CS 224n Final Project
Information Extraction from Housing Advertisements

David Murray (dimurray), Josh Herbach (joshy), Rohan Jain (rohanj)
Department of Computer Science, Stanford University

June 3, 2009

Abstract

The increasing popularity of sites like Craigslist[7] means that more and more of the world's classified ads are available in a digital format online. A desire for users to be able to search these ads for some semantic meaning rather than be limited to a keyword search is natural and requires first the ability to extract semantic meaning from ads.

In this project we designed a system to extract semantic information from Craigslist housing advertisements. Our system is based on partitioning Craigslist ads into title and body segments, training and running two Part-Of-Speech taggers on the resulting segments, and finally combining the resulting chunks using a series of handwritten rules. The resulting model is able to reliably extract many common features with high precision and recall and offers decent performance on less commonly occurring features.

1 Introduction

1.1 The Problem

Craigslist.org is one of the web's most pervasive sites for finding and placing personal advertisements. Among the challenges users experience on this site are difficulty navigating, sorting, and prioritizing advertisements. For example, upon reaching the San Francisco Bay Area "Apts/Housing for Rent" section, the user is presented with a reverse-chronological list of links, each of which typically has a rent value and number of bedrooms visible in its title. After clicking a link, one views a largely unstructured posting which may or may not match what the user is looking for. There is no way to sort this information, categorize it into condos, apartments, and houses, or anything else of that sort. As a result, finding exactly what you are interested in is a time-consuming task.

One approach to this task would be to define a schema for an advertising category and then require posters to manually populate exactly this set of features. This imposes a degree of structure which in many cases might run counter to the wishes of the advertiser. It also puts the burden of organization on the poster, making the relatively simple task of posting an advertisement much more complicated. For users that post many advertisements at a time – like apartment owners and property agencies – this becomes an arduous task. Providing an API could enable automation of such tasks, but this requires technical expertise on behalf of the end user.

An alternative to this approach is to use natural language processing techniques to decipher this information from the text provided by posters, resulting in no burden on either the poster or browsing user. Our goal in this project is to do just that. By using NLP to extract the critical features of housing postings, we will enable column-by-column sorting, viewing, and filtering of this information helping to make the search experience on Craigslist much more positive.

1.2 Challenges

Given that we start with a website and somehow have to process it using NLP techniques, the first challenges is to take the data from the website and automate the scraping and tokenization of its content in some reliable way so that the resulting text could be fed into the core NLP portion of our system.

The really interesting NLP problems are determining what the key elements of a given posting are and how to identify them. Certain elements of a typical housing posting are simple to parse; phone numbers, dollar amounts, and words like “rent” and “deposit” next to these values make it relatively simple to identify this information (although of course there are plenty of examples of “rent 1400 deposit 1600” and “1400 rent 1600 deposit” which ensure context is not always easy to use). There are also many features which are much harder to parse. For example, most postings have a location embedded at the bottom. In some cases this is the exact address; in others, it is a general neighborhood. In addition, addresses are sometimes included in the posting itself, and they may or may not match the bottom location. Determining which location to trust is difficult.

Similarly difficult, rent and deposit costs can sometimes depend on buyer or renter specifics, including duration of lease and presence of a pet. These statements are declared in sometimes less-than-formal words, and they are not frequent across the postings on the site. As a result, it is difficult to train on annotated instances of this information, because the ways in which they appear linguistically are highly variable. Thus, processing these nuances is challenging.

Finally, there are a significant number of fields that one may be interested in. From smoking status to presence of a backyard, patio, or even a fireplace, the number of potential columns on a list is huge, but frequency of each of these can be relatively low, like presence of a pool. In terms of identifying information via training on data sets, one is more likely to be successful by focusing on elements that are both frequency and of interest to actual users.

2 System

Our overall system has four major components: (1) Scraping, Tokenization and Annotation, (2) Title Tagger, (3) Body Tagger, (4) Chunking and Merging. We examine each component below.

2.1 Scraping, Tokenization and Annotation

Extracting raw postings from Craigslist proved surprisingly easy as their HTML is well suited for screen scraping because of their relatively consistent use of certain reserved elements to delineate posting urls, posting titles and posting bodies. A simple python script handled extracting the HTML and dividing it into {posting url, posting id, posting title, posting body}.[a] We then made use of a freely available tool for converting HTML to text[5] and ran the resulting data through another python script to XML tag the posting id, title, and body.[b] We then annotated the resulting text using the Stanford NLP annotation tool[4] and finally partitioned the title and body text, and tokenized and tagged the sections[c] so that we could use them for testing and training our taggers. Two other scripts[d] were also part of this pipeline – namely a tool for combining a collection of examples into a single file for the tagger and a tool to strip the tags from a testing file before feeding it into the tagger). Most of the work of gluing together this pipeline and managing it was done by Josh. For more details on the actual data that was fed through this pipeline see Section 3.

2.2 Taggers

With our cleanly formatted and tagged postings in hand we trained a pair of Part-Of-Speech taggers. We decided to use a POS tagger rather than a basic classifier because of the need to use surrounding words to help identify a token's class (without additional context its impossible to know if '2' refers to the number of bedrooms, bathrooms, stories, or parking spots). The decision to use a POS tagger instead of a NER was based on our desire to try a different approach from previous work in information extraction on Craigslist ads.[6]

Rather than reinvent the wheel, we downloaded and made use of the Stanford POS Tagger (acronym aside, a very useful tool)[3] for our taggers and simply added additional features to help improve performance. The Stanford POS Tagger is a maximum entropy-based tagger which makes use of a weighted linear combination of features (generated by so-called extractors) to determine a token's tag.

Our decision to train two taggers was based on the intuition that the title and body text of Craigslist postings tended to be very different. In particular, we noticed that the title text tended to contain objective features (rent, count of bathrooms, etc) in fairly consistent formats whereas the body text had a tendency to be much more free-form and filled with lots of subjective descriptions. As a result we believed that with a very simple model the title text would be able to select with very high precision a number of features whereas the features needed for the body would be much more complex. Much of the work with error analysis of the taggers was done by Rohan and Josh and the work on new features was driven by Rohan.

---

[a] src/scrape/grab.py

[b] src/scrape/cleanup.py

[c] src/converter/annotated_to_tagger_data.py

[d] src/converter/combine_files.py and src/converter/detag_file.py

2.3 Chunking and Merging

The final component of our system was a small tool to take the output from our taggers, chunk sequences of tokens that shared a tag, and then choose the most appropriate chunk for each tag.[e] This portion of our system is largely rule based. For example if a tag only appears in the title (or conversely in the body) the chunk for that tag is used (and if there are multiple such chunks the most frequently occurring chunk is used). For cases where a tag appears in both the title and body text, we look for chunks that appear in both (with some normalization to help recognize equivalent chunks) and if no such chunks exist, we fall back on the most frequently occurring chunk from either tagger (with a slight upweighting for the title text based on our belief that it tends to be higher precision). Most of the work for the Chunking and Merging script was done by Josh.

3 Corpus and Data

Our original focus was along the four major housing sections in craigslist: "Apts / housing", "Rooms / Shared", "Sublets / Temporary", and "Housing Wanted". From reviewing several postings from each of these categories, we quickly discovered a high variance between sections with respect to what information people included and what they omitted. For example, in the "Apts / Housing" section, most postings were focused on the elements of housing, like square footage, beds, and bathrooms. In the "Rooms / Shared" section, however, this information was not always present, and many posts focused around searching for ideal roommates – in some cases, information on a particular type of housing was completely omitted. We thus decided it would be worthwhile to focus on a particular section, as this would allow us to rely on a generally consistent set of attributes to work with. In the spirit of housing, we focused on the most housing-centric section, "Apts / Housing".

We began exploring this section in more detail by enumerating each of the different classes or pieces of information that were discussed across an informal sampling of 10-20 postings. The following data set is the union of elements we found in this sample:

Type (e.g. condo)	*Monthly rent*	*Deposit amount*	*Bedrooms*
Bathrooms	*Size (sqft)*	*Furnished or not*	*Available date*
Contact email	*Contact phone #*	*Name of poster*	*Smoking*
Number of floors	*On a cul-de-sac*	*Has pool*	*Has air conditioning*
Garage/# of cars	*Parking spot*	*Address*	*Backyard*
Frontyard	*Fireplace*	*Patio*	*List of floor types*
Adjective list	*Dogs allowed*	*Cats allowed*	*Other pets allowed*
References required	*Credit check*	*List of outside links*	*Remodeled/Upgraded*
Washer/dryer	*Wireless internet*	*Walk-in-closet*	*Sites nearby*
Room list	*Utilities included*	*Year built*	*Home style*
Gas BBQ	*Breakfast nook*		

[e] src/merge_results.py

Upon further review, we reduced this set to contain mainly elements that were of high frequency across posts on the site and likely to be of high interest to most end users. Moreover, we decided not to include boolean attributes, like whether or not a credit check or breakfast nook was included, as our system architecture was incapable of recognizing such features. In the end, our final set of attributes we chose to focus on was as follows:

Type	*Rent*	*Deposit*	*Bedrooms*
Bathrooms	*Size*	*Available date*	*Built year*
Floors	*Email*	*Phone*	*Address*

For our corpus we downloaded 500 postings. Of these, we identified 10 postings in our corpus that were not appropriate for training or testing, as they were not categorized correctly. For example, the following is an actual posting in this section that we consciously removed from our data set:

Can't refinance?? Protect your Home. Easier than a Refi (brentwood / oakley)

Artafdmwf ekdpb omgywa uzwpmzwoce ekdpb hmva ekdpb rtafdmwf ekdpb.

Jyglbu jumukkwm vgn mvbwimmwcgvn vgn uakbep ucgenwnpoc yglbu.

We divided the 500 postings in the San Francisco bay area Apts/Housing section between the three of us and then to try and enforce consistency between our annotations David did a second pass through all of the postings to ensure uniformity. As mentioned in Section 2.1 we used the Stanford annotator to identify this information for use in training and testing. It is worth noting that many of the posts have several references to the same piece of data, so on occasion, multiple instances of bedroom and bathroom, for example, were identified in the annotation.

The resulting annotated data looked something like:

http://sfbay.craigslist.org/eby/apa/1176224104.html 1176224104
<title><tag name="rent" value="start"/>$1725<tag name="rent" value="end"/> / <tag name="bedrooms" value="start"/>2<tag name="bedrooms" value="end"/>br - Furnished Large Sunny Quiet 2nd Flr <tag name="bedrooms" value="start"/>2<tag name="bedrooms" value="end"/>-bed Avail <tag name="available_date" value="start"/>5/17/09<tag name="available_date" value="end"/> (berkeley) (map)</title>

Fully Furnished large (<tag name="size" value="start"/>800+<tag name="size" value="end"/>sqft) Sunny 2nd Flr <tag name="bedrooms" value="start"/>2<tag name="bedrooms" value="end"/>- Bed apt., quiet building, quiet neighborhood, very private, hardwood floor, tasteful furnishings, lots of
light, two large bedrooms with 2 large closets, eat in kitchen, very quiet, including T.V., DVD, VCR, microwave, linens, kitchen supplies, toaster, kettle, coffee maker, and many other extras for a home away from home. Close to U.C., LBL shuttle, downtown BART, and shops. This is a quiet family centered neighborhood, very popular with postdocs and visiting scholars.
Located at <tag name="address" value="start"/>2315 Grant St<tag name="address" value="end"/>., at Bancroft. Pics available <tag name="available_date" value="start"/>May 17, 2009<tag name="available_date" value="end"/>. rent is <tag name="rent" value="start"/>$1,725 <tag name="rent" value="end"/>plus utilities. Will rent month-to-month and short or long term.
Call or email Bob at <tag name="phone" value="start"/>510-915-2288<tag name="phone" value="end"/> <tag name="email" value="start"/>rwr.korman@comcast.net
<tag name="email" value="end"/>

<tag name="address" value="start"/>2315 Grant St. at Bancroft<tag name="address" value="end"/> (google map) (yahoo map)

As can be seen, in cases like "3br" or "1600 sqft" we focused on annotating only the number. Despite our efforts to be consistent in our annotations and second reviews, we identified a number of issues in our annotations helping to demonstrate just how hard it can be to get good data from humans. Interestingly enough, we found these errors during our test phase when we looked at the 'errors' in our system and saw that frequently the errors were in the goldens themselves. We discuss this more in Section 4.

	P	R	F-Score
bathrooms	0.84	0.72	0.78
built_year	0.5	0.11	0.18
cars	0.83	0.51	0.63
deposit	0.71	0.46	0.55
bedrooms	0.82	0.69	0.75
O	0.97	0.99	0.98
phone	0.75	0.73	0.74
floors	1	0.27	0.43
rent	0.65	0.58	0.61
available_date	0.82	0.42	0.56
address	0.62	0.4	0.48
type	0.41	0.23	0.29
email	0.7	0.42	0.53
size	0.74	0.76	0.75

Table 1: Initial Body Tagging Results

4 Initial Model

In our initial exploration with the POS taggers we stuck with built-in architectures and extractors and we used wordshapes(1), bidirectional, generic, and prefixsuffix(3) for our initial exploration. We chose bidirectional because of the importance of context words, prefixsuffix and wordshapes primarily to capture some of the random character sequences embedded in some key tokens. generic similarly looks at some of the surrounding context as well as extended prefixsuffix features.

We then trained our taggers on the first 300 postings in our corpus and used the last 200 postings for testing. Unfortunately, given the uneven distribution of tags (the vast majority of our tokens were tagged as O), we found the accuracy numbers reported by the tagger to be basically useless. To work around this, we wrote another script[f] to scrape the results and produce precision and recall number for each tag. The results can be found in Table 1 and 2.

As expected the title classifier tended to have very high precision for some of the most common features such as bedrooms and rent but had no results for many of the tags which rarely appear in the title (in Table 2 the zero rows usually had between 1 and 5 examples). In contrast, the body tagger was able to identify tags for a lot more of the tokens although often with lower precision (although surprisingly it did better on bathrooms).

	P	R	F-Score
bathrooms	0.79	0.68	0.73
available_date	0.75	0.43	0.55
cars	0	0	0
deposit	0	0	0
bedrooms	0.91	0.96	0.93
O	0.98	0.99	0.98
phone	0	0	0
floors	0	0	0
rent	0.99	1	0.99
built_year	0	0	0
address	0	0	0
type	0.67	0.54	0.6
size	1	0.14	0.25

Table 2: Initial Title Tagging Results

[f] src/compute_results.py

Passing the results into the final conflict resolution Chunker and Merger was then fascinating because it led to the question of "how can we evaluate these results" because in many examples there were multiple correct results in our annotation. Our solution to this was simply to accept a result if it matched one of the chunks marked as correct in the golden set.

	P	R	F-Score
bathrooms	0.97	0.82	0.89
built_year	1	0.13	0.22
cars	0.9	0.58	0.7
deposit	0.98	0.55	0.7
bedrooms	0.99	0.98	0.99
phone	0.96	0.71	0.82
floors	1	0.27	0.43
rent	1	0.99	1
available_date	0.76	0.33	0.46
address	0.83	0.25	0.38
type	0.76	0.51	0.61
email	0.75	0.6	0.67
size	0.93	0.63	0.75

Table 3: Initial Final Results

Even with this method of evaluation, we found our initial results to be somewhat poor. However, when we started analyzing the results we discovered that in many cases this was because of annotation errors. For example, we often had our model choose a string like "apt" or "apartment" as the type when the golden set marked other answers like "Apart" or "Apt" and failed to mark the "apartment" or "apt" strings. Other obvious issues we saw included numerical mismatches between "1600" and "1,600.0", or between "1800 sq. ft." vs "1800". We even identified a few cases of a string being mis-tagged (a handful of street addresses were marked instead as email). To more fairly evaluate our model we introduced a series of equivalences classes. With these extra modifications to the merger's evaluation tool, we found that the combination of the two models often produced better results than the individual models (although this begs the question of whether the taggers are being unfairly evaluated because of bad annotations and no mechanism to detect equivalence classes). In particular we found that the final model tended to yield extraordinarily high precision for most of our features. Table 3 contains the final results.

The few features we did very poorly on were mostly because of low recall. In the case of built_year, we suffered from the problem that there were only ~10 instances in the test set. For available_date the problem was something different. available_date tends to come in many vastly different formats such as {date/month, immediately, now, first of the month, mid-April, ...} making it hard to identify a canonical format. Similarly, floors and cars tended to vary between numbers and those same numbers spelled out (e.g. one, two, etc), and sometimes cars was non numerical (e.g. street parking). Another feature with highly variable formatting was phone which could be anything from 123.456.1234 to (123) 456-1234 or 123-456-1234. As for address, a careful analysis of the data shows that our results tend to just focus on substrings of the golden address explaining the low recall. We also did a terrible job of detecting emails despite there being many clear signals a token is an email address.

Even for some of the features we did very well on, the error cases presented interesting challenges. For example, deposit often was expressed as "one month's rent" which is a perfectly reasonable answer, but we would hope the deposit could be easily expressed as

a number. Another interesting case where our simple approach to looking at context broke down was “rent and deposit are $1895 and $1000 respectively”.

5 Extensions

Based on our analysis of the results from the basic model described in Section 4 we considered a number of extensions, both in terms of additional features and in terms of changes to our preprocessing of the postings.

- To detect emails better, we added extractors that detected the presence of an @ sign, the presence of ‘.net.’, ‘.com ‘or ‘.edu’ in the word.
- To distinguish between rent/deposit and size we added an extractor to identify the presence of a $ sign in front of the word.
- To address our poor performance on built_year we added a feature that detected whether the word contained a number between 1800 and 2000.
- To improve the performance of address, we added features that detected whether the word was one of ‘Street’, ‘Parkway’, ‘Boulevard’, ‘Road’ or ‘Avenue’ and all their common known abbreviations.
- To address the issue of numbers being formatted differently (one versus 1) we created a second dataset in which we converted the written out form into the decimal one.

To make the comparison simpler we only present the results for the addition of our new extractors although we do discuss the impact of using the dataset with numbers converted into decimal form.

	P	R	F-Score
bathrooms	0.95	0.83	0.89
built_year	0.5	0.22	0.31
cars	0.87	0.55	0.67
deposit	0.69	0.54	0.61
bedrooms	0.9	0.77	0.83
O	0.97	0.99	0.98
phone	0.9	0.76	0.82
floors	1	0.27	0.43
rent	0.74	0.66	0.7
available_date	0.87	0.45	0.6
address	0.7	0.45	0.55
type	0.46	0.25	0.32
email	0.73	0.48	0.58
size	0.74	0.78	0.76

Table 4: Body Tagging Results with Extended Extractors

	P	R	F-Score
bathrooms	0.79	0.71	0.75
available_date	0.75	0.43	0.55
cars	0	0	0
deposit	0	0	0
bedrooms	0.91	0.96	0.94
O	0.98	0.99	0.98
phone	0	0	0
floors	0	0	0
rent	0.99	1	0.99
built_year	0	0	0
address	0	0	0
type	0.64	0.53	0.58
size	1	0.14	0.25

Table 5: Title Tagging Results with Extended Extractors

As shown in Table 4 adding the extra extractors discussed above often improved our body tagger's performance dramatically. The F-Score of bathroom increased from .78 to .89, the F-score of deposit increased from .55 to .61, and the recall of built_year doubled (although given the number of examples this means we just correctly identified a couple of extra strings) to name a few. In no case did a feature's F-Score lower with the additional extractors; although floors was not affected. Because the new extractors were

designed for rarely occurring features which mostly appeared in the body text, the additional extractors had no impact on the title tagger as shown in Table 5.

	P	R	F-Score
bathrooms	0.97	0.85	0.9
built_year	0.67	0.25	0.36
cars	0.88	0.58	0.7
deposit	0.96	0.56	0.71
bedrooms	0.99	0.98	0.99
phone	0.97	0.77	0.86
floors	1	0.27	0.43
rent	1	0.99	1
available_date	0.81	0.38	0.52
address	0.91	0.3	0.45
type	0.75	0.5	0.6
email	0.76	0.65	0.7
size	0.9	0.61	0.73

Table 6: Final Results with Extended Extractors

In terms of their effect on the final results of our model we found that the new extractors' impact was somewhat reduced given the already excellent performance of the title tagger. As shown in Table 6 we did see notable improvements in F-Score for built_year (.22 to .36), phone (.82 to .86), address (.38 to .45) and email (.67 to .7). In all of these cases the improvement came from increasing recall which makes a lot of sense given the intuition behind the newly added features. Like with our experience with our basic model, in looking at the final results we found that many of the false positives were in fact still results of annotation errors.

As for our other extension, we found the impact of converting written out numbers to decimal numbers to be relatively small. For the title tagger, the only effect of this extension was to slightly reduce the precision and recall scores for bathrooms. In the body tagger its effects were a mixed bag. It did improve the recall score for cars (although at the cost of a lot of precision), but it reduced the precision from floors with no corresponding improvement in recall. It also had slightly negative effects on available_date and type.

6 Conclusion and Future Work

It is clear from our results that we have demonstrated a working system that applies NLP techniques to the real-world problem of extracting meaningful features from Craigslist housing postings. Our system offers 90+% precision for almost all categories of interest and for many of the most common and important categories (e.g. rent, bedrooms, bathrooms) offers very high recall.

Nevertheless, our system falls short in a few ways. For one, in our architecture there is no easy way to identify boolean features (e.g. is there a pool) because there are so many different ways to negate a word many of which are not going to show up in the text in a sequence which our architecture requires. Another major issue is that because we have no understanding of the meaning of words our system is incapable or taking a line like “deposit is one month's rent” and making the connection with the result for rent. It would be interesting to explore deeper semantic elements of the postings to discover this information (say through parsing and semantic role labeling).

Also, although we improved performance via the addition of many new extractors to our taggers, it is clear that one of our system's major failings is its low recall for many features. To address this shortcoming there are a number of additional features we considered adding but did not have time to implement. For example there are a number of regular expressions we could use to improve recall of phone numbers and built_years (for built_years "\d\d\d0's" is one potentially useful extractor). We could have also used features such as detecting a word that is 5 digits in succession to identify zip codes for addresses. It might also be useful to assemble additional sets of word lists (e.g. {townhouse, condo, apartment}) and make features to detect memberships in those sets. These word lists could also include a set like {one, two, three, four,...} to try and capture our intuitions about spelled out numbers without making them appear exactly the same as a series of digits.

One of the best things we could do to improve our recall numbers, and help the wider research community, would be to produce a set of well-annotated Craigslist postings. With a series of correctly annotated postings we would expect our recall numbers to improve, and if we produced a much larger corpus we would hopefully overcome some of the issues of sparsity we encountered with our rarer features.

Another area to explore more is our method for combining title and body tagging results. Because our system was restrictive and relatively manual it would be interesting to investigate a more automated approach. For example, a system that made use of validation data to learn relative per-tag weightings between body and title chunks would be interesting and potentially have a lot to offer, especially for features where the title tagger has near perfect performance. Another interesting extension would be to have the tagger output a probability for each tag it assigns and then make use of those values in our merging,

7 Acknowledgement

We would like to thank Professor Manning and all of the TAs for CS224n for their advice and feedback during the development of our project. We are especially appreciative of the references to already-existing Stanford software used to classify and tag data; this allowed us to focus on the core problem

8 References

1. "The Stanford Natural Language Processing Group". 2009. Stanford University. 3 Jun. 2009 <http://nlp.stanford.edu/software/index.shtml>.
2. "The Stanford Classifier". The Stanford NLP Group. 2009. Stanford University. 3 Jun. 2009 <http://nlp.stanford.edu/software/classifier.shtml>.
3. "The Stanford POS Tagger". The Stanford NLP Group. 2009. Stanford University. 3 Jun. 2009 <http://nlp.stanford.edu/software/tagger.shtml>.
4. "Simple manual annotation tool". The Stanford NLP Group. 2009. Stanford University. 3 Jun. 2009 <http://nlp.stanford.edu/software/stanford-manual-annotation-tool-2004-05-16.tar.gz>.

5. “HTMLasText v1.06”. NirSoft. 3 Jun. 2009 <http://www.nirsoft.net/utils/htmlastext.html>.
6. Nipun Bhatia, Rakshit Kumar, and Shashank Senapaty. 2008. Extraction of Structured Information From Online Automobile Advertisements.
7. Craigslist. May 2009. <http://www.craigslist.org>.
8. Kristina Toutanova and Christopher D. Manning. 2000. Enriching the Knowledge Sources Used in a Maximum Entropy Part-of-Speech Tagger. In Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC-2000), pp. 63-70.
9. Kristina Toutanova, Dan Klein, Christopher Manning, and Yoram Singer. 2003. Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In Proceedings of HLT-NAACL 2003, pp. 252-259.