Language Identification

Shangyu Chen
405B 640s Swanston Street Carlton VIC
shangyucc@icloud.com

Abstract

Language identification is the identifying the language of a given document. This paper is a reflection of experiments of training a dataset with more than thirty thousand documents with given languages and test results.

1 Introduction & Datasets

Our training dataset has 20 languages and some unknown language instance:

ID	Language
ar	Arabic
bg	Bulgarian
de	German
en	English
es	Spanish
fa	Persian
fr	French
he	Hebrew
hi	Hindi
it	Italian
ja	Japanese
ko	Korean
mr	Marathi
ne	Nepali
nl	Dutch
ru	Russian
th	Thai
uk	Ukrainian
ur	Urdu
zh	Chinese
unk	"Unknown"

"train.json" which has 37022 lines has some lines like this:
""""

{"text": "\n Causa C-110/04 P: Ordinanza della Corte 30 marzo 2006 \u2014 Strintzis Lines Shipping SA/Commissione delle Comunit\u00e0 europee (Ricorso contro una pronuncia del Tribunale di primo grado \u2014 Art.\u00a085, n. \u00a01, del Trattato CE
………………………………
""""

It would be use to train the prediction model.
The file "dev.json" would be used to test prediction accuracy.

2 Background Research

Considering human's behaviours in language identification.
It is easy for human to identify Arabic, Chinese and English because they have absolutely different characters and shapes. Sometime it is more easy for machine, 'خ', '亡', 'i' have similar shapes but there utf-8 codes are different.
Assume now, machine know that this report is a English report. But 'خ', '亡' are found in this report, it is reasonable to guess 'خ', '亡' are English characters. Obviously 'خ', '亡' are not English characters. Humans get the idea that probability P(it is a English article | '亡' appears) is pretty low from they find the P('亡' appears | it is a English article) is pretty low.
It is from Bayesian theory:

$$P(B_k|A) = \frac{P(B_k) \times P(A|B_k)}{\sum_{i=1}^{m} P(B_i)P(A|B_i)}$$

For some languages which are really similar, such as Italian and English, in one character level they are really similar. The only difference is that Italian has 'é'. The probability of occurrence of 'é' is not so high in Italian, and 'é' can also appear in English text. Therefore it is not easy to distinguish English and Italian. Some other languages from a same language group are more difficult to distinguish such as Bulgarian and Russian.
But the basic idea of characters frequency are still effective.

3 Document Representation

We should do document cleaning before we start training. Some thing like punctuations, web links are not expected.
(Some punctuations might be useful such as in Italian, they write 10, 000, 000. 28 as 10. 000. 000, 28, but in most cases they cannot decrease information entropy.)
(Web links should be deleted because they always use Latin characters which will influence training result badly.)
Function text_filter(strlang) can do these things.
As mentions, for some languages, probability of occurrence tables of character tables are not enough to identify the language. It is better to train somethings like affixes or words. Nevertheless, if calculate every words or affixes, the dimension (number of attributes) of training model would be really high.
"N-gram" is the method to solve this problem.
Combination of two characters in different languages from same language groups always have different probabilities of occurrence.
For example, Italian words always have vowels at the end of the words. It means the combination of two characters with a vowel is the second one would be higher in Italian than in English.
This is a sample of bagging in machine learning.
In python's sklearn.feature_extraction.text, there is a class TfidfVectorizer can do this "n-gram".
After testing, combination two characters performs as well as combination of three characters. Therefore, n-gram range is set as 1~2.

4 Models

In my project, I use two systems of Logistic Regression and Support Vector Machines algorithms.
This two algorithms have better accuracies than other algorithms.

4.1 Other Models

4.1.1 Naive Bayes

Bayes theorem's idea is hidden behind all machine learning model but Naive Bayes classifier is hard to implement in this n-gram model.
In Naive Bayes, we predict class as:

$$c = argmax_{c_j \in C} P(c_j) \prod_{i}^{m} P(x_i \mid c_j)$$

Here, the x_i is a character or a character combination's â = frequency/total frequency of all characters and character combinations. While machine is predicting, it is very common that one character combination is not appeared in one text, but it high â in this language.
Naive Bayes can be used in language identification. However it is not suitable to use it after do "n-gram".
Before manipulate "unknown" instances, its accuracy is below 60%.

4.1.2 Voting

Voting is not necessary after use some different classifiers.
Experiences show that When Linear Regression and SVM make errors, GaussianNB, MultinomialNB, DecisionTreeClassifier or KNeighborsClassifier are also making mistakes.
For example, in many instance of Russian, Linear Regression predict them as Bulgarian, and their "predict_proba"s of Russian are not high.
This means, in this 1~2 n-gram level. The result is cluster to Bulgarian's elements really. More training datas are required, or word level trainings are required for correct prediction.

4.1.2 Decision Tree

Decision Tree is not really use for for numeric attributes. Discretisation is required.
Then Logistic regression seems understandable here.
Before manipulate "unknown" instances, its accuracy is below 60%.

4.2 Logistic Regression

Logistic function use its steep math character to make continuous value easy to be classified.
In this case, generalised logistic function with J dimension is required. (J > 2)

$$P(y{=}j \mid x_j \beta) = \frac{exp\left(\beta_j \cdot x\right)}{\sum_{k=1}^{J} exp\left(\beta_k \cdot x\right)}$$

Then use maximum likelihood estimation:

$$argmax_{\beta} \prod_{i=1}^{|} P(y_i \mid x_i \beta)$$

Python's sklearn.linear_model's LogisticRegression class can do all this things.
Before manipulate "unknown" instances, its accuracy can be higher than 75%.

4.3 SVM

SVM is from linear statistical model mathematically. It is not easy to find its physical meanings. Mathematically, it must find maximal likelihood and unbiased parameter.
I just use linear regression and linear kernel then I got better accuracy than logistic regression.

4.4 Model Validation

SVM is easy to be over-fitting because it is hard to interpret its physical meanings.
After solving "unkown" instance, SVM's accuracy and Logistic Regression's accuracy on any test dataset will between 0.85 and 0.9.
However, SVM's accuracy on training dataset "train.json" is as high as 0.990519 while Logistic Regression's is 0.955513.
It can be find that both systems are overfitting a little bit. But SVM's training accuracy is closed to 100%, this meanings SVM overfitting more.

4.5 Threshold to model "Unknown" documents

Two methods in python's sklearn are used to set threshold: decision_function and predict_proba.
Systems would be unconfident to predict languages in two case.
First case, the document seems like different language, systems may think that this document is highly like a Russian document, but is also highly like Bulgarian document. There is no doubt that it must be Russian or Bulgarian but not third unknown language. Systems should choose Russian or Bulgarian but not return "Unknown". Humans will say "I do not known" only when they never see this characters before. It is same for computers.
Therefore, the second case is, system snever saw this language before or they saw this language before but they never know which language is this before. However, system still have a probability list for attributes in this document.
To solve this problem, decision_function can be used. I do some supervised machine learning and use some hyper parameters to solve this problem.
The decision_function reflects model's confidences of predict this instance as a particular language.
A element in a predict_proba is that system thinks that this instance has this probability that it is this language.
Therefore, system always predicts the language have high predict_proba or high decision_function.
For instance in first unconfident case, it will have some languages have relatively high predict_proba or decision_function compared to other languages.
There are three parameters I used to determine threshold from predict_proba and decision_function:

1) *maximum of predict_proba*: If maximum of predict_proba is too small, it means the system is not confident to return this language with maximum predict_proba as prediction.
2) *entropy of predict_proba*: If system determine the predict_proba of this language is 100%, then the entropy would be 0. If it cannot determine between two languages, the entropy would be higher; if three languages, higher and higher. If every language has similar predict_proba, the entropy will pretty high, the system might never saw this language before.
3) *variance of decision_function*: Variance can also reflect data's divergence level. If decision_function for different languages has low variance, they are similar. Therefore, low variance means the system might never saw this language before.

4.5.1 Threshold for Logistic Regression

By test 3 parameters and observation on datas, for the logistic regression model, return "Unkwown" when the *maximum of predict_proba* is less then 0.5 and the *entropy of predict_proba* is greater than 2 will increase accuracy most efficiently.

4.5.2 Threshold for SVM

By test 3 parameters and observation on datas, for the SVM, return "Unkwown" when the *variance of decision_function* is less then 0.1 can increase accuracy efficiently.

5 Explanation of functions in codes

5.1 How to use Logistic Regression

Train:
>maketrainsample(train file’s name)
>datatrain_LogisticRegression()
Predict:
>predict_LogisticRegression(predict file’s name)
Use test file to test accuracy:
>predict_LogisticRegression_test(test file’s name)
#If not type two train functions, it also works, it will used trained model in the file ‘datatrain_LogisticRegression.pkl' to predict.

5.2 How to use SVM

Train:
>maketrainsample(train file’s name)
>datatrain_LinearSVC()
Predict:
>predict_LinearSVC(predict file’s name)
Use test file to test accuracy:
>predict_LinearSVC_test(test file’s name)
#If do not type two train functions, it also works, it will used trained model in the file ‘datatrain_LinearSVC.pkl’ to predict.

5.2 Other functions:

>entropy(somearray)
#somearray is a list are array of probabilities which have sum 1, it will return these probability distribution’s entropy
>text_filter(strlang)
#strlang is a text, it will the text without URL and punctuations.
>getfile(filename)
#return a list which elements are lines of the file
>char_fre_train_data(filename)
#return the data could be used for training from the file
>maketrainsample(filename)
#saving the result of char_fre_train_data(filename) to file ‘traindata.pkl'
>pkloutput(filename)
#load the pkl file to use python data in it