

Skylin Web Service and SkylinBot

By Christopher Lim, Luo Pan, Ting-You Wang, and Chris White

Overview

In order to fulfill the vision of the semantic web, it is necessary to bootstrap structured data for applications to leverage. A recent [paper](#) introduced the Kylin system for autonomously semantifying Wikipedia. This system extracted structured data (infobox information) from four classes of articles: U.S. Counties, Actors, Airlines, and Universities, with varying results. Skylin (Scalable Kylin) aims to make Kylin's information extraction and classification capabilities publicly available as a web service and it includes a bot for writing back extracted information into Wikipedia to update infobox information or assist authors in creating infoboxes for articles. The Skylin project also extended Kylin to more classes of articles, enabling it to finally classify and extract structured data from documents about automobiles, actors, airlines, company, countries, films, movies, persons, singles, universities and US counties.

Goals

- Reproduce Kylin and its original results
- Extend its capabilities to four new classes and measure performance with precision/recall statistics
- Extend recall of articles for a given category
- Make Kylin's information extraction and classification capabilities publicly available through a web service
- Create a WikiBot that will take leverage the web service to assist infobox authorship and editing and allow users to approve/disapprove autonomous changes
- Fully automate the reintegration of autonomously extracted data into Wikipedia

We also developed a web interface into Skylin for comparing Wikipedia articles and the data extracted from the article by the service. We were unable to extend Kylin's recall for a given category due to the complexities of the Kylin software and our limited time frame. We were also prevented from developing a truly autonomous WikiBot because of political issues with the Wikipedia community.

System Architecture

These were several key design decisions we made:

- Apache Tomcat Web Server to integrate nicely with Kylin's java codebase. Using servlets, user requests could easily interface with Kylin's process.
- Keeping Kylin's classification and extraction algorithms. We decided to focusing on extending Kylin to more classes of articles rather than attempting to improve its already good results with new algorithms for classifying pages (to increase recall). This would increase the scope of the service and areas where it would be applicable. Additionally, the complexities of the Kylin software and the semi-structured nature of Wikipedia meant that recall could be better improved in the short term by enhancing the quality of Wikipedia's structured data rather than developing new algorithms that would have to be grafted into the code.

- The following diagram illustrates the architecture of our complete system, including Kylin, the Skylin Web Service layer, and the SkylinBot build on top of the service.

Skylin Architecture



We essentially trained the original Kylin system on a static Wikipedia xml dump for 8 categories and serialized its extractors, sentence classifiers and other related files. These objects were made available to the service layer through an abstraction that made them easy to use and the functionality is exposed to the web. The Skylin service offers several web methods, the core of which are shown in this diagram. These methods are called by the SkylinBot, which communicates with Wikipedia through the Wikimedia API writing back infobox updates either to a user sandbox or the live Wikipedia site. The Skylin service itself makes use of Wikipedia's Special Export directive to obtain the wikisource of articles, which are the input format that Kylin uses for its processing.

Usage Scenarios

The following is an end-to-end example of using the SkylinBot with the Skylin Information Extraction Service.

- The user will run the SkylinBot and enter in his Wikipedia Credentials. The bot will make a request to <http://abstract.cs.washington.edu/~chrislim/skylin.php?path=/skylin/listCategories> to get a listing of the categories of Wikipedia documents the Skylin Web Service can extract information from. The following is an example of the output from the listCategories page:

```
- <categories>
  <category id="0">automobile</category>
  <category id="1">actor</category>
  <category id="2">airline</category>
  <category id="3">company</category>
  <category id="4">country</category>
  <category id="5">film</category>
  <category id="6">movie</category>
  <category id="7">person</category>
  <category id="8">single</category>
  <category id="9">university</category>
  <category id="10">uscounty</category>
</categories>
```

- This list of categories will be made available to the user of the SkylinBot to select which class of

articles to run an extraction on (assume the user selects the 'person' category for this example).

4. After selecting the category to work with, SkylinBot requests a list of Wikipedia articles that are classified to be within that category, based on a static analysis of the latest wikidump. This list is requested at


<http://abstract.cs.washington.edu/~chrislim/skylin.php?path=/skylin/listTitles?category=person> and sample output is as follows:

```
- <category name="person">
  <instance id="0" title="A. D. Hope"/>
  <instance id="1" title="A. J. Liebling"/>
  <instance id="2" title="Aaron Buerge"/>
  <instance id="3" title="Abba Eban"/>
  <instance id="4" title="Abbas-Ali Amid Zanjani"/>
  <instance id="5" title="Abdalqadir as-Sufi"/>
  <instance id="6" title="Abdul Aziz al-Hakim"/>
  <instance id="7" title="Abdul Majid al-Khoei"/>
  <instance id="8" title="Abigail Folger"/>
  <instance id="9" title="Abraham Abell"/>
```

- 5.
6. The SkylinBot user can then run Information Extraction on each returned title by calling the extractInfo method:

http://abstract.cs.washington.edu/~chrislim/skylin.php?path=/skylin/extractInfo?url=http://en.wikipedia.org/wiki/Albert_Einstein&category=person&format=kylin

This call will run Kylin's Person extractors on the live Wikipedia page about Albert Einstein and returns a custom XML document:

```
- <kylin>
  - <extraction>
    <attribute name="name" value="Albert Einstein" probability="0.8265500418444203"/>
    <attribute name="death_date" value="April 18 , 1955" probability="0.9499070325480579"/>
    <attribute name="spouse" value="Elsa L wenthal" probability="0.9883835291398091"/>
  </extraction>
</kylin>
```

7. The SkylinBot will parse this returned document and for all attribute name/value pairs that exceed a user-defined probability threshold (~80%), it will present the user with the option of accepting the extracted Infobox attribute or rejecting it. If an attribute already exists in the article and has a conflicting value, both the old and newly suggested value will be presented. Users also have the option of committing the suggested changes to their Wikipedia sandbox page and previewing the results before committing them to the live site. The following are some screenshots:

Please Add photos from SkylinBot Here

8. The Skylin Web Service also supplies some statistics obtained from the training of extractors and classifiers on the latest wikidump from Wikipedia. These statistics about the number of articles that had a particular Infobox or a particular attribute within that Infobox can be found at: <http://abstract.cs.washington.edu/~chrislim/skylin.php?path=/skylin/listInfoboxStats?category=person> (so a category must be specified) with the following sample output:

- ```

- <Infobox name="person" instances="52" attributes="41">
 <property name="name" instances="48"/>
 <property name="birth_name" instances="5"/>
 <property name="birth_date" instances="30"/>
 <property name="birth_place" instances="37"/>
 <property name="death_date" instances="20"/>
 <property name="death_place" instances="15"/>
 <property name="occupation" instances="32"/>
 <property name="spouse" instances="15"/>
 <property name="imagesize" instances="14"/>
 <property name="height" instances="2"/>
 <property name="image" instances="18"/>
 <property name="caption" instances="8"/>
 <property name="religion" instances="11"/>
9.
10. If for any of the Skylin methods an invalid parameter is supplied, our service will output an XML formatted exception; one example follows for an invalid category parameter:
- <kylin>
 - <exception code="0x02">
 <message>Please supply a valid category and or url</message>
 - <details>
 Given Url: http://en.wikipedia.org/wiki/Albert_Einstein Category: Person{automobile=0, university=9, country=
 company=3, uscounty=10, single=8, airline=2, film=5}
 </details>
 </exception>
11. </kylin>

```

Note: Our exception system is not fully implemented, documented or consistent yet. Most exceptions contain a useful error message, but the error codes are arbitrary and not useful yet. Some errors may also directly output stack trace and other debug information instead of packaging the error into the standard xml format first.

12. Finally, a sample web application can be found at <http://abstract.cs.washington.edu/~chrislim/skylin.php?path=/skylin/index.jsp?page=sample> or <http://attu.cs.washington.edu:9080/skylin/index.jsp?page=sample>. In this page you can provide a Wikipedia url and category and it will display side-by-side the current live Wikipedia page and the information Kylin extracted from the page next to it. The following is a screenshot:



This is a sample web application which leverages the Skylin Web service. Simply provide a valid Wikipedia article URL and the category that the page belongs to. This page will show the live page's infobox attributes as well as the ones extracted by Skylin.

http://en.wikipedia.org/wiki/Honda\_Civic      automobile      Submit

search

Go      Search

toolbox

- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link
- Cite this article

languages

- Català
- Česky
- Deutsch
- Español
- Français
- 한국어
- Bahasa Indonesia
- Italiano

references. Unsourced material may be challenged and removed. (May 2007)

The Honda Civic is a compact car

**Honda Civic**

|                     |                                                  |
|---------------------|--------------------------------------------------|
| <b>Manufacturer</b> | Honda                                            |
| <b>Production</b>   | 1972-present                                     |
| <b>Predecessor</b>  | Honda Z360<br>Honda Z600                         |
| <b>Class</b>        | Subcompact (1972-2000)<br>Compact (2001-present) |

manufacturer: Honda  
Prob: 0.9880458671969369  
name: Honda Civic  
Prob: 0.9999925388427284  
class: compact car  
Prob: 0.9997689328757813  
body\_style: three - door hatchback  
Prob: 0.9036503398370603  
transmission: manual  
Prob: 0.7036124901598457  
manufacturer: Toyota Corolla  
Prob: 0.7153456507055477  
related: Toyota Corolla  
Prob: 0.5098348233484155  
body\_style: 3 - door hatchback  
Prob: 0.9996300737446749  
engine: 1500 cc  
Prob: 0.9231154809180152  
engine: 1300 cc 8  
Prob: 0.5073015844148102  
transmission: 6 - speed manual  
Prob: 0.9997602994366311  
image: DOHC VTEC 160ps  
Prob: 0.562249553479006  
transmission: automatic

manufactured by Honda. It was introduced in July 1972

13.

Note: Because we did not have full control over our Apache Tomcat servers during development, the servers would sometimes crash and we were powerless to bring them back online. Therefore, we would start new nodes on other servers, but this constantly changing server scheme made it difficult to provide a constant URL. Therefore, we supply this page: <http://abstract.cs.washington.edu/~chrislim/skylin.php?path=> as a redirection to whichever server is currently operating (where the path variable is set to the path from the web root of the server). We used four servers over the course of our development:

<http://kiska.cs.washington.edu:8080/> <http://kiska.cs.washington.edu:9080/>

<http://umnak.cs.washington.edu:9080/> <http://attu.cs.washington.edu:9080/>. There is one caveat to this redirect: pages that require more than one parameter (i.e. extractInfo) will have the extra parameters stripped by the redirect page, so you must enter in the server's url directly when making calls to this method.

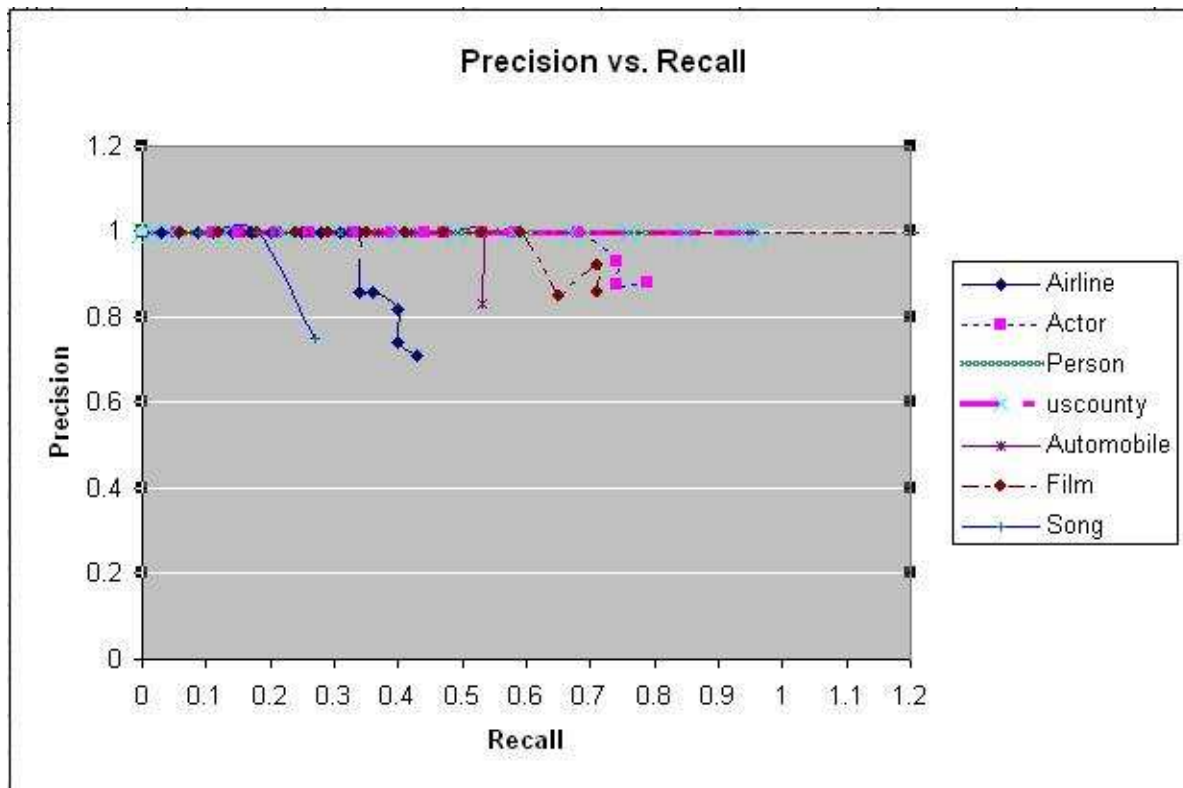
## Experiments

We extended Skylin to many other classes including: film, automobile, person, song, actor, and airline. For each of these categories, we analyzed their precision and recall by looking at the confidence probability for each attribute and only accepting certain ones if they are above a chosen threshold. By adjusting the threshold, we can get different points of the precision-recall graph. **Graph 1** shows the precision recall graph

we calculated from our results. For more precise numbers, we also provide a table, **Table 1**, that contains some of the data that we computed. As shown in **Graph 2**, the precision of our CRF extractor is relatively high and rather stable regardless the variation of recall. Among these seven categories, Person and uscounty are two categories with the highest precision value. It's probably because those two categories have lots of numerical but only a little alphabetical data, and numeric attributes are easy to be extracted. Another reason for that could be the sentences in those articles are well formatted and well structured compare to others. On the other hand, Airline category has relatively low precision-recall compare to other categories. It's a category mix with numerical and alphabetical attributes, but alphabetical data is dominant. Most of numerical attributes are correctly extracted, but some alphabetical attributes, such as callsign, could trick our extractor. For example, in the following sentence, "Our Airline is wholly owned by the state and has 144 employees ( at March 2007 ).", our extractor extracts "Airline" as callsign with 71% confident probability, but the actual callsign is "Our Airline". Moreover, the Song category contains lots of numerical data, and it seems like the extractor has hard time to extract those data. When we looked at files with all classified articles with the real data and the extracted data, we found out the extractor misses lots of relevant data and lots of data extracted from articles are actually incorrect. As a result, the song category has the lowest precision-call, and it is the worst category we experimented so far.

| Threshold  | 0.3   | 0.4   | 0.5   | 0.6   | 0.7   | 0.8   | 0.9   |       |
|------------|-------|-------|-------|-------|-------|-------|-------|-------|
| Airline    | 0.71  | 0.71  | 0.71  | 0.74  | 0.74  | 0.82  | 0.86  | 1     |
|            | 0.43  | 0.43  | 0.43  | 0.40  | 0.40  | 0.40  | 0.36  | 0.057 |
| Actor      | 0.88  | 0.88  | 0.875 | 0.93  | 0.93  | 1     | 1     | 1     |
|            | 0.79  | 0.79  | 0.74  | 0.74  | 0.74  | 0.68  | 0.58  | 0.053 |
| Person     | 1     | 1     | 1     | 1     | 1     | 1     | 1     | 1     |
|            | 0.857 | 0.857 | 0.857 | 0.857 | 0.857 | 0.857 | 0.857 | 0.071 |
| uscounty   | 1     | 1     | 1     | 1     | 1     | 1     | 1     | 1     |
|            | 0.96  | 0.96  | 0.96  | 0.94  | 0.94  | 0.94  | 0.94  | 0.015 |
| automobile | 0.83  | 0.83  | 1     | 1     | 1     | 1     | 1     | 1     |
|            | 0.53  | 0.53  | 0.53  | 0.53  | 0.53  | 0.53  | 0.53  | 0.053 |
| film       | 0.86  | 0.86  | 0.86  | 0.86  | 0.86  | 0.92  | 0.92  | 1     |
|            | 0.71  | 0.71  | 0.71  | 0.71  | 0.71  | 0.71  | 0.71  | 0.059 |
| Song       | 0.75  | 0.75  | 0.75  | 0.75  | 0.75  | 0.75  | 0.75  | 1     |
|            | 0.27  | 0.27  | 0.27  | 0.27  | 0.27  | 0.27  | 0.27  | 0.091 |

**Table 1** - These are the precision recall values for different classes of articles at different thresholds on the probability measure.



**Graph 1** - Precision vs Recall for different classes (categories) of articles.

These results reaffirm the already established belief that numerical data is much more easily extracted than textual data. We plot the U.S. County again as a reference for a highly numerical infobox. This category essentially stays at 100 percent precision regardless of the recall. On the other hand, Song is a highly alphabetical infobox. In this case we see that there is a very high trade-off between precision and recall. There is one final type of infobox, a mix between textual and numerical, such as the Person infobox (birth\_date is numerical and spouse is textual). In these cases, we still have consistent high precision, but recall in general suffers since extracting out the textual information is harder.

Another interesting observation that we made was that crossing two different, yet similar infobox categories together can yield many false positives. This was particularly apparent when using the Person infobox for an Actor. The problem can be seen in the death\_place attribute. Many actors will die in a certain film they were in. When this happens, the extractor will interpret this death as a real life debt. This is not really the fault of the extractor since the text in the document will literally say "so-and-so died in ...".

## Surprises

The greatest surprise was the difficulty of reproducing Kylin. Since we had full access to Kylin's author Fei Wu, we thought that it would be simple to get Skylin running and we could focus our attention on the other goals. Unfortunately, the complexity of the software made reproduction of Kylin one of the most difficult tasks of the project.

First, without a solid foundation of how the java-based Mallet library (which Kylin used) performs its classifications/extractions using Maximum Entropy and Conditional Random Fields, it was very difficult to understand Kylin's process flow. Classifications were easier to understand since we had performed classifications ourselves with the Naïve Bayes classifier and MaxEnt used a similar data format. However,

CRFs were completely new and required a great deal of preprocessing and sentence labeling.

Second, Kylin had nearly no documentation, which made it extremely difficult to understand its inner workings in order to reproduce its results. We understand that Kylin is still a work in progress, so documentation is not readily available, however, its actual code flows from class to class and tracing the execution of the software is nearly impossible for someone who was not involved in its development.

Finally, Kylin had a complex architecture, which made it difficult for us to bring all the necessary pieces properly together in order to simply run the software. Since it was originally a research project, the implementation was aimed for getting data and results as quickly as possible on a particular research machine. This meant that much of the code was specific to the the environment on the research computer (where the software was developed). Our project however was on an entirely different server. This required a great deal of communication between our group and Fei in understanding what the code was doing at certain points so that we could adjust them to our needs.

Even though we ran into these surprises, having the Kylin base was still a great benefit since we were able to leverage Fei's knowledge of classifiers and especially CRFs.

As we planned out the Skylin Web Service, it became apparent that we should develop it in Java because Kylin was written in Java. However, this was another big surprise because it required us to use the Apache Tomcat webserver, which no one in our group had experience with. There was a rush to have at least a couple members understand how to use Tomcat and JSP technology so that we could have our desired interface and service.

The last big surprise was the response we received from the Wikipedia community as we attempted to obtain permission to build the SkylinBot. There were apparently two 16-year-old contributors who had over 10,000 edits, who rejected our request to build such a bot because we did not have a sufficient edit history to verify the good intentions of our project. We therefore had to revert to the interactive command-line SkylinBot as a proof-of-concept in connecting the results from Kylin through the Skylin Web Service to a program that can re-input machine-learned edits into Wikipedia.

## ***Areas of Change***

One major area we would have chosen to do differently would be to put more effort to understanding Kylin right from the start. We had made the wrong assumption that we would be able to understand Kylin easily and quickly. This only led to much more work than anticipated, rushed project development, and unbalanced distribution of understanding.

Closely related to this, we believe it would have more beneficial for our own learning as well as understanding Kylin to have met with Fei for longer periods of time at earlier stages of the project to go over in depth details of how Kylin really performed classification and extraction in the code. By only speaking to him at a high level, we lost a great opportunity to understand how to use Mallet more effectively and how to perform Information Extraction well in general.

## ***Conclusions***

After working through the project and producing results, we found that numerical data has much higher



precision than textual data, as suspected before. But, we also discovered that crossing between two categories can be troublesome, especially if they are very similar. Crossing from person to actor was the clearest example, but this problem will probably come up in other cases as well. Therefore, it is very critical that the right infobox is chosen for a particular article or find ways to join classifiers together in such a way that confusions are minimized. However, this also points to the potential of having a class hierarchy where an actor belongs to a person class, inheriting attributes from that class while supporting additional actor-related attributes. A second interesting result was the increase in precision/recall on the same categories that were studied in the original Kylin paper. Between the Wikipedia dump used in the paper and the one we used, human users of Wikipedia had improved the quality of data by properly including US Counties in both the list page and tagging them with appropriate categories, or filling in more attribute information for these categories, or even creating totally new articles. This demonstrates an interesting dynamic between human users and machine agents and how they can work together to accelerate the development of the semantic web and the quality of its data.

## **Future Work**

### **Extending Recall**

One particular area that much work can still go into is extending the recall of finding articles for a certain category. In our project, we were unable to perform this task, but explored two possible approaches. First, we could use Skylin's current method of finding articles of a certain category (using the special title "List of ..." articles) as a training set and train a classifier (such as Naïve Bayes) on these articles. Then the classifier can be used on other articles to see if they belong in the same category. Second, it is highly likely that articles of a given category contain some sort of category tag and may also appear on the list page. Assuming this is true, we can easily find the union of all pages with a particular category type and all pages referenced in the list of pages and present all pages that are not in the intersection of these two sets to users, allowing humans to decide if a page belongs to the category in question. All pages that are approved would be added to the list page and tagged with an appropriate category tag. This way, the quality of Wikipedia's information would be improved and Kylin's current heuristic for classifying pages would achieve very good results.

### **Parallelization**

Another area of work is parallelizing parts of the Skylin system: when raw wikitext is pulled from Wikipedia to be broken into individual sentences a Sentence object must be created for every sentence in every article and it must be examined to find a potential infobox attribute match. If we could take the Wikipedia articles and create a sequence file with the title as a key and raw wiki content as the value, hadoop (or another map reduce system) could take each article and perform the matching in parallel. This was an originally planned task, however, we discovered that performing a classification and extraction took a few seconds and training extractors on the Wikipedia dump typically took on the order of a few hours. Since extractors would not need to be retrained often (only as frequently as new dumps are released), the time and effort necessary to parallelize Skylin did not seem worth it in the short term. There are however several key areas that can be parallelized and refactoring the code to improve its modularity and portability would be beneficial. Currently, the software is very much interconnected with the University of Washington Computer Science network and pulling out configurable parameters into files would help the software run more consistently in various environments.

### **Concurrent Requests and account locking**

A related code-change deals with the Skylin Web Service's ability to run multiple threads concurrently to serve requests. Because of Skylin's usage of the filesystem, multiple concurrent calls to the web service's methods may have interleaved instructions and result in corrupted results or data. Future work could resolve

these problems by properly implementing monitors and other synchronization primitives to ensure valid access to key resources. Additionally, we could leverage a login/API key system to make sure intermediate results are tagged (named) with unique session identifiers to prevent concurrent calls to Skylin from overwriting each other's intermediate output. In fact, if Skylin were to become a publicly available web service, we would need to implement an account system with assigned API keys to manage the load on the service. We would also want to engineer a caching system to increase performance for articles that have already been processed. A live service would also need to have a scheduled update process in place for retraining classifiers and extractors whenever a new Wikipedia dump is made available so that it remains up to date.

### **Wikipedians**

Finally, there needs to be further interaction with the Wikipedia community about developing a truly autonomous bot that can roam pages and update infobox information (or even generate infoboxes or new attributes) by using the Skylin service. This is a key component in bootstrapping the semantic web and would be beneficial to the Wikipedia community with the right safeguards in place. Developing such a bot may take much time and care, but all the building blocks are already available to make such an agent. A key aspect of this bot would be fully unsupervised, autonomous writing back of infobox data into Wikipedia for categories in which Skylin performs exceptionally well such as US Counties. Thus the bot would occasionally crawl Wikipedia articles of a particular class (category), and create or update certain attributes in each article's infobox.

### **Bot Extensions**

The initial implementation of the SkylinBot deals only with adding missing infobox properties to pages with pre-existing infoboxes. An obvious next step is extending the bot to handle the case where an infobox needs to be created from scratch. This would be done when another part of the skylin system has identified a page that *should* be in a category, but currently is not. A second interesting extension is enabling the bot to deal with inconsistent information between the body of the text and the infobox. One interesting case is when two values (one in the info-box, one in the text) *used* to be in sync, but no longer are. There is actually a very significant amount of complexity in this case, because doing it correctly involves analysis of the change logs as well as the text itself. Change log analysis is significantly complicated by the presence of reversions, and other common oddities.

### **Opportunity to generate training data**

Given that the SkylinBot must be run in a supervised mode for the near future, there is also an opportunity to use those supervised runs to generate additional training data. Each time a person makes a decision about an extracted property being incorrect this should be captured, and used as a negative example in retraining the extractors and classifiers, and each time one is accepted, it should be used as a positive example. As we consider building a browser plugin that lets users' confirm proposed edits, we should be sure we gather this data. In order to be effective, it must be on a per-item basis for rejections.

## Appendices

### Division of Labor

Chris White

- SkylinBot
  - Interaction with Wikimedia API for retrieving and writing pages
  - Wrote parsing and writeback code for the actual infoboxes (Complicated by internal links, multi-lists, and other constructs)
  - Interaction with the various Skylin web method endpoints
  - Interaction w/ the Wikipedians to figure out how to create and run SkylinBot without ruffling feathers
- Skylin Webservice architecture
  - Defined the webmethod endpoints
  - Defined broken up implementation plan for webservice (including some areas we did not get to, such as results caching and page caching.)

Ting-You Wang

- Kylin
  - Became groups' Kylin expert
  - Reproduced Kylin results in our new environment
  - Generated Kylin classifiers/extractors using Skylin
- Skylin
  - Designed and delivered an interface for the web service to access data and perform extraction
  - Wrote interface for connecting and accessing the DB for session management to make the service able to handle multiple requests
  - Generated output for the SkylinBot to use
- Web site
  - Codeveloped the skylin website
- Operations Management
  - Managed the MySQL database
  - Managed and set up the various Tomcat servers
  - Managed the group source control, and group folders
- Documentation
  - Added sections to Experiments, Areas of Change, Surprises, and Future Work

Christopher Lim. S.D.G.

- Skylin Web Service
  - Provided detailed design for the Skylin Web Service (internal architecture, class structure, actual output, etc.)
  - Implemented the Skylin Web Service
  - Developed and implemented additional web method endpoints that became necessary (e.g. providing service statistics, page listings belonging to a particular category).
  - Setup web application configuration for running in Tomcat (including source and deployment organization)
- Web site & Documentation

- Wrote Usage Scenario & Web Service Installation/Usage Documentation
- Developed the website: <http://abstract.cs.washington.edu/~chrislim/skylin.php?path=/skylin>
- Wrote sample application to demonstrate usage of the service (it shows a Wikipedia page and the information Kylin extracts from it side-by-side)

Luo Pan:

- Generated results of Skylin and calculated the precision-recall for various categories.
- Gather data and analyze them in our experiment
- Explored different approaches to improve recall (an area we did not get to)

Note: Our group suffered from a lack of consistent meeting times to work together. Communication was largely done by e-mail and brief communications after class, which made it difficult to measure our progress and continue moving forward with our milestones. There was a big crunch during the few weeks at the end of the quarter to bring the pieces together. Perhaps individual e-mails with you would be the best way to discuss group dynamics.

## External Sources

We used several external libraries and other code for our project:

- Kylin + Mallet
- Tomcat
- JQuery javascript library
- A CSS web template designed by gorotron.
- DotNetWikiBot Framework
- Wikimedia API
- We took examples from various code snippets on the web, which are documented in the code.

## Instructions

We include instructions for how to setup the Skylin web service on a java node running Apache Tomcat. We also include how to setup and use the SkylinBot software. We assume that Kylin has already been setup at a location accessible by the Tomcat server because many documents must currently be referenced by an absolute pathname. Visit <http://abstract.cs.washington.edu/~chrislim/skylin.php?path=/skylin> to view a website that hosts a running Skylin service and for additional documentation.

- Install the Skylin Web Service
  - This guide assumes that Kylin has already been installed (and trained) and is at a location accessible to the Tomcat server you will run Skylin on.
  - Our configuration is running Skylin on Apache Tomcat 6.0.14 with Servlet/JSP spec 2.5/2.1. Tomcat is running on a Linux server with MySQL installed and with NFS directories mounted so that our serialized Kylin extractors and related data would be accessible to the web service. Currently, Skylin has certain paths hardcoded to fit our current configuration, so it only works within the University of Washington Computer Science department's network on servers that have access to the `/instr/projects/07au/cse454/projects/e` directory.

- Go to your Tomcat server's web application manager
- Find the deployment field stating "Select WAR file to deploy" and browse to the skylin.war file provided by our project. Click deploy.
- Now view the main webpage for the project at your Tomcat server's web application URL under the /skylin path (e.g. <http://yourserver.com/skylin>).
- Using the Skylin Web Service
  - Using the web service can work on most modern OSB (OS+Browser) configurations, but it was only tested on the following:

| <b>Operating System</b> | <b>Browser</b>                      |
|-------------------------|-------------------------------------|
| Windows XP              | Internet Explorer 7 and Firefox 2.0 |
| Windows Vista           | Internet Explorer 7 and Firefox 2.0 |
| Linux Fedora Core 8     | Firefox 2.0                         |
| Mac OS X                | Safari 3.0.4 and Firefox 2.0        |

- When using Safari, the xml output of a extraction will not be displayed in the browser itself. To view the extraction, one will need to view the source of the page and the xml document will be stored there.
- See the Usage Scenarios section of this document to see an end-to-end usage of the Skylin Web Service.
- See <http://abstract.cs.washington.edu/~chrislim/skylin.php?path=/skylin/index.php?page=docs> for further documentation on using the WebAPI
- Visit <http://abstract.cs.washington.edu/~chrislim/skylin.php?path=/skylin/index.php?page=sample> to quickly view the service in action.
- Installing and using the SkylinBot
  - Unpack skylinbot.exe into a directory and run it from the command line.
  - Run with no parameters to see complete usage instructions
  - Some high levels:
    - SkylinBot -lc -- This makes the bot get the list of categories from the webservice.
    - SkylinBot -c <categoryName> -pp -- This makes the bot print out all titles that it would operate on (based on getting the titles from the service)
    - SkylinBot -c <categoryName> -sandBox -forReal -- This makes the bot iterate over all items in the category, and offers to commit changes (if required) first to the user specific sandBox, and then to the live site.
    - SkylinBot -c <categoryName> -stats -statName <fileName> -- This makes the bot iterate over all titles in the category, generating stats as to whether the bot thinks the infobox for that article needs to be updated. Results will be written to c:\stats\<fileName>.xml