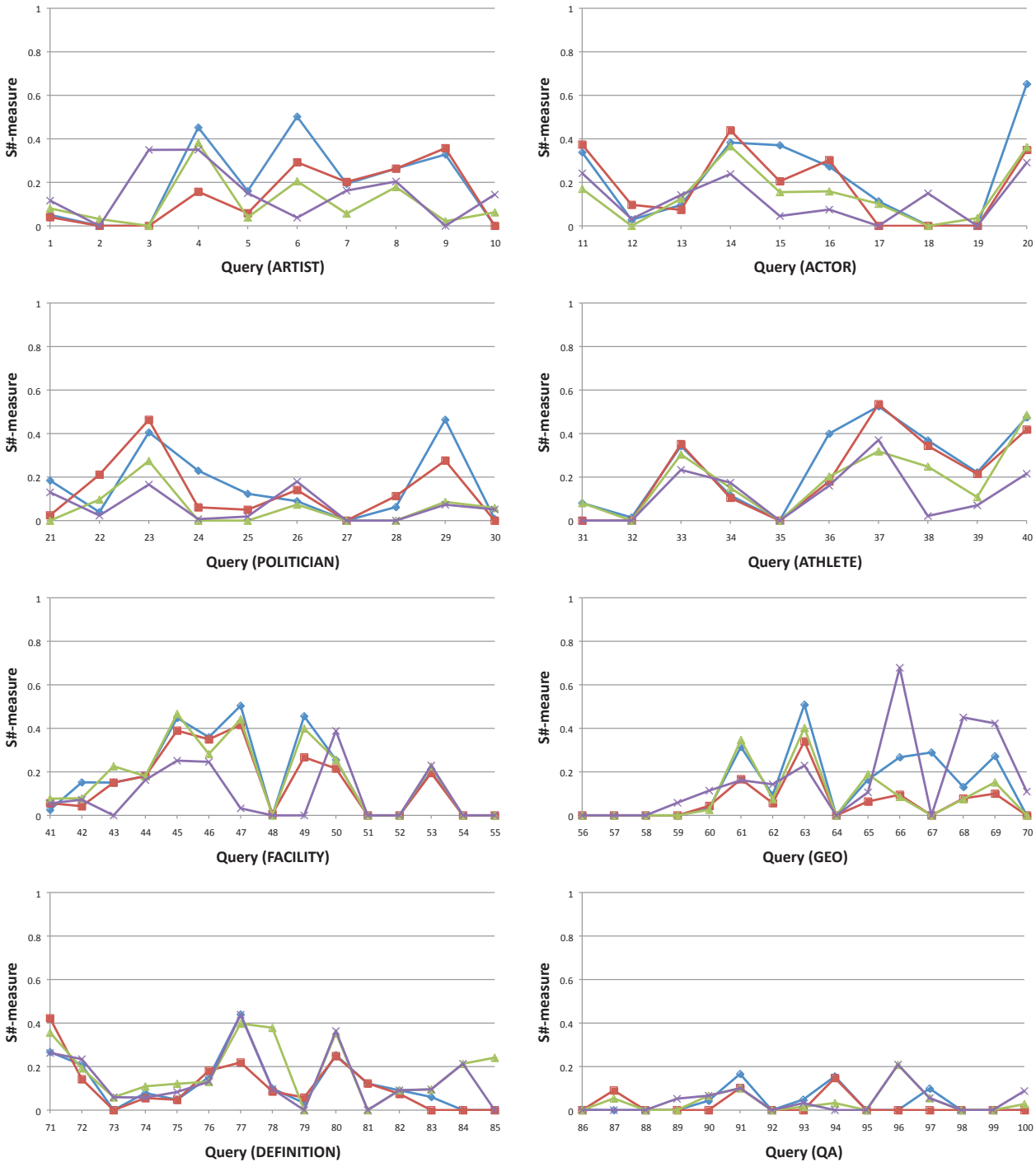


**Figure 2: Per-query $S\sharp$-measures (union): Blue lines are MSRA-J-MAND-1; Red lines are MSRA-J-MAND-2; Green lines are MSRA-J-MAND-3; Purple lines are MSRA-J-MAND-4. The $x$-axis shows the query IDs.**

| gold \ system | ARTIST | ACTOR | POLITICIAN | ATHLETE | FACILITY | GEO | DEFINITION | QA | Recall |
|---|---|---|---|---|---|---|---|---|---|
| ARTIST | **9** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0.90 |
| ACTOR | 1 | **9** | 0 | 0 | 0 | 0 | 0 | 0 | 0.90 |
| POLITICIAN | 1 | 0 | **8** | 0 | 0 | 0 | 1 | 0 | 0.80 |
| ATHLETE | 0 | 0 | 0 | **10** | 0 | 0 | 0 | 0 | 1.00 |
| FACILITY | 1 | 0 | 0 | 0 | **7** | 1 | 5 | 1 | 0.47 |
| GEO | 1 | 0 | 0 | 0 | 0 | **14** | 0 | 0 | 0.93 |
| DEFINITION | 0 | 0 | 0 | 1 | 1 | 1 | **12** | 0 | 0.80 |
| QA | 0 | 0 | 0 | 0 | 0 | 0 | 1 | **14** | 0.93 |
| Precision | 0.69 | 0.90 | 1.00 | 0.91 | 0.88 | 0.88 | 0.63 | 0.93 | |
| $F_1$ score | 0.78 | 0.90 | 0.89 | 0.95 | 0.61 | 0.90 | 0.70 | 0.93 | |

**Table 5: Confusion matrix for Query Classification subtask**

| Query ID | Query string | Gold type | Predicted type | Applied rules |
|---|---|---|---|---|
| 1C2-J-0006 | 三谷幸喜 | ARTIST | ACTOR | 7 |
| 1C2-J-0015 | シルヴェスター スタローン | ACTOR | ARTIST | 7 & Bing Search API |
| 1C2-J-0021 | ロバート ケネディ キューバ | POLITICIAN | DEFINITION | 11 |
| 1C2-J-0029 | 野田聖子 | POLITICIAN | ARTIST | 9 & Bing Search API |
| 1C2-J-0042 | 京都真如堂 | FACILITY | DEFINITION | 11 |
| 1C2-J-0043 | ホテルアンビア松風閣 | FACILITY | QA | 11 |
| 1C2-J-0045 | 南ヶ丘牧場 | FACILITY | DEFINITION | 11 |
| 1C2-J-0048 | カーサ・ディ・ナポリ | FACILITY | DEFINITION | 11 |
| 1C2-J-0049 | ザ・ペニンシュラ東京 | FACILITY | ARTIST | 9 & Bing Search API |
| 1C2-J-0052 | アメリカン エアラインズ アリーナ | FACILITY | DEFINITION | 11 |
| 1C2-J-0053 | アトランタ 空港 | FACILITY | GEO | 6 |
| 1C2-J-0055 | らーめんてつや | FACILITY | DEFINITION | 11 |
| 1C2-J-0070 | 恵比寿 結婚式場 | GEO | ARTIST | 9 & Bing Search API |
| 1C2-J-0074 | 秘密の花園 | DEFINITION | FACILITY | 5 |
| 1C2-J-0075 | トルネード | DEFINITION | ATHLETE | 7 |
| 1C2-J-0081 | 感謝祭 カナダ | DEFINITION | GEO | 6 |
| 1C2-J-0087 | もち米の炊き方 | QA | DEFINITION | 11 |

**Table 6: Query Type Classification Errors**

queries in rule 7. So it is assumed that a wiki page is also efficient for FACILITY queries.

## 4.2 Sentence Ranking

We showed overall S♯-measure in Table 4 in section 3.1. MSRA-J-D-MAND-1 (query type dependent; attributes with tf-idf weights) shows the best performance of our runs but the performance of MSRA-J-D-MAND-2 (query type dependent; attributes with binary weights) is lower than MSRA-J-D-MAND-3 (query type dependent; no attributes). So we find that the query attribute is efficient but we should consider how to weight the query attributes.

We analyze the effectiveness of the query attributes in detail by comparing the results of MSRA-J-D-MAND-1 and MSRA-J-MAND-3. Table 7 shows mean S♯-measure (union) of our runs for each query type. MSRA-J-D-MAND-1 shows higher performance than MSRA-J-D-MAND-3 except DEFINITION and QA. Especially, there are differences between two runs for celebrity queries. We therefore find that the query attributes are efficient for celebrity queries. For example, the query attributes such as “所属” (affiliation) and “打率” (batting average) are efficient for the ATHLETE query “イチロー” (Ichiro; professional baseball player). On the other hand, for DEFINITION and QA queries, it is hard to consider the query attributes by the nature of the queries. Thus, the query attributes are inefficient for DEFINITION and QA queries.

By the way, it is interesting that MSRA-J-D-MAND-4 (query type independent) shows the best performance of our runs for GEO and QA queries. There is little practical difference among 4 runs for QA queries. However, for GEO queries, MSRA-J-D-MAND-4 outperforms our other runs. This means that a similarity-based approach is better than an approach based on clue words (including query attributes). It is assumed that a pattern-based approach may be better but that approach will be addressed in the future.

# 5. CONCLUSIONS

In this paper, we presented Microsoft Research Asia’s approaches to the NTCIR-10 1CLICK-2 task. We constructed the system based on some heuristic rules and showed the effectiveness of our query attributes.

In the future, we will improve the accuracy of query type classification by knowledge on web and consider new framework instead of query attributes for DEFINITION and QA queries.

# 6. REFERENCES

[1] J. Carbonell and J. Goldstein. The use of MMR, Diversity-Based Reranking for Reordering Documents and Producing Summaries. In *Proceedings of the 21st*

| Run name | ARTIST | ACTOR | POLITICIAN | ATHLETE | FACILITY | GEO | DEFINITION | QA |
|---|---|---|---|---|---|---|---|---|
| MSRA-J-D-MAND-1 | **0.1950** | **0.2256** | **0.1606** | **0.2543** | **0.1820** | 0.1381 | 0.1234 | 0.0343 |
| MSRA-J-D-MAND-2 | 0.1367 | 0.1840 | 0.1340 | 0.2148 | 0.1512 | 0.0629 | 0.1105 | 0.0227 |
| MSRA-J-D-MAND-3 | 0.1492 | 0.1475 | 0.0589 | 0.1899 | 0.1754 | 0.0900 | **0.1517** | 0.0373 |
| MSRA-J-D-MAND-4 | 0.1514 | 0.1215 | 0.0649 | 0.1246 | 0.0958 | **0.1649** | 0.1418 | **0.0403** |

**Table 7: Mean per-query $S\sharp$-measure. Bold indicates the highest performance in each query type.**

*annual international ACM SIGIR conference on Research and development in information retrieval*, pages 335–336, 1998.

[2] Z. Dou, S. Hu, Y. Luo, R. Song, and J.-R. Wen. Finding dimensions for queries. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1311–1320, 2011.

[3] M. P. Kato, M. Ekstrand-Abueg, V. Pavlu, T. Sakai, T. Yamamoto, and M. Iwata. Overview of the ntcir-10 1click-2 task. In *Proceedings of NTCIR-10*, 2013.

[4] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford University, 1998.

[5] D. R. Radev. Lexrank: Graph-based lexical centrality as salience in text summarization. *Journal of Artificial Intelligence Research*, 22:457–479, 2004.

[6] T. Sakai and M. P. Kato. One click one revisited: Enhancing evaluation based on information units. In *Proceedings of the 8th Asia Information Retrieval Societies Conference*, pages 39–51, 2012.

[7] T. Sakai, M. P. Kato, and Y.-I. Song. Click the search button and be happy: evaluating direct and immediate information access. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 621–630, 2011.