Identification/Classification of Chinese Proper Names

Jin Zhao, Kun Li, Qingwen Ye, Erik Andersen

How to Run the Code

Requirements:

Scipy >= 0.13.3

Numpy >= 1.8.2

Scikit-learn >= 0.20.2

Matplotlib >= 1.12.0

Installation with native pip (for python3):

pip3 install --user -U scikit-learn matplotlib

pip install namesex

Type *python3 master.py* into the command line.

Goals and Motives

The goal of this project was to create a system that identifies Chinese proper names based on defining characteristics found in these names. While this is rather straightforward in English as proper nouns are indicated with capitalization, Chinese does not have any case distinction, so this problem requires much more investigation.

Design and Task Division

We split the project into four parts: Chinese surname extraction(by Jin Zhao), given name analysis(by Kun Li), place name classification(by Qingwen Ye), and foreign names identification(by Erik Andersen).

Part I: Chinese Name Extraction

(Code Found in surname_detection.py)

In this part, our task was to extract the Chinese names from a segmented text. We adopted a purely rule-based approach to locate those names. The approach is largely surname driven because of two reasons. The Chinese surname characters make up a relatively closed set that we can loop through. Though there are certain patterns appearing in given names, it is much more flexible than surnames in multiple aspects, hence harder to create rules for them. Besides, a lot of the surname characters are strong indicators of the presence of a name. Some surname characters are potentially ambiguous, which means that it could mean other things other than names, but there are still a considerable amount of surname characters only appear in environments that are Chinese names .

There are two types of Chinese surnames: single-character surnames (e.g. "赵") and double-character surnames (e.g. "欧阳"). We put all the surnames in the dc_surname file. First, we grab a word from the text and decide if it's a name. To do that, we loop through the surname character lists and find if the word contain any surname character. If it doesn't, we decide that word is not a name. If it does contain a character or two characters from the surname character lists, we decide the word could potentially be a name.

After we got the word that could be a name and its index number, we further checked the context of that word. There are three types of segments that appear most frequently on the left side of a name: 1. Punctuation. Names can be subject of the sentence, therefore they can be find in sentence initial position, this means that it follows punctuation immediately, such as period, comma and so on. 2. Pre-title. Like foreign names, Chinese names also sometimes follows title words such as "先生（Mr.)，女士(Madam)". Other pre-title words as "总理(president)，市长 (mayor)" indicating occupation or social status are also frequently found preceding names. 3. Verbs that subcategorize for noun phrases that are usually person such as "授予，称赞，会见". Another common position for names to appear is object position. Words that appear as the direct object of those types of verbs are usually names.

There are four types of segments that appear most frequently on the right side of a name: 1. Punctuation could again be an indicator, because when the name is an object, it could mean the end of the sentence. 2. Post-title such as "老师，先生". 3. Verbs with certain radicals such as "命令，说，做，打，通知". Given the logographic characteristic of Chinese writing system, some verbs that containing a radical “扌”(literally meaning hand, indicating the meaning of action taken by human), “口”(literally meaning mouth, indicating the meaning of speaking) always have a subject that is a human. Hence names are usually found preceding those verbs. 4. Possessive morpheme （的）， conjunction words （和，与，连）are also usually found with noun phrases that are persons.

Those frequent surrounding words are put into the vocab.title file. We took the words that contain surname and grabbed the word before it and word after it by index. We compared the word we grabbed with the surrounding words list. If there was a match, we concluded the word under investigation is a Chinese name.

We tested the program with train.snt file. The program picked out all the 7 Chinese correctly, but also 3 incorrect nouns. Those three nouns contain potentially ambiguous surname characters, and it also passed at least one of the surrounding word rules. One of the wrong name picked out because it precedes the possessive “的”; the other two passed because they precede punctuation. Apparently that none of the rules is completely reliable in all situations. Some of them are more reliable in some situations. In the future, I would put weight on those different rules to make some of the rules more important than others to achieve better result.

Part II: Chinese Given Name Analysis

Before you run the code, please install namesex.

(Code found in given_name.py)

In this part, we worked with the Namesex package that predicts gender tendency of a Chinese given name. This file uses the training data from the package, which contains 10730 Chinese given names with labels from public data. At the start of the data extraction, the training data is not accessible directly from the package. By changing the origin source code downloaded from the python package index, the training data is formed into a txt file and we divide the data into a training set and testing set. We choose the NaïveBayes classifier to train the data. Since Chinese characters itself each have special meaning, we make each character as its own feature and if the given name has two characters, we will make two of them as two separate features. After we finish training, the test case can achieve 89% of accuracy distinguishing male or female names. From the result, we find that if we test given names that were popular from the early 70s to 90s, we can achieve very high accuracy, but if we test names that are more common nowadays, the accuracy seems to go down. When we analyze the result, we conclude that the training data is old and parents in China nowadays do not care about the sexuality of the name as parents did in the past. Therefore, the challenge right here is that in modern China, when parents give a name to their children, sexuality is not as great a concern. Some parents will give a girl a male name, which makes the task more difficult. In the future, we might need to analyze the context to determine the sexuality of a name. For example, we could extract occupation information from context to determine the sexuality of the name.

After determining the sex of the name, we dive into quality of the name. The rule to determine whether it is a good name is based on Chinese grammar. Each Chinese character represents special meaning, and if a sentence has two character represent the same meaning, then the sentence is redundant and it needs to be fixed. Same idea here when parents give name for their children. We are expecting there are no duplicated meaning in a name. In our code, we come up with five most frequent meaning that parents like, such as wealth, health, smartness, beauty, and good personality. We test each character and to see whether they are in the same category. If they are in the same category, we conclude that the name is a bad name. In the future we would like to add more rules, for example the pronunciation of the name to determine quality of the name.

In our given_name.py file, the result of a name will be 1 if it is a male name, 0 otherwise. Based on the rule stated in last paragraph, if each character represents different meaning, then it is a good name, otherwise it will print out "it is not a good name". Last, we will show the accuracy of our detection of distinguishing the sex of the name.

Part III: Chinese Place Names

(Code in places.py & tools.py)

Introduction

In this section, we tackle place name identification and classification. Place names convey important information in various language contexts. Automatically identifying place names offers great social benefits as well as research values to tasks such as knowledge distillation and machine translation. However, the task of identifying place names in Chinese can be very challenging due to: (a) a wide range of characters used in place names and (b) their varying lengths from a single character (such as "京", the abbreviation for Beijing) to character combinations involving multiple nouns (such as "哈尔滨中央大街", the Central Street of Harbin). All these point to the need of additional novel techniques for effective yet robust place name classification mechanism.

There have been two common practices to tackle this problem:

- The first one is to manually design a rule-based system. However, hand-crafted (HC) models may not be comprehensive enough to capture all the patterns, hence normally leads to lower accuracy.
- The second one is to use machine learning (ML) models [e.g., neural networks, support vector machines (SVMs), and naive Bayes]. Though generally yield better accuracy, these models are typically more complicated and difficult to be interpreted than HC models.

In this work, we propose a novel method that combines the strengths of both ML and HC models. Our model is accurate yet interpretable. We show our methodology in Fig.1. We first manually design a rule-based classification model. We then train an interpretable ML model (e.g., decision tree) and extract the rules and patterns. We upgrade the HC model with the rules extracted from the ML model. We show that, by assisting the HC model design with ML models, our final model yields significant accuracy gain against the original HC model. Moreover, we compare our final model against a wide spectrum of ML models and show that our HC yields similar or higher accuracy while preserving the advantages of simplicity and interpretability.

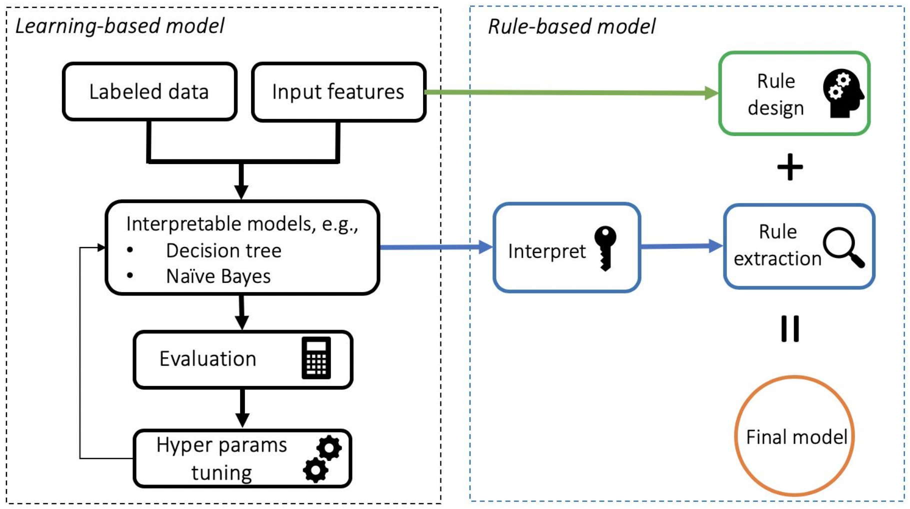

Fig. 1. Steps required to combine the ML model and hand-crafted rule-based model

Methodology

The flow of our methods involved three different parts: feature extraction, ML model training, and refined HC model design. We zoom into each part and show the details as follows.

1. Feature extraction and primitive HC model building

Based on observation, we first built a set of signal-word lists:

a) Administrative units: country(国), province(省), city(市),county(县/乡), town(镇),street(街), road(路);
b) Directions: east(东), south(南), west(西), north(北);
c) Verbs denoting directions: head for(前往/去), come(来), visit(访), locate at(位于/地处);
d) Geographic keywords: mountain(山/岭), sea(海), river(江/河/川), bay(湾);
e) Popular characters used in place names: 华, 州, 城, 阳, 江, 安, 平, 宁, 新, 昌, 丰

Consider a word w_t, we primarily focus on three types of features based on:

- Words prior to w_t
  - We adopt a markovian assumption and consider only the first word w_t-1 prior to the current word. Specifically, we extract the following information:
    - The POS of w_t-1
    - Check if w_t-1 is in the signal-word lists

- Current word w_t:
    - We extract the following information from the current word w_t:
        - Check if w_t is in the signal-word lists
- Words post to to w_t
    - Similar to prior analysis, we extract information for the w_t+1

We then built a primitive HC model based on these features.

2. ML training

Our proposed methodology utilizes a ML model to assist with refined HC model design. In our experiment, we choose the decision tree (DT) model as our ML model. This is because DT have better interpretability than the other ML models such as neural network and RBF SVC. Also, its simplicity enables easier rule extraction. We show the decision path in the DT model as follows:

```
if adm == 0:
 if direction == 0:
  if geo == 0:
   if char == 0: return 1
   if char == 1: return 1
  if geo == 1: return 1
 if direction == 1:
  if prev_tag in ['<START>', ‘ADJ’, ‘FUNCTIONS’, ‘IDIOMS’]: return 1
  if prev_tag == 'ADV': return 0
  if prev_tag == 'N':
   if post_tag == 'ADV': return  0
   if post_tag in ['FUNCTIONS', ‘N’, ‘NS’, ‘NUMBER’, ‘V’]: return 1
  if prev_tag == 'NS':
   if post_tag in ['ADV', ‘ELSE’, ‘N’, ‘NS’, ‘V’]: return 1
   if post_tag == 'FUNCTIONS': return 0
  if prev_tag == 'NUMBER': return 1
  if prev_tag == 'V': return 1
if adm == 1: return 1
```

We next show how we refine our HC model with the rules extracted from the ML model.

3.Refined rule-based model design

As mentioned earlier, we first train the DT model on the training dataset, then extract the rules embedded in the decision tree by exploring the decision path. This is a two-step process. We first prune the decision logic of the DT model to distill the most valuable rules. We do this based on studying each decision path and evaluating its correctness with our prior knowledge. Substantial amount of decision logics were pruned away in this step. For example,

if post_tag == 'NUMBER': return 1

The combination of place name followed by a number is not a common combination, and this logic seems to be an overfitting path. Thus, we prune it away in the rule-distillation process.

Next, we add some manually designed rules to the system. For example, we know that a adv cannot be followed by a place name, thus we add the following rule:

if prev_tag == 'ADV': return 0

After pruning away redundant ML decision logics and adding knowledge-based rules, we obtain our final model as shown in Fig. 2.

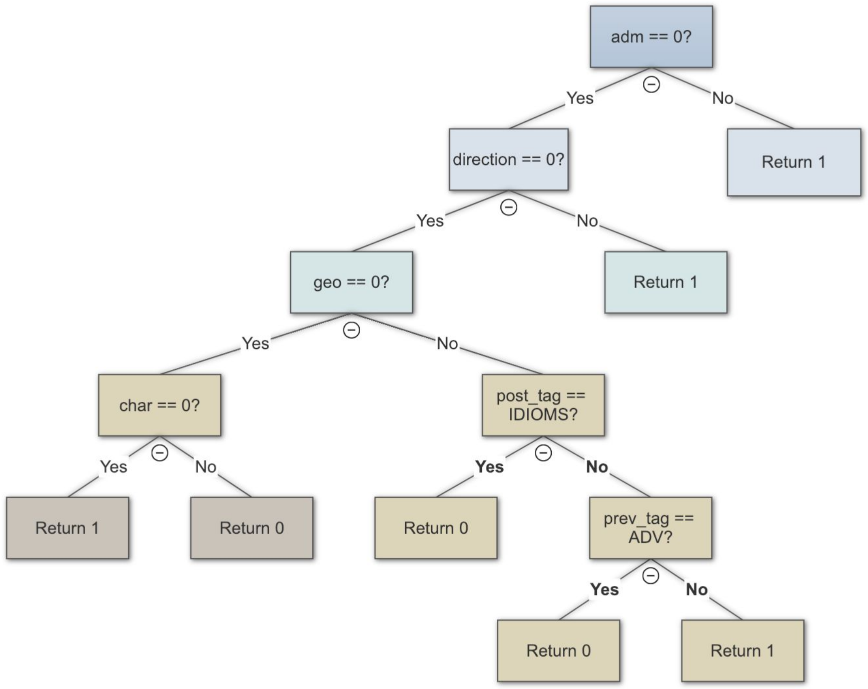

Fig. 2. Refined rule-based model for place name classification

This refined rule-based classifier is much simpler than the original DT. We will later show that the refined rule-based model yields same accuracy and other performance metrics (e.g., recall).

Experiment

In this section, we evaluate our final model against other ML and HC models. We show that our derived model is accurate, simple, and interpretable.

Experiment setup

The data ('taggedwords.txt') used for training is the *People's Daily Corpus(1998)* consisting of articles that had been segmented and POS tagged, with a total number of over six million Chinese characters. For simplicity, we randomly select 10K sentences from the text. We first extract the words with labels and form the instance pool. We randomly select 80% of the instance pool as the training set, and use the rest 20% as the test set.

Imbalanced data handling

Since place names only account for a very small portion(i.e., <1%) of all the words within the text, the overall labelled dataset is very imbalanced. Thus, we sample fewer negative data in the training set. The final ratio of positive and negative data is 2:5 (29% of the whole dataset is positive). Furthermore, we introduce metrics including false-positive (fp), true-negative (tn), false-negative (fn), and true-positive (tp).

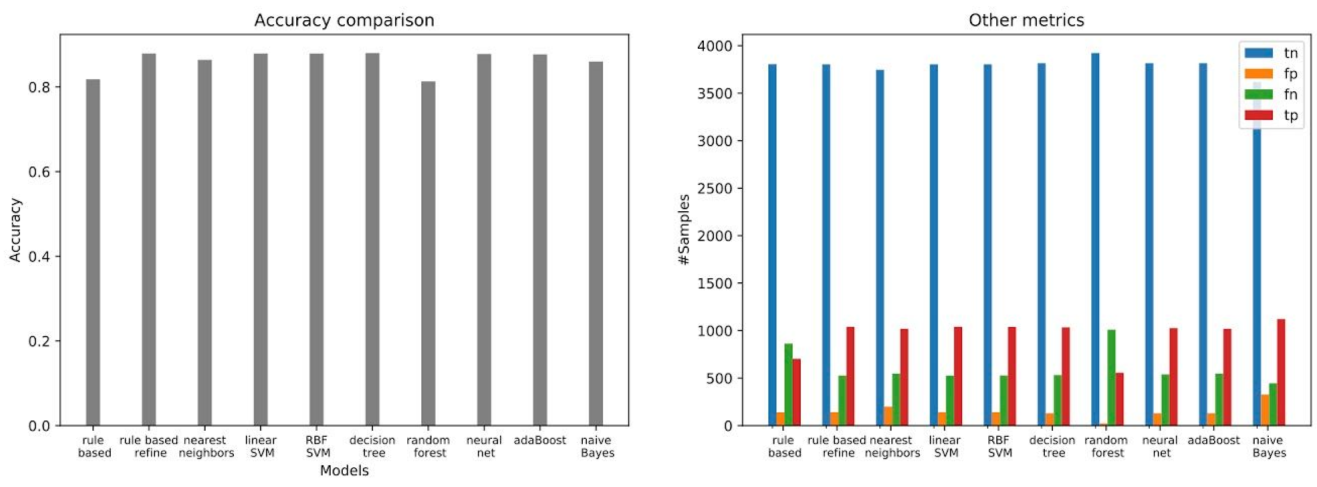

Fig. 3. Comparison among different models.

We compare our final model against the original HC rule-based model as well as eight different widely-used ML models including nearest neighbor, linear SVM, radial basis function SVM, DT, random forest, neural network, AdaBoost, and naive Bayes. We compare the classification accuracy, tn, fp, fn, tp of all the models in Fig. 3. We show that our refined rule-based model outperforms the original rule-based model. The overall accuracy is improved from 81.8% to 87.9%, while the number of recall(TP/(TP+FN)) is increased from 44.9% to 66.3%. Our refined rule-based model actually yields similar or higher accuracy against the ML models, as shown in Fig. 3. Meanwhile, it preserves the advantages of HC rule-based models: simplicity and interpretability.

Conclusion

In this work, we propose a novel approach that leverage both ML and HC models to build a refined rule-based model. We show that the combination of ML and hand-crafted models can be advantageous. Our approach yields similar or higher-level of accuracy compared to ML models. It's also simple and very easy to be interpreted as the conventional rule-based models.

Part IV: Foreign Names

(Code found in foreign_names.py)

Note: this section is further divided into more sections, as indicated by italics and underline.

Background

We also tried to figure out a way to identify foreign names. In Chinese, most foreign names are transliterated using specific characters. Some characters for English names are laid out in Xinhua's *Names of the World's Peoples*, a chart of which can be accessed online at https://en.wikipedia.org/wiki/Transcription_into_Chinese_characters#Transcription_table. Unfortunately, some of these characters are not helpful in determining 'foreignness,' so we removed these characters. The set of transliterated characters used is as follows, which comes from removing some perceived high incidence and unhelpful characters, while adding in a few characters, such as 大 （as in 加拿大 'Canada' or 大卫 'David'). This set is dubbed 'translit_characters':

'布普德特格克夫弗兹茨斯丝什奇赫姆恩尔伊古库胡帕达塔加卡瓦娃法扎察萨莎贾查哈马
玛纳娜拉亚娅瓜夸贝佩泰盖凯韦费泽策塞热谢杰切黑梅内雷蕾耶圭奎惠厄伯珀沃瑟舍哲彻
赫默娜纳勒果阔霍伊比皮迪蒂吉基维威齐西希奇米尼妮利莉里丽伊圭奎惠欧奥博波多托戈
科沃福佐措索若肖乔莫诺洛萝约乌杜图古库武伍富祖楚苏茹舒朱穆努卢鲁尤久丘休缪纽柳
留艾拜派代戴怀宰蔡赛夏柴海迈奈赖鲍保道陶高考藻曹绍'焦豪毛瑙劳尧安班潘丹坦甘坎
万凡赞灿桑詹钱汉曼兰关宽环昂邦庞当唐冈康旺方仓让尚章昌杭芒南朗扬光匡黄本彭登滕
根肯文芬曾岑森任申真'琴亨门嫩伦延昆因英宾平丁廷金温津欣青辛兴钦明宁林琳翁宏蓬
顿敦东通贡孔丰尊宗聪孙松容顺雄准春琼洪蒙农云隆龙律大施卜"

Next we created a set of high-incidence characters; these were characters that we felt were quite rare outside of foreign names. This was as follows, called 'high_incidence_foreign_chars':

'罗埃菲莱巴克'

Next, we isolated the deceptive characters. These were characters that are found in the character table recommended by *Names of the World's Peoples*, but appear too frequently outside of foreign transliterated names to be made use of. These were as follows, called 'deceptive_chars':

'藏日华京'

In these, 藏 is read as 'zang' and surfaces in the proper noun 汉藏 (Sino-Tibetan). This letter codes for syllables in English starting with z or dz and ending in a nasal, which are not numerous. Next is 日, which surfaces in 日本 ('Japan') or just as 日 ('Japan'), coding for ʒ or ʒi: in English, both of which are not numerous. Finally, we isolated 华, which in shorthand refers to China or Chinese culture. Thus, its appearance in native proper names is very common. As a character used in transliteration, it would code a syllable beginning with h^w, which refers to the sound found only in dialectal pronunciations of the initial consonant of 'which' or 'where.' Finally, 京 features most commonly in Chinese local place names, such as 北京 or 南京.

Regular Expression

Next, we used the recommendations found in the *Names of the World's Peoples*. If this is used as a recommendation for transliteration, we figured a good method for determining 'foreignness' of a name would be if all characters found in the proper name also feature in the chart provided in *Names of the World's Peoples*. Thus, we made a regular expression that captures every word that only includes characters from translit_characters. Interestingly enough, full foreign names are often segmented using an interpunct (・ or ·), but of course people can be listed just under first or last names as well. Thus, the full regex for characters found in translit_characters was as follows:

```
([%s]+|[A-Z])([・·]([%s]+|[A-Z]))*\b), (translit_characters, translit_characters)
```

In other words, grab all words that are made up of one or more translit_characters (or one English initial), followed by any number of a sequence of an interpunct followed either one or more translit_characters, or an initial. Eventually, a word boundary will be hit.

We also wanted to factor in the high incidence foreign characters, so we added words that includes at least one of these. Our final regular expression also consisted of any word that contains English letters, i.e. [A-Za-z]. This is represented as follows:

```
r'(([%s]+|[A-Z])([・·]([%s]+|[A-Z]))*\b)|.*[%s].*\b|[A-z]+\b' % \
                    (translit_characters, translit_characters, high_incidence_foreign_chars)
```

In other words, we have the translit_characters regex, as described above, combined with all words that contain a high incidence character, as well as all words made up of at least one English alphabetical character.

<u>*Data Reading and Labeling*</u>

For the data regarding names in this part, we made use of Chinese Treebank data graciously provided by professor Nianwen Xue. These included files with split Chinese sentences tagged with their part of speech. We used data from all files ending in 'bn.pos', or the 'broadcast news data' for training data, which amounted to over 1200 files, as broadcast news would be likely to have a high concentration of proper names. In these files _NR refers to a word tagged as a proper name, a designation which was very important for this project. The method readposfile(file) reads in a pos-tagged file and places all lines containing _NR (or a proper noun) into a list. In make_labeled_data(sentences), this data is split into sentences and smoothed out, then each entry is split on '_' to form a list of tuples containing the word and POS tag. Then each tuple is appended to a new list together with the label. This label is based on the regex we determined above. If the tag is NR and there is a regex match, then l_foreign (short for likely foreign) is used as a label. If the tag is NR and there is no match, then l_Chinese (short for likely Chinese) is used as a label. All other words are labeled as 'Chinese.'

Feature Sets

Of course, when building a classifier, it is necessary to build a set of features. As we wanted to see how much each character found in translit_characters, we used the existence of each character separately as a feature, as we wanted to see which other characters in translit_characters were important for foreign names. We also have crossed off has_NR (proper name tag) as a feature, as it became evident that this would not be helpful for what we did; though it may prove useful for larger-scope experiments. We created the feature sets based on all words tagged NR, that is, proper names, because we wanted to focus on a system that classified names as foreign vs. native/Chinese. The extra data was siphoned off in the method make_proper_noun_labels (labeled_data). The first 100 words tagged NR were test_data, whereas everything else was the training data.

Classifier

Next, we passed the training data to the classifier, and showed the 50 most informative features. The top 10 are shown here.

```
察 = True      l_fore : l_Chin =      219.9 : 1.0
律 = True      l_fore : l_Chin =      180.6 : 1.0
坦 = True      l_fore : l_Chin =      158.0 : 1.0
舍 = True      l_fore : l_Chin =      98.3 : 1.0
勒 = True      l_fore : l_Chin =      61.5 : 1.0
容 = True      l_fore : l_Chin =      51.5 : 1.0
什 = True      l_fore : l_Chin =      48.0 : 1.0
戈 = True      l_fore : l_Chin =      47.9 : 1.0
米 = True      l_fore : l_Chin =      46.4 : 1.0
布 = True      l_fore : l_Chin =      45.4 : 1.0
```

Evaluation

Then we evaluated the classifier on the test data, as shown in the method evaluate_classifier(classifier, test_data). We printed the confusion matrix based on the classification of the test_data versus the labels found in the test_data, and also printed the accuracy. Using the confusion matrix, we achieved precision, recall, specificity, NPV (negative predictive value) and accuracy of predicting likely Chinese versus likely foreign on the test data as shown below, rounded to 3 decimal places.

Precision	Recall	Specificity	NPV	Accuracy
0.856	0.975	0.380	0.800	0.850

Note that the low value here is the specificity. This indicates a problem with the classifier in classifying a number of likely Chinese NRs as likely foreign instead.

Running the Classifier

Unfortunately, the above description says very little about the *actual* accuracy of this classifier, as all the labels were created based on a regular expression. We wanted to run the classifier on data that was tagged by a human as foreign or native, so we tagged all the NRs in 3 files not used in the training (ending in .bc.pos) as either NRnat (native) or NRfor (short for foreign). Native here refers to anything that has a native Chinese reading, so even Korean and Japanese given and place names are included in this categorization. Thus, a better term might be 'Sinitic' NRs instead of native Chinese NRs. NRs tagged as NRfor are those that are transliterations of non-Chinese or non-Sinitic words. Each file had a high incidence of NRs, as shown below.

File Name	chtb_4112.bc.pos	chtb_4113.bc.pos	chtb_4114.bc.pos
NR Count	212	252	407

This results in 871 NRs tagged more specifically.

With help from the method create_evaluate_data, we read the data from each of these files separately, split them into sentences and then tuples based on word and POS tag as we did with the training data. Then we labeled each word: NRnat was labeled '1_Chinese,' NRfor was labeled '1_foreign', and all others were labeled 'Chinese'. We then had a set of data that had been verified by human eyes to truly test the accuracy of our classifier. We next ran the classifier on the NRs found in each bc.pos file, printing the confusion matrix each time. We got the following results, rounded to 3 decimal places.

	Precision	Recall	Specificity	NPV	Accuracy
chtb_4112.bc .pos	0.846	0.993	0.509	0.965	0.863
chtb_4113.bc .pos	0.987	0.959	0.250	0.091	0.948
chtb_4114.bc .pos	0.886	1.000	0.325	1.000	0.892

Our accuracy, precision, and recall were high in all cases. This shows that the regex used to train our classifier was in fact not an abysmal one. However, note the low specificity. This will be something for us to examine if we are to expand on this project. Note also the very low NPV for chtb_4113.bc.pos. This is most likely due to the very low number of NRfor tags in this file, as there are 248 NRnat tags and only 4 NRfor tags.

In the next stages, we might want to isolate more deceptive_chars and add more to the high incidence characters in order to raise the specificity values. However, the classifier has a surprisingly high accuracy for a rule-based training schema.

Please note that a run through of foreign_names.py by a TA using our github repository will most likely not result in the same data found above, as we were only able to include about 150 out of the 1200 files used to train our classify. Thus, all the empirical data will be different - though possibly not much different.

Integration

The above four parts were all integrated using one master file, which we titled master.py. This file first imports each of the four modules that we worked on, then executes the main method of each in the following order: surname_detection, given_name, places, foreign_names, where given_name uses the return value of surname_detection as its argument. When viewing the printed data of master.py, headers with entirely capital letters indicate a boundary: personal names, place names, and foreign names. With the master.py file as a kind of link file, we could all show the work we had done without compromising or cutting out large portions of code.

Conclusions

For foreign_names.py, we built a classifier to classify proper nouns as likely Chinese or likely foreign, but at this moment, it only works for sentences that are already POStagged, and works better for sentences containing further tags of foreign/native tags. We would eventually like for the classifier in foreign_names.py to identify Chinese versus foreign names in free (untokenized) text, but this will require much more work. We hope to keep working on this in order to make these aspects fully implementable.