

Entertainly.com - A Travel Search Classifier

Samrat Jeyaprakash
Emory University
samratjp@gmail.com

ABSTRACT

The information retrieval process for travel search usually involves a narrow sequences of actions including searching for travel destination(s), destination research, and a purchase decision. Entertainly.com is a travel search engine classifier that targets the searching part of the process and employs classification techniques to help end-users narrow down results that fall into all parts of those categories mentioned above.

Categories and Subject Descriptors

1.5.2 [Classifier design and evaluation]: In this project, I used a Naïve Bayes classifier to filter the results and employed a simple re-ranking method based off the bayesian values.

General Terms

Algorithms, Management, Human Factors.

Keywords

Naive Bayes, travel search vertical, Yahoo BOSS, Google App Engine, Python, Django

1. INTRODUCTION

Travel search is a very interesting monetizing problem for search engines and travel sites because of the expiring inventory nature of the industry. These travel websites, like any other business that depend on search engine forwarding, Search Engine Optimize (SEO) their websites for optimal monetization. As about the consumers, they are bombarded with these results and have a hard time narrowing down the results. This project gets results from Yahoo BOSS and classifies the results using a Naive Bayes Classifier that learns automatically and has relevance feedback. In essence, Entertainly.com's utility can be coined as an entertainment muse, an inspiration tool for travelers and entertainment related searchers.

2. Project Design

Inspired by the simple yet utilitarian interface of Google, Entertainly.com uses a simple yet easy to read interface for the frontend. The Interface has a base search page and a results page with classification divided throughout.

The whole project uses Python based frameworks. The backend and the frontend is written completely in Python and Django, a python based web framework. One of the most salient features of the project is its scalability. Due to the nature of its search engine aspects, the project can accommodate a highly scalable traffic because it is setup on Google's App Engine servers and uses a non-hierarchical, data store.

The classification is done using an open-source python based, Naive Bayes classifier called Reverend (alluding to the Reverend Bayes).

The search engine is built using Yahoo's Build Your Own Search Engine service called BOSS. Entertainly also uses Yahoo's Query Language or YQL to create dynamic filters to query some additional social networking based sites.

3. Experimental Approach

3.1 Design

I used an iterative approach to the project because of the consumer focus approach. I ended up using two different classification schemes based off the feedback I got in my first iteration (each iteration took me 10 days and a total of two and a half iterations).

In my first iteration, I used an open-source python project called Django-springsteen to query Yahoo and a handful of other social bookmarking/sharing sites. I also used Django-springsteen because of its excellent thread and resource management and I plugged in my own frontend and was able to modify it for deploying it to Google App Engine. The first iteration contained five categories that narrowed down travel results by travel destination attractions such as adventure, nightlife, dining, nature and shopping. Of course, my interface was minimal and contained basic presentation. The control baseline was Yahoo BOSS Results.

The first iteration uncovered some major flaws with technical design on Google App Engine and some classification training data (discussed in detail under evaluation).

The second iteration took into account of classification flaws and included new training data (again, more details under evaluation).

For both the iterations, thirteen subjects volunteered to test out the system and do a survey (a Google forms survey). They were also informed that their queries would be logged after the evaluation and were guaranteed anonymity of the data collected (I even looked up to make sure that if this were to be a grant funded project, whether it would pass ethics committee standards and also asked a few scientists across different disciplines on their opinion :-)

3.2 Naive Bayes Classifier

As mentioned previously, I employed a Naive Bayes classifier for both iterations. I used a python library for both training and information retrieval. In hindsight, hierarchical clustering would have been better, but because I wanted to have relevance feedback and an user oriented project, I opted for Bayesian learning.

For obtaining a decent sample, I queried Yahoo BOSS for a few keywords that I thought were representative of each category in the first iteration. My training classifier looked at the title and the snippets of the first 1000 results for each keyword. I had a total of 5000 title + snippets in my training corpus in the first iteration.

I employed a similar strategy for the second iteration, but I added negating queries to narrow down my training data (I will discuss this further in results).

3.3 The App Stack

The app contained a simple stack. In the first iteration, I queried for data from Yahoo BOSS, Twitter, del.icio.us, Freebase, and Yahoo Local. I was able to extend an open source project called django-springsteen by writing my own models for Google App Engine, and rewrite the whole querying stack using Yahoo Query Language for better filtering. I also wrote my own scripts for gathering and training the classifier.

A sample run would for example be like this:

- 1) User enters a query and hits search
- 2) Django based form sends a controller request to my views (Django has an odd MVC naming), which loads my classifier data from the model and meanwhile, on a separate thread, Django-springsteen makes several calls simultaneous to the various APIs.
- 3) Numerous JSON based responses (50 results from yahoo + other APIs [3 responses each]) are retrieved and gathered. The results are sent through the classifier and the classifier returns a list of categories and their likelihood in a percentage scale (0-1 for easy formatting). The top category is the first in the list. My first iteration took a deep hit because of Google App Engine's lack of a filesystem and storing my files in the datastore was yet another nightmare.
- 4) The query is also logged on my Google App Engine server (also, the logs were stored on Google's data store which uses a simple, non-relational database like BigTable).
- 5) Now, the results are parsed and formatted for the user. Something so trivial yet was not helped with Google App Engine's slightly older versions of python and Django!

The whole stack was mostly the same for both iterations but on iteration two, I ended up moving my stack offline due to persistent problems with Google App Engine's datastore performance.

There was also a relevance feedback but was not tested extensively (see Discussion).

3.4 Evaluation

The hardest part for this study I found out was evaluation, not because it was a hard metric to measure, but because of finals time and a general tendency for surveys to sit unattended for a while. Fortunately, I was lucky enough to get my friends to take the surveys and test out the system from my laptop and theirs (I had my birthday during finals and it was an helpful event to get them to donate their time :-). I had a total of thirteen subjects (I had to drop two of them because they didn't either complete the survey or had insufficient data). Of my eleven subjects, I got a wide range of feedback on the results for the first iteration.

Before I discuss the feedback, I'd like to point out my metrics. I used a very simple measure for evaluation - number of hops to find an information need and the time it took. Though I could've included several browser and session level time measurement, I just used a simple offline tool called RescueTime that kept track of which app was being used at which time - I let my subjects have a basic task and gave them the address for the site (sometime it was deployed on Google App Engine like the first iteration and the second time around, I had it offline. The reason I wanted to use RescueTime was because it tracked all the tabs and windows and time spent on different apps - a poor man's real life scenario simulation, if you will. I also wanted to see if they decided to use

another source for checking my results with a comparable base such as Yahoo itself. At one point, I was using mouse flow (<http://mouseflow.com/>) to track my subjects, but that proved to be difficult and non-relevant because my interface was minimal (but it was good to know that such tools exist).

My evaluation consisted of a series of travel search tasks. Each travel search task was a one-liner information retrieval task such as "Find a bar in San Francisco" using both the baseline Yahoo and the Entertainly app.

3.5 Results

As I've mentioned in my proposal, I used Yahoo BOSS Search as a baseline. This seems to be a point of confusion in both my presentation and proposal and I'd like to clear it up.

It's true I had two baselines. I used Yahoo as the baseline in both iterations, but as I've mentioned I had to scrap my first data set and categories for an important reason - users were not not happy with the results and so I ended up using iteration 1 as a "terrible" baseline and to improve upon it in the 2nd iteration.

The results are summarized briefly below.

Table 1. Summary of # of Hops / Minutes Taken (average)

	version 1	version 2	Yahoo
Iteration 1	5.8/10	_____	4/6
Iteration 2	5.7/10	3/4	3.8/6

3.6 Discussion of Results

The first iteration classifier took a beating in ratings by the users. Of course, it was expected because many results were not classified properly in one of the five categories based by entertainment type such as nightlife, adventure, nature, dining, shopping. I discovered from the 5000 results of training data that it was pretty evident that these sites SEO their sites very much. The titles often had more than one mention of these categories and made it hard for the classifier to discern the difference.

To get over this in iteration 1, I even tried query expansion using a simple idea - get Yahoo's suggestions and query them and get up to 5000 results per category for training data. Of course, this proved to be challenging still, for the pseudo query/synonym expansion still brought similar results and the classifier had a field day with it. I had originally selected these classification categories because of the inspiration from www.tripbase.com. Unfortunately, my approach is nothing like theirs, for with theirs, they recommend cities, not classify results i.e. they give out travel inspiration ideas.

With my approach, I wanted to target consumers who already knew where they wanted to go but wanted to narrow down their travel search results to specific parts of their decision process. So, the second iteration contained four categories that were directly relevant to the purchasing decision process and travel research problem; the four were: guides, attractions, reviews, and deals. I got a total of 4000 snippets and titles for the training data.

Fortunately, the results were much more easier to classify for these categories and even the classifier had clear differences between first and second iterations for there was at least a 15% improvement in the results (I summed up all the result scores per

category per task - I didn't collect this information extensively for all the users, but I did take their queries from logs and ran them through my backend and did a quick sanity check to ensure my results were consistent or not).

Again, I believe that choosing newer categories helped and in the second iteration I forced some training data results to not contain too many SEO'd words that would kill my trainer initially by using negation. For example: "travel guides -buy" actually returned much better results for a training set and thus better guessing of results.

The free response part of Google form survey showed that in the first iteration almost everyone complained about the narrow categories and the terrible classification.

In the second iteration, the users much more happy and liked the results overall in terms of classification. I also developed my own re-ranking algorithm that worked fairly well. One approach in the first iteration that I used was to just punish more frequently occurring domains. But, in iteration 2, I decided to make use of the Bayesian scores for each category and sort them per category, which the users found to be a great improvement. Moreover, I think the whole search process was much speedier in iteration 2 because of this simple optimization idea and the users perceived the interface to be faster.

Moreover, I also wrote a relevance feedback that I implemented in the background but not on the interface itself; I tested this feature with 5 subjects by literally just asking them to report . I didn't demo this, for I was a bit apprehensive about getting to next exam that I had stage freeze and forgot what to demo next (sorry!) But, the relevance feedback was only marginally better by 5%. What I did was to take the index position of the result and send it through my Bayesian unlearning python module and update the central learning corpus. Again, this was a very nice to have feature that worked fairly well, but couldn't develop further because of devotion to more important features.

3.7 Conclusion

I definitely learned many valuable lessons with this project. One was that Python is a very prototyping friendly language with a great library base and was a great tool. But, on the other hand, I learned the perils of a framework. I spent more time fixing problems with Google App Engine and Django than any other bugs. I also was dubious of the classifier, so I spent a good part of my weekend just testing out the classifier with my own personal spam email and looking up some projects that used this python library.

As about the classification systems itself, I learned that I should have implemented an interface level relevance feedback. Overall, I do think clustering would have helped immensely, then Bayesian filtering of those clusters would have been a nicer hybrid strategy.

Overall, this project taught me a lot about working on a disciplined pace at this magnitude. I was glad to apply some software engineering principles that I had learned ranging anywhere from proper version control, iterating, a scrum approach with my user base. As the semester evolved, I very much wished I had discovered some of the more interesting topics a few weeks earlier. After learning about collaborative search, I went haywire about the idea of incorporating that into my project, but I just didn't want to change my variables and just stuck with it and finished.

Misc. Notes

Once again, my apologies for an incomplete demo and presentation on Wednesday, I was too nervous about getting to my next exam that I skipped around too much of my presentation. Perhaps, my biggest mistake was expecting Google App Engine data store to be easily accessible, but that wasn't the case. Overall, I really liked this project and I intend on fixing some App Engine quirks and fork it as an open source project.

I am including my last working gist from my git repository. This is the non-django-springsteen extension (although I did have 4 different deployed instances of that on my App Engine). I wrote the whole stack myself for the second iteration. If you happen to have django, you can run it locally if you'd like to test it out. "python manage.py runserver port #" Briefly, the samql.py was a prototype. Views.py is django's rendering system for the frontend (and is pretty much a glorified version of sam.py) The other files are pretty self-explanatory (my training data is in there too in plain text if you'd like to examine).

I've enabled verbose mode on, so when you query something on the webpage, terminal prints out everything. The scores are printed right after each category and shows them before being sorted (sorry, I just forgot to tag my git branches and couldn't find the right one with proper labeling - I just wanted to include this to show you that this does indeed should you need proof.

In essence, I am happy with the results and did learn a lot from my users. I do intend on cleaning it up and forking django-springsteen with a Bayesian Classifier and a poor man's relevance feedback. Once again, thanks for a great semester! I am so sorry that I've had to miss a few classes because of the family situation.