IFT6285 TP1

Jinfang Luo, Jiechen Wu

March 16, 2018

Abstract

Lemma to Form prediction problem can be seen as a problem of reverse lemmalisation, which is theoretically can't reach the 100% accuracy. The statistical method may be used to get a solution. We will give the result by applying the model MLE, Naive Bayes with the help of nltk package. And we will discuss the feature format and their influence to the performance while using the nltk naive bayes model. We found that how to extract feature is the key factor to improve accuracy for NB.

1 Introduction

We are provided the samples with two lines of words, the words linked to a sentence. one line is the word with the tense, another line is the lemma of the word. A lemma is the canonical form of a set of words. For example, 'found, finds, find' are words with tense of the word 'find' as the lemma.

We should implement a system to predict the pattern sequences by dealing with lemma sequences. We may call this issue as 'lemma-form' mapping problem. We are not just focusing on how to find a solution on solving the mapping issue. What we would like to know is how the models work, and what's the limit to them.

2 Analyze the data

2.1 Description

The data in three sets are contains of English language, they are extracted from wikipedia and is encoded in utf-8. It comes from totally 2020 slices of 1000 articles. The article is represented by a succession of lines, and a line represents two terms delimited by <tab> presenting the format (form,lemma), in which the lemma corresponds to form, each sentence is marked with an empty line.

2.2 The relation between Training set, Dev-Test Set and Test Set

Training set is a set of examples used for learning, that is to fit the parameters of the classifier. Dev-Test Set is a set of examples used to tune the parameters of a classifier Test set is used only to assess the performance of a fully-specified classifier.(Ripley, 2007)

We train our model in dev-test set, there is another advantage for this, is the cross validation for better training our model. During our development, we use a part of dev data, and split the data by 9:1 for assigning to training set and test set. After having our classifier model, we use it to predict the form with lemma of the word and then compare with the data in test set.

2.3 Rules based method or Probability based method

There are two methods to process language problem. One is building the system by rules, another is based on statistic.

If we implement our system based on rules. First step is performing lexical analyst, then parser the sentence and analyst syntax, after execute semantic parsing, the last is context analyst. So if we would like to predict the form, the processing is inverse.

For example, we need to predict form of the word 'become' in our sentence which contains by lemmas: "a standardized version of the game eventually become know as bagatelle." First we need to know the grammatical tense according to the context, if it's past tense, then we analyst semantic meaning, then parser and perform lexical analysis, at last we should know the form of the word 'become' is mapping to 'became'.

Another method is by mathematic statistical processing. We build the mathematic model to learn the feature or rule for the problem. By using statistic method, we calculate the probability of the form by the model, then we choose the highest probability as the predict result.

Not so much similar work can be refereed on this lemma-form prediction problem. We choose the statistical method. Because rules based method will ask for the collection of the rules. It's obviously not the purpose of this project.

2.4 Reduce the noise

The punctuations in the raw data can be seen as the noise if we apply the POS tagging. But consider about the prediction may based on the appearance of the certain punctuations, For example, the word of the beginning of the sentence will be with the first letter in capital. The word follows a comma is usually in lowercase, et. We decide to keep these information and don't treat them as the noise. Non-UTF8 characters won't bother us if we read the file in "latin1". And we will discuss the possible reason that cause this problem and our solution to get rid of them in related work section.

3 Tools

3.1 Open source tools

There are some open source tools for handling Natural Language process, we compared some of them and then we choose to use NLTK in Python because of the program language and the powerful modules it contains.

We list some features of these open source tools for the future needs in case:
Stanford CoreNLP: linguistic analysis tools, Java
TextBlob: Word inflection and lemmatization, Java
NLTK: Naive Bayes, HMM, N-gram Tagger, Python
SpaCy: 13 statistical models, deep learning, Python
Apache Lucene: work with finite state and get free search engine, Java

3.2 NLTK Naive Bayes Related

1. nltk.ConditionalFreqDist - Mainly used to calculate the conditional frequence distribution. Can be taken as dict which each value is the FreqDist for each key. Take tuple list as parameter.

2. nltk.NaiveBayesClassifier - The nltk Naive Bayes Classifier. The train() method of this class take the train data and return a classifier object, which can be use to invoke classify() to label the data. train() take labeled feature tuple list and classify() take the feature list.

3. nltk.FreqDist - Derived from high performance Counter object in python. Calculate the frequency for each item.

3.3 System Environment

{"python 3.6" , "nltk 3.2.5" , "macOS version 10.13.2" , "cpu 2.6GHz Inter Core i5" , "Memory 8GB 1600 MHZ DDR3"}

4 Fondation of Algorithm

Consider the prediction for single word a time, lemma can be treated as the input, and the form as label.

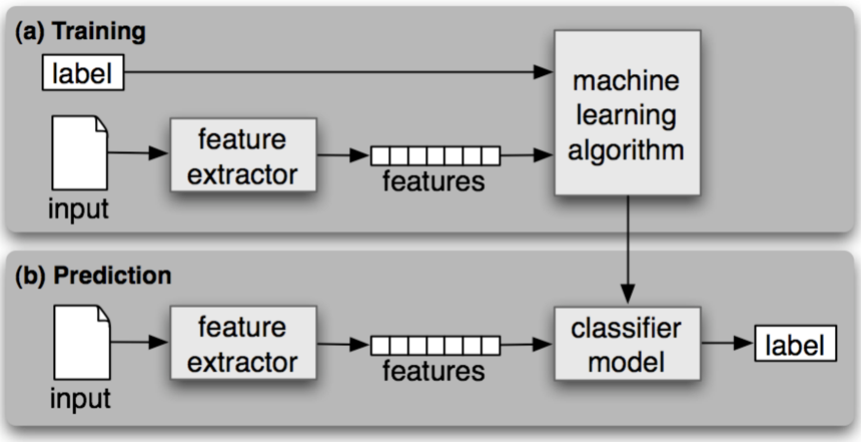

Figure 1: Supervised Classification[1]
(Bird, Klein, & Loper, 2009)

4.1 Maximum Likelihood Estimate and Bayes Theorem

Express our idea using Bayesian probability terminology:

$$posterior \propto prior \times likelihood$$

When we compare the probability of different form for lemma l_i, we are comparing the $p(f_i|l_i)$, posteriors of lemma l_i predicted as form f_i. It depends on the prior of the form f_i and the likelihood of the lemma l_i when form f_i shows.

$$\frac{p(f_i|l_i)}{p(f_j|l_i)} = \frac{p(f_i) \cdot p(l_i \mid f_i)}{p(f_j) \cdot p(l_i \mid f_j)} \qquad f_i, f_j \in Forms(l_i) \tag{1}$$

And $p(f_i) \cdot p(l_i \mid f_i)$ can even be simplified if we apply the MLE estimator $\frac{f}{n}$ to it.

$$\begin{aligned} \hat{p}(f_i) \cdot \hat{p}(l_i \mid f_i) &= \frac{freq(f_i)}{n} \cdot \frac{freq(l_i, f_i)}{freq(f_i)} \\ &= \frac{freq(l_i, f_i)}{n} \end{aligned} \tag{2}$$

The equation 2 is a simple theoretical model. It maybe a little astonishing that,based on this model, if we want to compare $p(f_i|l_i)$ and $p(f_j|l_i)$, just need to compare the co-occurrence $freq(l_i, f_i)$ and $freq(l_i, f_j)$. The problem is we can't deal with those lemmas we haven't seen. Since we use NLTK package, the smooth method of NLTK will be given later on.

The result of unigram MLE will be used as our baseline.

4.2 Naive Bayes

Naive Bayes is a classification technique based on Bayes' theorem with independence assumption. given a problem instance to be classified, represented by a vector $\mathbf{x} = (x_1, \ldots, x_n)$ representing some n features (independent variables), it assigns to this instance probabilities for each of K possible outcomes or classes C_k. (Murty & Devi, 2011)

$$p(C_k \mid x_1, \ldots, x_n) = \frac{1}{Z} p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)$$

With the the "naive" conditional independence assume that each feature x_i is conditionally independent of every other feature x_j for $j \neq i$, given the category.

The prediction lemma-form problem, For Any given lemma l_i we suppose that there is a form list $Forms(l_i) = \{f_{i1}, f_{i2}, ..., f_{ik}\}$ which contains all its possible forms. The lemmas sequence $(l_1, l_2, ..., l_n)$ are the sequence we can observe, so we take them as features. Then the form sequence is the classes that we want to put the features in[2]. Actually,in this way, we built up a label list contains all the forms show up in the corpus $\{Forms(l_1), Forms(l_2), ..., Forms(l_n)\}$, the size of this list will be V, which is the total vocabulary of the forms in the corpus. Unfortunately, due to the immense of the vocabulary, the conditional probabilities table will be infeasible.

$$p(V|l_1, ..., l_n) = \frac{1}{Z} p(V) \prod_{i=1}^{n} p(l_i \mid V)$$

Actually when we do predicting for the lemma l_i in a sequence $(l_1, l_2, ..., l_n)$, we just need to look at $Forms(l_i)$ and don't have to take account the $Forms(l_j)$ $(j \neq i)$. In this section, We see the prediction of form for each lemma $l_i \Longrightarrow f_i$ as a classification by choosing the labels only from $Forms(l_i)$. It's reasonable since the lemma is kind of the product of the lemmalization of the correspond forms $\{f_{i1}, f_{i2}, ..., f_{ik}\}$. The one feature NB model correspond to our problem is like:

$$p(f_i|l_i) \propto p(f_i) \cdot p(l_i|f_i)$$

And since one time training will prepare the parameters for the prediction for one lemma, if the train set contains n different lemmas, then the model will be trained n times. Nevertheless the same lemma will share the same parameters. So we use a dictionary to store them to avoid the the repetition work.

The bigram model we built is taking account the influence of the l_{i-1} to f_i.

$$\begin{aligned} p(f_i|l_{i-1}, l_i) &\propto p(f_i) \cdot p(l_{i-1}, l_i|f_i) && (3) \\ &= p(f_i) \cdot p(l_{i-1}|f_i) \cdot p(l_i|l_{i-1}, f_i) && (4) \\ &= p(f_i) \cdot p(l_{i-1}|f_i) \cdot p(l_i|f_i) && (5) \end{aligned}$$

Apply the Naive Bayes assumption to get from equation 4 to 5.

And in our case, if we see "of find" to the probability of classifying "find" into "finding" it would be:

$$p(finding|of, find) \propto p(finding) \cdot p(of|finding) \cdot p(find|finding)) \qquad (6)$$

$form_i = "finding" \quad lemma_{i-1} = "of" \quad lemma_i = "find"$

4.3 Practical Work with NB

Take the lemma "find" in file <dev-24> as an example(The first 90% of the file is taken as the train set N=1576467). Table 1 shows the conditional frequency of the forms of "find" in the train set.

form f_i	found	find	finds	finding	Finding	Finds
freq(f_i,"find")	778	204	65	**32**	4	2
freq(f_i)	778	204	65	**45**	4	2

Table 1: conditional frequency of the forms of "find"

Although we don't even need to use the $freq(f_i)$ in our current model, it's for the future use.

Suppose our one feature NB model is classifying the lemma "find", the only feature is {"lemma":"find"} and the classes to be chosen from are {found, find, finds, finding, Finding, Finds}. The model will compare the max $p(f_i|l_i)$ and do the classification. No doubt, $p("found"|lemma = "find")$ is the max, thus the classes will always be form "found". We call this unigram. Because this model just has one feature and only the l_i will affect the classes f_i.

4.3.1 Smooth method of NLTK Naive Bayes

According to the source code of NLTK class ELEProbDist which is the feature distribution function of NaiveBayesClassifier, it takes Expected Likelihood Estimate method, the probability distribution is given as below:(Loper, n.d.-b)

$$p(x_i) = \frac{C + 0.5}{N + 0.5B} \qquad X = \{x_1, x_2, \ldots, x_o, \ldots\} \qquad (7)$$

C is the frequency we count with N outcomes and B the number of the observed value of X.

[2]We will probably use both "label" or "class","tag" or "classify" in this notes, they will mean same thing.

4.3.2 Training set

We didn't notice this problem until we looked at the table 1. The $freq(''finding'') = 45$ is slightly bigger that the $freq(''finding'', ''find'') = 32$. That means the form "finding" correspond to other lemma other than "find", for example, lemma "finding". Then question is when we train the classifier for lemma "find", whether should we put the ("finding","finding") in the train data? Or we can put this question in another way, does ("finding","finding") show or not in train data affect the training result of the model?

We observe train data [("find","finds"),("find","finding"),("finding","finding")][3].

```
m1 = [({'lemma':'find'},'finds'),({'lemma':'find'},'finding')]
m2 =
    [({'lemma':'find'},'finds'),({'lemma':'find'},'finding'),({'lemma':'finding'},'finding')]
b1 = [({'find':True},'finds'),({'find':True},'finding')]
b2 = [({'find':True},'finds'),({'find':True},'finding'),({'finding':True},'finding')]
```

$$
\begin{aligned}
p^{*}_{m1}('lemma':'find'|'finding') &= (1+0.5)/(1+1/2) = 1 \\
p_{m2}('lemma':'find'|'finding') &= (1+0.5)/(2+2/2) = 0.5 \\
p_{b1}('find':True|'finding') &= (1+0.5)/(1+1/2) = 1 \\
p_{b2}('find':True|'finding') &= (1+0.5)/(2+2/2) = 0.5
\end{aligned}
$$

*These values can be check by the _feature_probdist method of a classifier of class NaiveBayesClassifier. For example:

```
classifier_m._feature_probdist[('finding', 'lemma')].prob('find')
classifier_b._feature_probdist[('finding', 'find')].prob(True)
```

For p_{m1}, C equals to the count of feature 'lemma'='find' and classifier form 'finding' show up, which is 1 in this case. And B is the number of the observed value of the feature name 'lemma', we just can see 'lemma'='find', so it's 1. But after we add the ("finding","finding"), which corresponds to the feature. For p_{m2}, C keeps unchanged. Though it has two possible values 'find' and 'finding' for feature 'lemma' now. So B=2.

Same situation for p_{b1} and p_{b2}. But we should pay attention that in NLTK NaiveBayes model, it will complete the known features for every element with a None value if this feature doesn't show up in the current item. That means the td_b2 will be completed as below:

```
b2 =
[({'find':True,'finding':None},'finds'),
({'find':True,'finding':None},'finding'),
({'finding':True,'find':None},'finding')]
```

It shows us adding ("finding","finding") does matter the probability. Compare the train data set (2) to (1) for l_i. So those pairs which contain the forms of the current objective lemma added to the train set.

$$
\begin{aligned}
&train\_data\_set_{l_i}(1) = pairs_{l_i} = [(l_i, f_{i1}) * freq(l_i, f_{i1}), \ldots, (l_i, f_{ik}) * freq(l_i, f_{ik})] \\
&train\_data\_set_{l_i}(2) = \sum_j pairs_{f_{ij}} \\
&pairs_{f_{ij}} = [(l_{j1}, f_{ij}) * freq(l_{j1}, f_{ij}), \ldots, (l_{jm}, f_{ij}) * freq(l_{jm}, f_{ij})] \\
&f_{i1}, \ldots, f_{ik} \in Forms_{l_i} \\
&l_{j1}, \ldots, l_{jm} \in Lemmas_{f_{ij}}
\end{aligned}
$$

Add the ("finding","finding") to the train data, to decrease the p(find|finding) it's like to tell the model that, if there are more form "finding" correspond to lemma "finding", then probably few correspond to lemma "find".

[3] in the format (lemma, form)

4.3.3 Feature format

And take a look of this example below, it will show us the difference between the muti-value format feature and bi-value format feature.

```
m3 = [({'lemma':'find'},'finds'),({'lemma':'find'},'finding'),
    ({'lemma':'finding'},'finding'),({'lemma':'pseudo-f'},'finding')]
b3 = [({'find':True},'finds'),({'find':True},'finding'),
({'finding':True},'finding'),({'pseudo-f':True},'finding')]
```

$$p_{m3} = (1 + 0.5)/(3 + 3/2) = 0.43$$
$$p_{b3} = (1 + 0.5)/(3 + 2/2) = 0.5$$

We set a pseudo lemma 'pseudo-f' for 'finding', we add the ('pseudo-f','finding'), the probability p(find|finding) dropped more quickly for the multi-value format feature. For the bi-value format feature, the B value remains 2. But for the multi-value format feature, the B value will grow as we add new lemmas, which will leave more possibility for the unseen situations. This difference won't be very huge for unigram, since most forms just have not more than 2 corresponded lemmas. Actually that's also why we have to set a pseudo lemma for "finding". But if we start to use bigram in the multi-value format feature, this difference is much bigger.

Bigram

Before we implement the equation 6 for bigram, we face the problem then, how can we prepare the feature set to feed the NLTK Naive Bayes model? The feature is {"lemma-1":"to","lemma":"find" } or {"bigram":("to","find"),"lemma":"find" }? Based on equation 6. We have to chose {"lemma-1":"to","lemma":"find" }. But how about {"to":True,"find":True }?

Using equation 6, We calculate the p(find|finding) for both multi-value and bi-value form feature for the train data in <dev-24> and listed in 1:

The right part of equation 6 can be presented in 4 different forms in both multi-value and bi-value format feature.

$$(1) = p(finding) \cdot p(''lemma - 1'' : ''of''|'finding') \cdot p(''lemma'' = ''find''|'finding')$$
$$(2) = p(finding) \cdot p(''of'' : True|'finding') \cdot p(''find'' = True|'finding')$$

$$p(''lemma'' = ''find''|'finding') = (32 + 0.5)/(45 + 2/2) = 0.707$$
$$p(''find'' = True|'finding') = (32 + 0.5)/(45 + 2/2) = 0.707$$
$$p(''lemma - 1'' : ''of''|'finding') = (2 + 0.5)/(45 + 316/2) = 0.012$$
$$p(''of'' : True|'finding') = (2 + 0.5)/(45 + 2/2) = 0.054$$

The "of" shows 2 times in the bigrams of lemma for form finding. The types of the values in the name of "lemma-1" is 316. And it makes the p("lemma-1":"of"|'finding') value quite small. And of course, we don't know the difference of the classification result yet. Because when you compare the p('finding'|"lemma-1":"of","lemma":"find") and p('finds'|"lemma-1":"of","lemma":"find"), the latter will take exactly the same B value, then it reduces the influence of the ratio.

The result proves result by using the multi-value format feature and bi-value format feature is very close.

File	Baseline	Bigram multi-value	Bigram bi-value
dev-24	78.09%	84.38%	84.37%
dev-24...241	79.55%	86.97%	86.91%

Table 2: Bigram accuracy of multi-value and bi-value format feature

- dev-24 (1 file) (1578267 itmes for training, and 175363 items for test)

- dev-24...241 (10 files): dev-24,dev-25,dev-26,dev-27,dev-28,dev-237,dev-238,dev-239,dev-240,dev-241 (12176586 itmes for training, and 1352955 items for test)

And for the unigram, the model performs the same with two kinds of format feature on dev-24, it gives 78.07%.

4.3.4 Over-fitting?

Originally for Naive Bayes, one of its advantage is that it's so simple, so few parameters to be trained. This fact makes it won't be trapped by the local maximum. But since in our model, we train the model separately for each word. Let the model greedily search for the max probability locally rather than locally. We have the reason to believe that this problem exists in our problem, but we haven't prove it.

5 Related work

Lemma == Form

The prediction of whether Lemma & Form are identical. It's a simplified problem we tried to resolve. By using it test our models and ideas, we get a very short training and testing time and the performance of the result can also be a reference for the solution to the original problem. Because the performance of the original problem can never exceed the result of this simplified problem.

	unigram	bigram	trigram
dev-24	84.12%	89.53%	87,93%

Table 3: Lemma == Form prediction

POS

We believe that Part of Speech tagging will help to eliminate the ambiguity of the lemma, e.g. play as verb or as noun. Result shows it will slightly increase the accuracy but not more than 0.4%, which is a little lower than we expected. The possible reason for this is: firstly, the lemmas who have the ambiguity between different kinds of POS tagging are not so many. Second, apply the real text trained POS tagging to lemma, it gives a relative lower performance also.

HMM

Our first idea is to apply HMM to this problem. The output observation is the lemma sequence and the form sequence are hidden states. But after the trial of the HMM tagger in NLTK, and we found it's incredibly slow. A training model with only 2000 sentences predict incorrectly 5 words in about 5 minutes.

Non-utf8

The appearance of the Non-utf8 will cause the code can't succeed to decode the corpus text in encoding "utf8". We found most of the non-utf8 characters show in lemmas and the non-utf8 characters byte minus the byte at correspond position in form equals 0x20. It's the value we usually use to get a uppercase letter to lowercase. So we guess the problem is caused by the program lemmalisation lowercased incorrectly some utf-8 letters. If the encoding "latin1" is used for both read and write the corpus file. The problem is ignored. Otherwise, if the encoding utf-8 is used to read the file. Since the non-utf8 will cause the error of the decoder, two situation will happen to the non-utf8 characters: ignore or replace. And both situation will cause the mismatching of the lemma and form. It will slight affect the accuracy of the prediction. Since we haven't done the complete statistics for these non-utf8 characters, the reason caused this problem remains a guess. But if the related work would like to be done by someone, We suggest the exception tackle mechanism in python to fix or count these non-utf8 characters. It seems very practical.

6 Discussion

Smooth for unseen

And if the bigram model encounter a unseen bigram pair, which means it will simply delete the new unseen feature.(Loper, n.d.-a) Thus for that pair, only the original lemma feature provide the information, in this situation, it's a unigram model. If the model encounter a unseen lemma, we predict a identical form for it. That's out of the nltk Naive Bayes.

Limit

Due to the Naive Bayes Assumption, the features are independent. Then we ignore the influence between the lemmas. But it's not realistic.

Additionally, since we classify the lemma one by one. In our model, we assume that for two lemmas l_i and l_j, the prediction form f_i is independent of every other f_j for $j \neq i$. Obviously, it's not realistic. For example, the lemma series "student be", if the "student" is predicted as "students", then the "be" should be either "are" or "were".

Expected Likelihood Estimate

The Naive Bayes Model is not trained in NLTK. But we have the reason to believe the ELE model or ELE technique is chosen specially for the classification task. The ELE model is the additive smoothing with the lambda = 0.5, comparing to Laplace estimation which is with the lambda = 1. But we are still doubt that whether it's the best distribution for our case. More work should be done here.

7 Conclusion

Naive Bayes classifier in NLTK is capable of resolving the lemma-form prediction problem and gives us a good performance compare to the baseline. But its relatively fixed interface makes the user not easy to adjust the parameter to achieve a better performance.

References

Bird, S., Klein, E., & Loper, E. (2009). *Natural Language Processing with Python.*

Loper, E. (n.d.-a). *nltk.classify.naivebayes — NLTK 3.2.5 documentation.* Retrieved 2018-03-16, from http://www.nltk.org/_modules/nltk/classify/naivebayes.html

Loper, E. (n.d.-b). *nltk.probability — NLTK 3.2.5 documentation.* Retrieved 2018-03-16, from http://www.nltk.org/_modules/nltk/probability.html

Murty, M. N., & Devi, V. S. (2011). *Pattern Recognition: An Algorithmic Approach.* London: Springer-Verlag. Retrieved 2018-03-11, from //www.springer.com/gp/book/9780857294944

Ripley, B. (2007). *Pattern Recognition and Neural Networks.* Cambridge University Press. Retrieved from https://books.google.ca/books?id=m12UR8QmLqoC