CS 224n Final Project
Name Extraction on Chinese Novels

Jing Chen and Raylene Yung

June 07, 2008

Description

Performing NER on Chinese is fundamentally different from performing NER on English text, due to differences in the language structure and representation. We chose to attempt to extract the names in Chinese novels, since we can use the repetitiveness of character and location names in novels to our advantage.

The problem can be stated as: Given the text of a Chinese novel, output a list of strings of characters that are thought to be character names or location names.

For native Chinese speakers, a list of characters from a novel may not be very useful, since they will be able to pick these out easily. However, for an intermediate Chinese student, reading novels can frequently be difficult because names can be difficult to distinguish from the rest of the text, particularly when the name consists of unfamiliar characters. Then having a list beforehand (or having those names automatically highlighted) can greatly speed up reading, allowing the student to focus on learning unknown vocabulary words.

Our initial approach was to extract the names using entirely unsupervised methods. However, we found that this required hand-setting several parameters in the system (such as thresholds and coefficients for combining feature scores), which made the results much less robust across all texts. Instead, we chose to use one training set to pick optimal parameters, and then examine the performance using those parameters on a separate test set.

As for existing work on this subject matter, most of the papers surveyed dealt with NER on much larger data sets, including those of colloquial Chinese as well as news articles. When specifically looking for personal names, their results often built probabilistic models based on prior probabilities of a given character appearing in a name. For example, one research result we studied used a large existing database of Chinese names in order to establish a prior probability that a given string is a Chinese name. In our case, we wished to provide as little additional information as possible to learn our character names. Additionally, since the data sets we chose mostly involved fictional or transliterated names, such existing databases of traditional Chinese names would not be that useful. From our brief survey of existing work, we saw results of around 0.7 for F1 scores. For our very specialized data domain, we were able to achieve recall scores of about 0.6, and even better precision scores. In the end, with limited testing of our features we were able to achieve an F1 score of about 0.57, which given our assumptions and limited data was very promising.

Procedure

In order to generate data sets, we took the text of the first two Harry Potter books in Chinese and created lists of the names of people or locations from the text for each book. Any mention of a person or location was included in the list, not just the characters that occur frequently. All of the names were transliterated separately for the first name and the last name (so that Harry Potter becomes 哈利。波特), and these were treated as separate names, particularly since characters were mostly referred to using their first name. Also, names that end in titles (such as 先生, which is analogous to "Mr.") were taken such that they did not include the titles.

The text files were encoded using UTF-8. These were read in as sentences separated by whitespace and stored as strings. The strings were stored in CounterMaps to give the bigram, trigram, and quadrigram counters.

The correct character name lists were stored in newline-separated text files. For testing, the output list of possible names was compared to the correct name list to give both precision and recall scores, which were then combined to give an F1 score.

We initially planned for the system to be completely unsupervised, using just one data set. After trying some features, we decided to use a training set to set weights on the features, and a test set separately. Thus, the first Harry Potter book was used as the training set, and the second book was used as the test set.

Because parts of the system was developed in parallel, it is difficult to give meaningful incremental results. In addition, some ideas turned out to harm performance after other pieces were added, even though they initially appeared to help (see the Postprocessing section). Constructing the correct name lists was also imperfectly done, due to misunderstandings of Chinese (such as not realizing that 著 is not part of an author's name), so the set of correct names sometimes had to be updated during development.

Thus, we will give ablation test results for the features, and show comparison results for disjoint features. We will refer to relative results in explaining our process for choosing features.

If we had 3 full data sets, a better method would be to use one as a validation test, and use that to choose the features and components to use in the final system. However, since producing the data sets is quite time-consuming, we chose a set of components that we felt would perform well, and tested with those, plus all of the features turned on. This way, we can avoid contaminating the results on the test set with knowledge of which ones work the best. The obvious extension to this project would be to add another data set, use the current test set as a validation set, and finally test the overall performance with the set of best components and features on the new data set.

In the end, most of the components that we originally picked actually turned out to do worse than their alternatives. However, it is hard to get a good picture of which features work best until the entire system is set to some baseline, and the components/features used are varied around the baseline.

The baseline is:

(a) Removing common characters is turned on.

(b) Use the word segmenter as a filter.

(c) Use the dictionary as a filter.

(d) Use F1 score maximization for thresholding.

(e) Do not use post-processing (merging, removing prefixes/suffixes).

(f) Turn on all of the features (except word segmenter and dictionary features).

Baseline Results				
Feature	F1 Score	Precision	Recall	Difference
Baseline	0.4809	0.4117	0.5779	0.00

In all of the tests below, the baseline result is always listed first. The difference in F1 score from the baseline is also listed. The best choice out of all of the listed components/features is bolded.

In the Conclusions section, we will give a list of the set of components and features that we believe would give the best overall results based on the performance results that we see in the tests below.

Preprocessing

Removing Common Characters

In general, we can expect that character and location names will not include extremely common characters, such as articles or pronouns. We built a list of common characters from a list of the most common Chinese characters. Since N-grams containing any these characters would be automatically eliminated, the list is fairly conservative.

We also want to ignore N-grams that include symbols (particularly punctuation) or numbers. To make the implementation easier, the symbols and digits were included on the list of common characters, so they are removed along with the common Chinese characters. The lists of N-gram strings were automatically scrubbed of any strings that had a character on this list before scoring.

Common Characters Tests				
Feature	F1 Score	Precision	Recall	Difference
On	0.4809	0.4117	0.5779	0.00
Off	**0.5039**	**0.4413**	**0.5871**	**+0.023**

We can note here that the overall score is actually better when removing common characters is off, which suggests that the names list actually has a few names that contain common characters (such as "了" and "人", even though we tried to make the list conservative. It is possible that this has some performance overlap with the word segmenter though, since the word segmenter does a very good job of eliminating common words that are not part of a name from N-grams.

Using the Word Segmenter

Many of our guessed names were real Chinese words, which was solved by the dictionary feature (see below). After these strings were eliminated, the output had a lot of strings that were normal Chinese phrases, but they crossed the boundary between two words. For example, we saw strings like "是什么" (which roughly translates to "is what"), where the trigram is not a dictionary word, but is clearly a common Chinese phrase, and not a name. This is a case where the word segmenter can help boost performance, since it would segment the phrase into "是 | 什么", thus eliminating the more common phrases.

We used the word segmenter from Pi-Chuan Chang (http://nlp.stanford.edu/software/segmenter.shtml) as a preprocessor. Because we had some difficulty using the newest version, we chose to use the 2006-05-11 version. The output of the word segmenter was a text file with the segmented words separated by spaces. A

quick scan of the text indicated that the word segmenter was generally very good at grouping the characters together that belonged to names, and separating it out from surrounding characters. Empirically, over 80% of the actual names were indeed returned as segmented words by the segmenter.

We first read the word segmented text into a set of all the unique whitespace-separated strings, which are all the distinct words the word segmenter considered to be separate words. We initially threw out all N-grams that were not in this list. This meant that any names that the word segmenter always segmented incorrectly would be thrown out.

Upon examination, the word segmenter did separate names into more than one word for some cases. For example, the translation for King's Cross Station is 国王大道车站. This is segmented (correctly) into three words, since the words are translated directly here (国王 = king, 大道 = avenue or large street, 车站 = station). Thus, this is never seen in the segmented text as one word.

A common mistake was on a particular section of the first book. Harry Potter is given a list of books, and when translated this list contained several author names. The author's name was often followed by the character 著, which means "author". For many of these, the name was split up incorrectly. For example, "沃夫林著" became "沃夫 | 林著", when the correct segmentation was actually "沃夫林 | 著". This caused the name to be eliminated completely.

Thus, we tried using the output of the word segmenter as a feature rather than a preprocessing measure. This would make the cases where the word segmenter segments a name (correctly or otherwise) have less impact on the result. This turns out to be the best choice, as we would expect, since using it as a feature allows for more flexibility around mistakes that the word segmenter makes.

Word Segmenter Tests				
Feature	F1 Score	Precision	Recall	Difference
As Filter	0.4809	0.4117	0.5779	0.00
As Feature	**0.5173**	**0.4466**	**0.6146**	**+0.0364**
None	0.4927	0.4071	0.6238	+0.0114

Dictionary

One of the issues is that Chinese words can often be bigrams, where the first character is almost always followed by the second character in common usage. For example, the word "东西" means "thing." The two characters individually mean "east" and "west", but when used in a bigram, "东" almost always comes before "西", particularly if the words are segmented. Since our frequency and ratio of occurrence feature looks for exactly this (frequently occurring bigrams where the unigram nearly always occurs with the bigram), many common Chinese bigram words were appearing in the output list.

We added a feature that checked a dictionary for the n-gram strings. We used the CEDICT dictionary, obtained from here:

`http://www.mandarintools.com/cedict.html`

This gives a list of roughly 44,000 Chinese words (ranging in length from 1-4 characters, at least), and their definitions. We parsed out the Simplified version of the words, and added these to a HashSet. Then each string could be looked up in the dictionary, and removed if it was in the dictionary.

This has the same potential problem as the word segmenter, where names could appear as words in the dictionary. Thus, we tried using the dictionary as a feature as well.

Dictionary Tests				
Feature	F1 Score	Precision	Recall	Difference
As Filter	**0.4809**	**0.4117**	**0.5779**	**0.00**
As Feature	0.4598	0.3818	0.5779	-0.0211
None	0.4466	0.4742	0.4220	-0.0343

The result is that the dictionary performs better as a filter than as a feature, possibly because the dictionary provides noise. Note here that if the dictionary is used as a feature, the precision goes down, but the recall stays the same. This indicates that it may interfere with the thresholding method, causing the threshold to be set at a less optimal value, so that the same words are output, but a few more words are output than it would not have output otherwise.

Determining Parameters

Logistic Regression

Initially, we experimented with only a handful of features and used hand-set constant values for their weights. However, once we started to increase the number of features as well as consider N-grams as distinct classes (bigram, trigram, quadrigram), it became clear that we needed a more efficient and correct way to compute feature weights. To do this, we implemented logistic regression to return optimized theta-parameters for a given set of feature value vectors. For training data, we computed feature vectors for all examined strings in the training set, along with a corresponding score vector. The score vector consisted of 1s and 0s, where a value of 1 corresponded to a string that was an actual name, and a 0 otherwise. We then ran logisitic regression with stochastic gradient ascent. At first, we continued iterating until the successive difference in theta-values was below some threshhold, but this proved to be very time-consuming. Subsequently, we chose to iterate until the value of the difference itself converged. This greatly decreased the number of iterations, with little to no effect on performance. Finally, we experimented with other constants (such as the learning rate) to find the best possible combination.

Thresholding

One of the problems involved in outputting a list of names is determining where to stop outputting the names. In particular, this threshold needs to be independent of the data set, since data sets vary quite widely in terms of number of actual names. We initially used hand-set parameters for the purpose of testing features out, but this is clearly not applicable for general data sets. We tried two methods for setting the threshold, one unsupervised and one supervised, and compare their results below.

Otsu's Method This is the first method we tried, which is commonly used in computer vision to differentiate the background pixels from foreground pixels in an image. It does not require training to set a threshold. Instead, it finds a threshold that best separates a set of points into two groups, which we then applied to the output of the test data.

The motivation behind this is that if logistic regression is able to find proper coefficients for the feature vectors, the names should be separated from non-name strings quite clearly. If this is the case, Otsu's method has a good chance of finding the boundary between the two classes.

Our implementation took a list of the scores for all the N-gram strings (where N = 2, 3, or 4), and produced a threshold that maximized the between-class variance for the two classes that the threshold split the data into. This gives one threshold for bigrams, one for trigrams, and one for quadrigrams. This is needed because the different N-gram scores have different scales, and thus are not comparable. Using one threshold for each is more reasonable in this case.

Once a score is assigned to each N-gram string on the test set, Otsu's method was used to find the best threshold to set. Any string with a score higher than or equal to this threshold would be classified as a name, and strings with lower scores were discarded. The composite list of strings from all three N-gram lists with a score higher than the threshold was the final output.

The resulting performance of this was lower than the hand-set parameters for thresholds, but this is expected since Otsu's method is done with no knowledge of the correct answers, or even the scores for the correct answers. Given that is quite blind with respect to the data, its performance is surprisingly good: it is only a few points below the hand-set parameters for the thresholds.

Maximize F1 Score The second method was to find a threshold that maximized the F1 score on the training set. The idea here is that a good threshold for the training set is likely to be near the correct boundary given the logistic regression coefficients. Thus, using the same threshold on the test set should also work, particularly since that threshold does not depend on any features of the test set, but rather only on the actual scores.

Then for each N-gram set, the threshold was set to each one of the scores in the list of scores, and the F1 score for the training set was found from this. While this may seem performance intensive, the number of correct words can be updated whenever the threshold is moved by the number of correct strings that move across the boundary, thus avoiding calculating this number from scratch for each possible threshold value.

Note that we have two choices here for calculating the F1 score. Since the score is calculated for only the set of bigram strings, etc, the recall rate can actually be considered to be $\frac{number\ of\ correct\ bigrams}{number\ of\ actual\ bigram\ names}$, rather than $\frac{number\ of\ correct\ bigrams}{number\ of\ correct\ names}$. Since this score is not used to measure performance of the system, we could reasonably use either one. The difference is whether precision is emphasized more or less in the search for the optimal threshold.

As we can see from the table below, maximizing the F1 score actually performs better (and also better than Otsu's method) if it uses the number of actual bigram names in the denominator for the recall rate. This means that it tries harder to increase the precision, since the recall rate is higher than it would be if the number of actual names was in the denominator. Otherwise it weights the recall rate too much in maximizing the F1 score (because it cannot possibly be higher than $\frac{number\ of\ actual\ bigrams}{number\ of\ correct\ names}$, and the result is that the precision of the resulting system is worse than it should be.

One thing we noticed when looking at the output of the F1 maximization method of thresholding is that it was too sensitive to differences in the data set. For example, the training set had far more names than the test set did (166 vs 109). (This is partly because the first book listed all of the authors in a list of textbooks that Harry Potter needed to buy, but the same list in the second book consisted largely of books by the same person.) There were also more names in the training set that were mentioned only once (such as historical characters), which skews the threshold toward names that may have lower scores, because of low occurrences.

When this method was used to train on the first book and test on the second book, it gave roughly the correct number of names in the output (105 found vs 108 actual names). When the two data sets were switched, it gave too few names for the result (70 found vs 165 actual names). Although the precision is very high, the recall is very low. The overall effect is a higher F1 score, but this indicates that the threshold set by the F1 method is too sensitive to differences in the data, particularly inherent differences in the training set and the test set.

Otsu's method does not have this problem because it finds a threshold based on the distribution of scores for the test set, and does not depend on the training set at all. However, we expected it to do worse than the F1 maximization method because it did not use information from the training set. Instead, we had the complete opposite result:

Thresholding Tests				
Feature	F1 Score	Precision	Recall	Difference
F1 Maximization	0.4809	0.4117	0.5779	0.00
Otsu's Method	**0.5252**	**0.5842**	**0.4770**	**+0.0443**
100 words per N-gram	0.3520	0.2400	0.6605	-0.1289

(Note: We compared thresholding with setting a 100 word limit for each N-gram. We don't expect it to do well at all, nor is it a good method overall, but we need some method of comparing thresholding versus no thresholding.)

Otsu's method does surprisingly well when compared to the F1 maximization method, given that it doesn't use trained information. However, it is more robust overall, and is probably the better method to pick. We did notice that if we used fewer features (rather than all the features), then Otsu's method does much worse. This means that both methods have their drawbacks: F1 does badly if the number of names varies widely across the training and test sets, and Otsu's method does badly if there are not enough features to separate the output into two classes.

Features

The logistic regression is based on a set of features extracted for each n-gram string. Each of the following features were used in the feature vector.

Frequency and Ratio of Occurrence

These features attempted to capture the relationship between a given n-gram and it's embedded (n-1)-grams. For example, for a given trigram, this feature would compute the ratio between the number of times the trigram appeared and the number of times the prefix bigram appeared: $\frac{trigramCount}{bigramCount}$. The intuition behind this is that for a trigram that is a name, the prefix-bigram should really only appear as part of the entire trigram, and not as a stand alone ngram. Thus, the ratio would be higher for actual names versus arbitrary trigrams. Finally, we weighted the feature by a scaled version of the (n-1)-gram weight, in order to re-scale between very common and uncommon trigrams. The main feature tested was to compare an n-gram with it's prefixed (n-1)-gram, but we also implemented one with the suffixed (n-1)-gram for comparison.

Transliterated Characters

Since the data sets were texts translated from English, many of the character names are transliterated directly, using the phonetic sound of the name. Transliteration from foreign languages to Chinese generally uses the same set of characters. This set of characters has been standardized, so that translations use the correct numbers. We used the following webpage as a source for the list of characters that are commonly used in transliteration:

`http://zh.wikipedia.org/wiki/Category:%E5%A4%96%E8%AA%9E%E8%AD%AF%E9%9F%B3%E8%A1%A8`

These characters were converted to Simplified form, and then added to a newline-separated file with one character on each line. Then this feature, for each n-gram string, consisted of the number of characters in the string that are on this list.

Word Segmenter

For a motivation of using the word segmenter as a feature, see the section "Using the Word Segmenter" above. Each n-gram string was given a score of 1.0 for this feature if it appeared in the list of segmented words, and 0.0 otherwise.

Appears in Dictionary

For a motivation of using the dictionary as a feature, see the "Dictionary" feature above. Each n-gram string was given a score of 1.0 for this feature if it appeared in the dictionary words, and 0.0 otherwise.

Word Counts

In the frequency and ratio features mentioned above, we used n-gram counts to compute the appropriate feature values. For these features, we instead ignore n-gram counts and look solely at the set of segmented words. From the original list of segmented words (duplicates included), we compute the counts for the n-gram as a whole, as well as for the (n-1)-gram prefix and suffix. The rationale for this is that if a given n-gram is often segmented as a word by the word segmenter, it is more likely to be a name as compared to an arbitrary n-gram that is rarely segmented. Additionally, for a perfect word segmenter, we would hope that both prefix and suffix (n-1)-grams should have low counts for an actual n-gram name.

Prefix/Suffix Classification

For the next four features, we simply added binary features for whether an n-gram's prefix or suffix was a segmented word or in the dictionary. As mentioned previously, all binary features were given the value 1.0 for a true case, and 0.0 otherwise. Intuitively, we would expect that for an actual name, both prefix and suffix strings would neither be segmented nor in the dictionary. However, this makes assumptions about the accuracy of the segmenter as well as the characteristics of the names (i.e. that names are not built of actual dictionary words).

Mutual Information

This feature is probably most applicable to the bigram case, where we compute the mutual information between the first and last characters. One version uses counts from the LanguageModel (which tallys general n-gram counts) and the other comes from using the set of segmented words (as in Word Counts above). The idea behind this feature was derived from some existing work on Chinese NER, as well as from some of Pi-Chuan's previous work on news documents. We would expect that for a correct name, the mutual information between the two characters would be higher than for any arbitrary bigram. For the higher n-gram cases, we computed the mutual information between the beginning (n-1) gram prefix and the last character.

Unfortunately, by the time we had this feature implemented and working, we had already run most of our ablation and other post-processing comparison tests. Thus, we decided to run these features on top of the baseline results to guage their efficacy. In all of the results, unless otherwise specified, mutual information features are *not* considered to be part of the feature set. The results for tests included these features are included in their own section.

Ablation Tests

We can see from the ablation test table below that most features had little impact on the overall performance, with the exception of feature 6. The most likely reason for this is that many of the features overlap, so that

removing one does not remove all the information, so the performance does not suffer too much. In many cases, the performance is actually improved by removing a feature, and this is most likely because too much information in a certain area may cause the logistic regression to overfit to the training data, so that removing one of the overlapping features reduces this overfitting.

(The bolded ones below are the ones that are farthest from the baseline.)

Ablation Tests				
Feature	F1 Score	Precision	Recall	Difference
Baseline (All features)	0.4809	0.4117	0.5779	0.00
Feature 1 (Freq and Ratio)	0.4809	0.4117	0.5871	0.00
Feature 2 (2nd Character N-1 Gram)	0.4866	0.4155	0.5779	+0.0057
Feature 3 (First Char is Transliterated)	0.4814	0.4203	0.6055	+0.0005
Feature 4 (Last Char is Transliterated)	0.4864	0.4200	0.5779	+0.0055
Feature 5 (Number of Transliterated Characters)	0.4978	0.4750	0.5229	+0.0169
Feature 6 (N-gram Word Count)	**0.4219**	**0.3906**	**0.4587**	**-0.059**
Feature 7 (Prefix Word Count)	**0.5019**	**0.4333**	**0.5963**	**+0.021**
Feature 8 (Suffix Word Count)	0.4758	0.4000	0.5871	-0.0051
Feature 9 (Prefix in Dictionary)	**0.5019**	**0.4436**	**0.5779**	**+0.021**
Feature 10 (Suffix in Dictionary)	0.4620	0.3809	0.5871	-0.0189
Feature 11 (Prefix in Segmented List)	0.5000	0.4580	0.5504	+0.0191
Feature 12 (Suffix in Segmented List)	0.4883	0.4228	0.5779	+0.0074

Since the individual ablation tests did not help much in showing which features were useful, we re-ran the tests with related sets of features taken out, to give an idea of how much each characteristic actually helped.

Group Ablation Tests				
Feature	F1 Score	Precision	Recall	Difference
Baseline (All features)	0.4809	0.4117	0.5779	0.00
Minus Transliterated (3, 4, 5)	0.4705	0.4109	0.5504	-0.0104
Minus N-gram Word Counts (1, 2, 6)	**0.1045**	**0.1818**	**0.0733**	**-0.3764**
Minus Prefix (7, 9, 11)	0.4754	0.4038	0.5779	-0.0055
Minus Suffix (8, 10, 12)	0.4575	0.3827	0.5688	-0.0234
Only N-gram Counts (1, 2, 6)	0.4682	0.5000	0.4403	-0.0127

We can see from this that the actual N-gram counts accounts for most of the score, since leaving only those features gives a score that is pretty close to the baseline. Taking it out reduces the performance by a large amount.

Mutual Information Feature Tests

Here result are given by adding the two mutual information features on top of the previously run tests. Here the MI-1 feature refers to MI computed using n-gram counts, while MI-2 refers to MI computed using word counts.

Mutual Information Results				
Feature	F1 Score	Precision	Recall	Difference
Baseline	0.4809	0.4117	0.5779	0.00
+Feature MI-1 (N-gram)	0.4824	0.4189	0.5688	+0.0015
+Feature MI-2 (Word Count)	0.4822	0.4236	0.5596	+0.0013
+Both MI-1, MI-2	0.4938	0.4477	0.5504	+0.0129

Here we see that using any MI feature improves performance, and using only both MI features gave the best results. Unfortunately, it is hard to compare this to our other results because they were achieved when compared to a different baseline. Given more time, we would have re-run all of our tests using the new baseline in order to evaluate the true efficacy of these features. Interestingly, these features decrease recall when compared to the baseline, but all increase precision. This could be related to the idea that more features leads to better gradation of scores, and better threshholding.

Analysis of Feature Results

Although we had prior intuitions about the relevance of our features, after looking at both the performance results as well as the learned feature weights, we found that many of our initial beliefs were not exactly true. Since we were looking to maximize n-gram scores, the value of a given feature's weight can be thought to be correlated to the importance and effect of the feature. Unfortunately, since the individual feature values (for non-binary features) were not normalized, it is hard to compare relative magnitudes of weights across them. However, the sign is still a useful indicator, as is the magnitude compared across binary features.

Here are some of our learned feature weights:

Baseline Feature Weights												
N-gram	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12
B	0.11	-0.64	0.81	0.78	0.59	2.75	-1.46	-0.18	-0.50	-0.67	1.29	-1.03
T	-0.08	0.35	-0.09	0.10	0.89	2.81	-0.38	-0.82	-0.41	-3.14	-2.31	0.66
Q	2.07	-0.04	2.68	1.90	0.12	1.41	-0.54	-0.89	-0.44	1.12	0.04	-0.50

Baseline (+ MI-1 and MI-2) Feature Weights														
N-gram	F1	F2	F3	F4	F5	F6	F7	F8	MI-1	MI-2	F9	F10	F11	F12
B	-0.039	-0.77	0.91	0.70	0.62	2.32	-1.23	-0.25	1.94	0.13	0.54	0.24	0.52	-1.54
T	0.42	0.15	-0.03	0.16	1.20	3.34	-0.25	-1.07	1.10	1.54	-0.52	-4.83	-1.90	1.12
Q	1.72	1.24	2.84	2.14	0.28	1.76	-0.29	-0.37	2.86	1.00	-0.51	2.51	-0.05	-1.10

The following analysis is done only using the parameter values for the original baseline, while the baseline + MI features is simply included for completeness.

Looking at families of features, we see that the first group 1, 2 and 3(freq and count ratio) gives interesting and varied results across the three n-gram types. Of the three, quadrigrams wear this feature most heavily, which makes sense in that a quadrigram's 3-letter prefix probably carries more significance than a bigrams single first character. In terms of the second group (4, 5, 6), transliterated characters, clearly the number of transliterated characters is a positive feature for all types of n-grams (all three features have positive weights). Interestingly, bigram and trigram names weight the total number of transliterated characters most heavily, while quadrigrams give the first character's transliteration property an unusally heavy weight. Looking at the next grouping of features of 7 and 8, we see that they are negative features across all n-gram types. This group corresponds to prefix and suffix counts, which means that the more frequently a prefix of

suffix occurs, the less likely the given n-gram is a name. This goes along with our intuition that for an actual name, the prefix and suffix should not be very frequent when compared to the whole n-gram. The last four features are binary features that reflect whether an n-gram is segmented or in the dictionary. Interestingly, here we see that it is more useful to look at the results by analyzing each type of n-gram separately. For bigrams, having either character in the dictionary is a negative feature, while having just the first character a segmented word is a positive feature. Additionally, the relative magnitude suggest segmentation is a more important feature for bigrams than the dictionary. For the trigram case, the most important features are having either the suffix in the dictionary or the prefix a segmented word (both negative features). This may be related to the examples of accidentally attaching particles to names. In the bigram case, since the prefix and suffix were single characters, this possibility was not considered. In a trigram case, we could have a bigram word with an attached particle or verb being interpreted as a name.

Postprocessing

Merging Words

The bigram, trigram, and quadrigram cases will only give up to strings of length four. However, there are many names on the list that have lengths greater than four, particularly for characters with long English names and for locations. For example, "Voldemort" becomes 福尔得摩特. These names are missed completely, because the system does no account for them. On the other hand, iterating over all unique strings of length five would be a very large space, making it impractical.

By looking at the output, we can see that many quadrigrams have their prefix or suffix showing up in the trigram output list. A common one was "卫文卡罗" (Ravenclaw), where both "卫文卡" and "文卡罗" showed up in the trigram lists. There were also cases (although less common) of names with 5 characters where parts of the name show up in the trigram or quadrigram lists. Therefore, we tried the simple technique of merging the resulting names if they had sufficient overlap (two characters for trigrams and three characters for quadrigrams, etc), and adding the result to the list.

This process occurred after the list of possible names had already been constructed, so it only added names to the list, and did not remove any names. The result was that the F1 score went up slightly overall, because the recall score would increase, but the precision score decreased, since it added many quadrigrams that were not actual names, because the trigram names just happened to overlap. Removing the added strings that never actually appear in the text improve performance slightly, but there were still many cases of incorrect merging, such as merging 荷米恩 ("Hermione") and 米恩说 (the last two characters of "Hermione" and the word "said"), since the action followed the character name quite often.

This method was added while using hand-set coefficients for the scoring, where it appeared to help performance slightly. However, after implementing logistic regression to find the coefficients for the feature vector, this method turned out to hurt performance slightly. Upon examining the results, we could see that adding this method increased the recall rate, since it was adding a few words that were missed in the selection process, but it decreased the precision rate to the point that it cancelled out the benefit on the recall rate. It was adding character strings that were often character names followed by a verb, including several strings where the same character name was followed by different verbs.

This behavior makes sense though, since the logistic regression caused the system to output roughly the correct number of names, and adding names, particularly incorrect ones, decreased the precision. In addition, the logistic regression gave more suited coefficients for the feature scoring, so that it missed fewer names, thus this addition did not boost recall as much, making the merging even less useful overall. Therefore, we removed this addition after logistic regression was implemented.

The most likely explanation for why merging did not help much after logistic regression is that the logistic

regression phase did much better on the quadrigrams than the hand-set coefficients, leaving fewer words for the merging to be useful on. Also, the higher order n-grams (5 or more characters in length) did not show up often enough for their quadrigram prefixes and suffixes to show up in the list of most likely quadrigrams. Another issues is that these higher order n-grams were most likely segmented into separate words by the word segmenter, which made the quadrigram prefixes and suffixes less likely to be chosen.

Removing Prefixes/Suffixes

One of the problems with the merging was that it introduced redundant strings, so that the first three characters of a correct quadrigram still showed up in the resulting list. In order to alleviate the problem, we added another postprocessing step that removed the prefixes and suffixes of longer strings from the output list. In other words, if a string like "abcd" appeared in the output list, then both "abc" and "bcd" would be removed from the output list, if they appeared.

The naive implementation always removed the shorter string. While this worked well on the training set using hand-set parameters, and improved precision by a fair amount, this did not work in general. The reason is that the training set rarely had cases where the shorter string was actually the correct one, but the test set had more of these cases (the character-action string example above was quite common).

A more intelligent version would take the string with the higher score. However, the scores were not comparable between trigrams and quadrigrams, since the feature coefficients were different, and thus the distributions would be different. A trigram with a higher feature score is not necessarily more likely to be a name than a quadrigram with a lower feature score. Even if it happened to work for the data sets that we used, this would not be a robust method in general (nor would it be well-founded), so we chose to not implement this version.

Since the naive implementation did not help in general, and for the same reasons that merging was removed, this addition was also removed. It seemed that the logistic regression did a pretty good job optimizing the output, to the point where postprocessing did not have much to add.

Removing Prefixes/Suffixes Tests				
Feature	F1 Score	Precision	Recall	Difference
None	0.4809	0.4117	0.5779	0.00
Merging	0.4790	0.4090	0.5779	-0.0019
Removing Prefixes/Suffixes	**0.4920**	**0.4335**	**0.5688**	**+0.0111**

Merging words does slightly worse, in that it seems to add a few words that are not correct, while it doesn't change the recall score. Removing prefixes/suffixes improves precision by a fair amount, indicating that it removes incorrect words that are part of a longer word, but the recall also goes down, which indicates that it is removing the correct word sometimes. Since we used the naive implementation, it makes sense that recall would go down. However, removing prefixes/suffixes could be used to improve performance if it was implemented more intelligently.

Conclusions

From the test results above, we would hypothesize that the following set is the best set of components and features for the system. The ones in italics are the ones that are different from the set that we chose at the beginning of testing.

(a) *Removing common characters is turned off.*

(b) *Use the word segmenter as a feature.*

(c) Use the dictionary as a filter.

(d) *Use Otsu's method for thresholding.*

(e) *Use removing prefixes/suffixes as post-processing.*

(f) *Turn on all of the features, except for feature 11, and the dictionary feature.*

We also turned off the prefix features for the bigrams only, since they seemed to hurt performance. This makes sense, since the bigram's prefix is one character only, and does not really provide enough information. This system, tweaked using the tests above and the bigram features, gives the following results:

Optimal Components Results				
Feature	F1 Score	Precision	Recall	Difference
Baseline	0.4809	0.4117	0.5779	0.00
Optimal Components	**0.5714**	**0.6750**	**0.4954**	**+0.0905**

We would like to stress here that *these numbers are tweaked to the test data set*, and are not indicative of how good the system is. However, it is still interesting to note that the optimal set of components and features increases the precision dramatically, while dropping recall slightly. It outputs fewer words, which may indicate that Otsu's method (which is not used in the baseline set) is better at finding a good threshold robustly, as long as it has enough features to work on.

We saw a lot of interaction between the different features (for example, removing common characters overlapped with using the word segmenter, even though they appear to be disjoint). Thus, it is difficult to predict which components are good in isolation from the rest of the components. If we used fewer features, Otsu's method would do worse than the F1 maximization method, since it may stumble on score distributions that clump around more than two centers. However, if we used different data sets, we suspect that Otsu's method would fare much better (and be less likely to make mistakes in the number of words to output) than the F1 maximization method.

Note also that a few of the names from the correct list were longer than four characters in length, which we did not track because of the performance hit that would be involved. Since merging failed to help in this case, none of the output included names with more than four characters, so these names were missed completely.

By examining the output of the guessed names, we saw that most of the names that were missed were the minor characters and rare locations - names that appear very rarely, and sometimes only once in the entire text of the book. It seems likely that while the N-gram count features contribute highly to the performance, they also outweigh the other features (which may still catch the minor character names) by far too much, so that the minor names are pushed off the charts.

One idea to solve this problem is to create a new list for the training set of just the minor characters, and train a separate set of coefficients for these. Then there would be two sets of outputs on the test set, one for the major characters, and one for the minor characters. Then the N-gram counts would no longer outweigh the other features on the minor characters.

Although we mainly dealt with transliterated names here, the features used can also be applied to general novels by changing the list of transliterated characters to a list of common Chinese last names. Some further work might include a more exhaustive testing of our feature set, introduction of new features, more thorough

labeling of data, intelligent handling of minor characters and location names, and normalizing non-binary features just to name a few.

Overall, we feel that with the limited time we had and the relatively low-quality of hand-labeled data, we were able to produce a very promising name recognition system.

Acknowledgements

We would like to thank Pi-Chuan for letting us use her word segmenter, which is available here: `http://nlp.stanford.edu/software/segmenter.shtml`, as well as Paul for advice on training features and thresholding.

Workload Distribution

Jing: Creating data sets, some features, post-processing, dictionary, thresholding, most of the writeup

Raylene: Reading in data, most features, logistic regression, word segmenter, code maintenance and refactoring, some of the writeup