Deep Classifier: Automatically Categorizing Search Results into Large-Scale Hierarchies

Presented by Qiang Yang
Hong Kong Univ of Sci and Tech.

Dikan Xing, Guirong Xue, Yong Yu
Shanghai Jiao-tong University
Qiang Yang
Hong Kong University of Science and Technology
Search Result Presentation

- List form or hierarchical form
- Hierarchical form preferred by many users
  - [Chen and Dumais 2000]
  - [Hearst 2006]
  - [Etzioni et al. : Grouper; WWW ’99]

**Question**
- How do we automatically categorize search results from a list form into a hierarchical form?
  - Based on classification rather than clustering
  - Deep vs. Shallow
Users prefer categorization but too shallow so far

Data Mining can help, but needs to be efficient and effective
All Results(apple)(100)
  - Arts
    - Music
      - Bands and Artists
        - Apple, Fiona(5)
    - A

Computers
  - Software
    - Operating Systems
      - Mac OS(3)
  - Systems
    - Apple(3)
  - Games
    - Card Games
      - Special Decks
        - Apples to Apples(37)
  - Home
    - Cooking
      - Beverages
        - Apple Juice and Cider(1)
      - Fruits and Vegetables
      - Apples
        - Apple Dumplings(1)

Shopping
  - Food
    - Confectionery
      - Candy Apples(46)

26. FIONA APPLE
Official website provided by Sony Entertainment. latest videos and album releases.
http://www.fiona-apple.com

52. Apple İMC Türkiye
Apple e-Bülten’e abone olmak için tıklayınız... Ap sites around the world:. Seçin... Africa, Asia, http://www.apple.com.tr

54. Apple Magyarországi Képviselet
Apple IMC Hungary. Visit other Apple sites aroun Asia, Australia, Austria, Azerbaijan, Belgium, B http://www.apple.hu

56. Applegeeks 3.0
Fission is a streamlined audio editor that allow Apple Lossless and AIFF with no re-encoding, so:
http://www.applegeeks.com

67. UVM Apple Orchard
The UVM Apple Program: Extension and Research fo in Vermont and beyond...
http://orchard.uvm.edu
Query="Saturn"
Deep Classifier: Detailed Steps

1. Identify a large online taxonomy $T$ for categorization
   1. Open Directory Project, Yahoo directories, etc.
2. Given a query $Q$, obtain a set of candidate categories $C(Q)$
3. Prune $T(Q)$ using $C(Q)$
   1. The result is a deep and narrow taxonomy $T(Q)$, where all leaf nodes are candidates
4. Build a classifier into the leaf nodes in $T(Q)$
5. Classify each search result in $S(Q)$ into $T(Q)$
6. Present $T(Q)$ to the user

Properties:
- The search results $S(Q)$ are classified into different categories $C(Q)$ for different queries $Q$
- A classifier is trained online for each incoming query
  - Is this feasible?
- The classifier should be both trained efficiently and accurate
Research Question 1: How to build a classifier online?

- Given a query, we can use the search functions of various online taxonomies to find the candidate categories
  - ODP already does this
- To build a classifier into these candidates, we must **collect training data** for each category
  - How?
Question: How to build a classifier in real time?

- Flat Strategy
- Hierarchical Strategy
- Ancestor-Assistant Strategy
Flat Strategy for Training a Classifier

- **Pros:** Simple
- **Cons:**
  - Multi-class!
  - Data Scarcity
    - 21.6 docs per class node
    - Easy to overfit!
Hierarchical Strategy [related works…]

- Classify top-down, level-by-level

\[
P(c_j|x)
= P(c_j^1, c_j^2, \ldots, c_j^{l_i}|x)
= \prod_{k=1}^{l_i} P(c_j^k|x, c_j^1, c_j^2, \ldots, c_j^{k-1})
\]

- Problem:
  - Slow
  - Few docs under each node
  - Top level docs too general
Ancestor-assistant Strategy

To build a classifier for results of query Q, let Ci be a category of T(Q)
- T(Q) is the pruned taxonomy tree of Q

For each candidate Ci,
- Collect training documents
  - from Ci,
  - Father of Ci, Cousins of Ci
  - Grandfather of Ci, Uncles of Ci
  - ... 
  - Until an ancestor is reached, which includes a competitor candidate Cj as descendent

Let the union of these documents be D
Label D by category Ci
Build a classifier for Ci using D and using the flat strategy
Ancestor-Assistant Strategy

- Find the highest ancestor that does not include another candidate node as descendent
- Borrow data from ancestors, and their descendents
  - Now 661.2 (vs. 21.6) per class
Deep Classifier: choice of classifier

Available strategies include:

- Flat Strategy: Naïve Bayes Classifier - fast
- Hierarchical Strategy: Standard NBC Hierarchical Strategy
- Ancestor-Assistant Strategy: Discriminative NBC
Next Research Question: How to choose a classifier

- A classifier is trained for each query
- Thus, efficiency is a concern!
  - Using SVM or other time-consuming classifiers would not be feasible
  - Using Naïve Bayesian Classifiers (NBC) is a good choice
    \[
    \Pr(C_i \mid Doc) \propto \Pr(C_i) \times \prod_{j=1}^{N} \Pr(word_j \mid C_i)
    \]
    - We can calculate the conditional probability table beforehand
    - Thus only need to multiply some factors in real time
Online Classification: choices

\[
\Pr(C_i \mid Doc) \propto \Pr(C_i) \prod_{j=1}^{N} \Pr(\text{word}_j \mid C_i)
\]

- Two Problems with NBC:
  - Probability of each category in ODP is fixed
  - Probability of each category in search result varies w/ Q
    - \( \Pr(C_i) \) not the same between training (ODP) and test data (top-100 search results)
      - Thus basic machine learning assumption violated, and we may need transfer learning, or…
    - Count(terms)/Count(categories) may be too small
      - when Count(categories) too large (>100),
      - The contribution of each term is tiny, thus not discriminative enough!

- We wish to make the contribution of each term much larger than in traditional NBC
Making NBC Fast and Accurate

Two Assumptions:
- Let $\Pr(C_i) = \frac{1}{n}$, where $n$ is the number of classes, for all $C_i$
- $\Pr(C_i|\text{Doc})$ proportional to $\Pr(C_i|\text{word } j)$, which is proportional to the number of categories per word
  - This is much more discriminative than $\Pr(\text{word } j|C_i)$

$$
\Pr(C_i \mid \text{Doc}) \propto \prod_{j=1}^{N} \Pr(C_i \mid \text{word}_j)
$$
When testing a search result, only words occur in the snippets are considered.

The time complexity for testing is $O(n \times \log N + n \times m + K)$,

- $n$ is the length of the snippets,
- $m$ is the number of category candidates
- $N$ is the size of the whole term vocabulary

The first item denotes the time to convert snippets into word ID,

the second item denotes the time to classify,

$K$ is the time for memory swapping

However, the computational efficiency part needs to be explored much further in our future works

Instead, in our experiments, we focused on accuracy only
Experiments

- We first collected 1000 popular queries from a search engine, and computed the distribution of their results among the top-level categories in ODP.
- ~94.7% of the queries are distributed over less than six categories,
  - of which about 74.2% of queries are over three or less categories.
  - The two most widely distributed
    - games (in 14 top-level categories) and books (in 12 top-level categories).
- This indicates that
  - top-level categories may be too coarse for many queries
  - deep categories are necessary
Experimental Hypotheses

- The Ancestor-assistant strategy may outperform the hierarchical and the flat strategies.
- The discriminative naive Bayesian classifier may outperform the traditional NBC.
- The discriminative naive Bayesian classifier is much faster than SVM.
Evaluation Data

Data Sets for Evaluation

- Data Set I
  - Search results from simulated search engine
  - Randomly picking 100 from query log.

- Data Set II
  - Case study: ambiguous queries.
  - Real search results from Google.

<table>
<thead>
<tr>
<th>Pages</th>
<th>Categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,297,222</td>
<td>157,927</td>
</tr>
</tbody>
</table>
Different Training Data Selection Strategies
Different Classifiers

(Each is averaged over all queries in the data set.)
## Results on Queries as function of training data selection strategy

<table>
<thead>
<tr>
<th>Query</th>
<th>Micro-F1</th>
<th>Macro-Precision</th>
<th>Macro-Recall</th>
<th>Macro-F1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>flat</td>
<td>hie</td>
<td>aa</td>
<td>flat</td>
</tr>
<tr>
<td>ajax</td>
<td>0.99</td>
<td>0.86</td>
<td>0.99</td>
<td>0.92</td>
</tr>
<tr>
<td>apple</td>
<td>0.77</td>
<td>0.21</td>
<td>0.72</td>
<td>0.74</td>
</tr>
<tr>
<td>dell</td>
<td>0.67</td>
<td>0.62</td>
<td>0.64</td>
<td>0.20</td>
</tr>
<tr>
<td>jaguar</td>
<td>0.61</td>
<td>0.41</td>
<td>0.94</td>
<td>0.59</td>
</tr>
<tr>
<td>java</td>
<td>0.93</td>
<td>0.29</td>
<td>0.83</td>
<td>0.48</td>
</tr>
<tr>
<td>saturn</td>
<td>0.71</td>
<td>0.41</td>
<td>0.98</td>
<td>0.60</td>
</tr>
<tr>
<td>subway</td>
<td>0.91</td>
<td>0.86</td>
<td>0.94</td>
<td>0.70</td>
</tr>
<tr>
<td>trec</td>
<td>0.60</td>
<td>0.52</td>
<td>0.80</td>
<td>0.34</td>
</tr>
<tr>
<td>ups</td>
<td>0.72</td>
<td>0.81</td>
<td>0.81</td>
<td>0.32</td>
</tr>
<tr>
<td>(average)</td>
<td>0.78</td>
<td>0.55</td>
<td>0.85</td>
<td>0.56</td>
</tr>
</tbody>
</table>
Conclusions

Objective

- Applying Hierarchical Classification to Search Result Categorization

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large Hierarchies</td>
<td>Pruned for each query</td>
</tr>
<tr>
<td>Few Training Data</td>
<td>Ancestor-assistant Strategy</td>
</tr>
<tr>
<td>Efficiency for Online Application</td>
<td>Faster and more effective Discriminative Naïve Bayesian classifier</td>
</tr>
</tbody>
</table>