COMP90049 Knowledge Technologies
Semester 2, 2016
Project 2: Geolocation of Tweets with Machine Learning

1 Introduction

This project aims at building, comparing and evaluating geolocation classifier for twitter users based on the content of twitter. It is implemented in two main phases: feature engineering and building classifier using machine learning methods. The features are generated by approximate match algorithms: local edit distance(LED) and global edit distance(GED), then combined findings together with the “best 35” terms generated by mutual information(MI) method. The best solution used 44 features including tweet id, user id and location. Three classifiers have been attempt: Zero-R, Naïve Bayes, and 1-Nearest Neighbour. If consider both accuracy and time, Naïve Bayes won the battle.

2 Methodology

2.1 Overview

There are five city classes in this project: Boston (B), Houston (H), San Diego (SD), Seattle (Se), and Washington DC (W). There are two sets of software implemented in Python, one online converter and a machine learning software are being used to do classifying:

a) Feature engineering: Approximate match systems (LED and GED)
b) Value generating: Tweets reader
c) .CSV to .Arff converter: ikuz.eu/csv2arff
d) Classifier building: Weka machine learning software

The inputs for feature engineering systems are the training tweets file and the city name file. It is a step-by-step process to use these four systems.

First, two approximate match systems would find all the approximately matched with the 5 city names in the training tweets; then processing the training tweets, development tweets and test tweets using 9 approximate matched words combine with other attributes and mark the frequency of the attributes in a CSV file; then convert the CSV file into Arff so that Weka could read it and build classifier from it.

2.2 Pre-processing

The pre-processing for data sets including remove all the digital characters in tweets and create a file for the five city names with one city per line. For GED and LED, the pre-processing for city names are the same. Beside the 3 single word city name (Boston, Houston, Seattle), the 2 two-word names (San Diego, Washington dc) and all the 4 tokens from the multi words names (san, diego, Washington, dc) are included to be approximately matched from the tweets. And tweets are being pre-processed differently for GED and LED to handle: they are tokenised in GED while not in LED.

2.3 Feature engineering: approximate match

There are different ways to find proper features to identify the current city of the tweet user including token-level features, sequence-level or semantic generalisations (Scott & Matwin, 1999). In tweets, approximate matched words would be the type of features that may easily ignored because they are not going to happen a lot of times. But they still usually exist in tweets, as tweets are under an informal social media context. So two approximate match methods are being used to find misspelling city names from the training tweets. As GED is suitable for matching content in similar length, the tweets and city names are both pre-processed into tokens. As for LED, each line in tweets will be searched to find city name tokens. The misspelling city names found are:

City Name	Misspelling name	Method
San Diego	San Dieg (san diegm/ san diegan / san dieg)	GED/LED
San Diego	Sun Diego	LED
Seattle	Seatle	GED
Seattle	Seattl (Seattl / Seattlites)	LED
Washington DC	Washingtion	GED/LED
Washington DC	Washingston	LED
Washington DC	Washingtondc	GED/LED

Table 1- Misspelling city names

Using these 9 misspelling city names as attributes, it could generate an "Approx17" feature set which combined with 9 origin city names and an "AddApprox44" feature set which combined with "best 35" after removing features duplication. Then the tweets reader system will generate a csv file mark the frequency of the city names.

2.4 Machine learning classifiers

In order to investigate if the new features generated by approximate match methods and which classifier is better, the following feature sets and classifiers matrix has been attempted:

"best35" features	"Approx17" features	"AddApprox40" features
0-R	0-R	0-R
Naïve Bayes	Naïve Bayes	Naïve Bayes
		1-Nearest Neighbour

Table 2-Feature sets and Classifier Matrix

There are three machine learning classifiers being compared: 0-R, Naïve Bayes and 1-Nearset Neighbour (1NN).

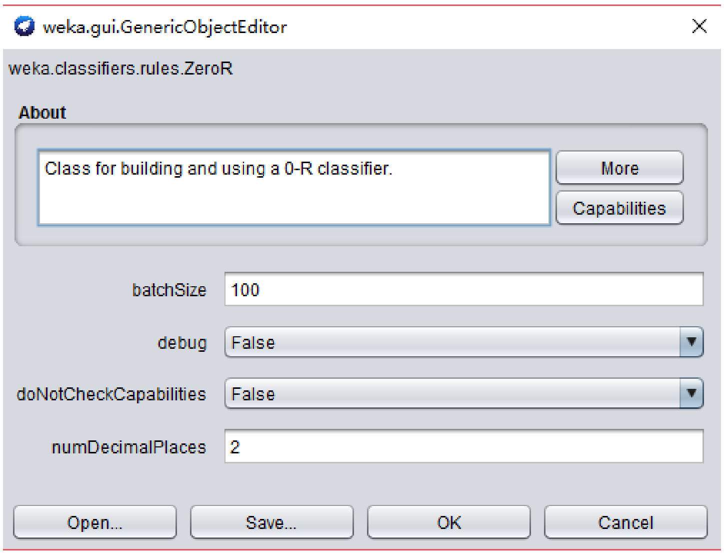
As Zero-R is the most commonly used baseline classifier in machine learning, the project has adopted all three sets of features to train Zero-R classifier as a baseline.

Similar reason to adopt Naïve Bayes, as it is a simple built and fast classifier to use, it should be attempt before trying more sophisticated classifier.

Regarding the problem is classifying locations, the 1-nearest neighbour classifier is expected to have satisfying result so it was attempt as the third classifier although it is relatively time-consuming.

The machine learning classifier “support vector machines” (SVM) is not being used, not only because it is time-consuming but also because it is relatively complex to use SVM in this multi (five) classes problem. Neither does decision trees, because it should be more suitable applying to problems with binary value features to generate binary trees.

In Weka, all the classifiers used the arff file generated from training data set to train, and the arff file generated development data set as test set to generate output report with accuracy. Parameters are set as following figures:



weka.gui.GenericObjectEditor

weka.classifiers.rules.ZeroR

About

Class for building and using a 0-R classifier.

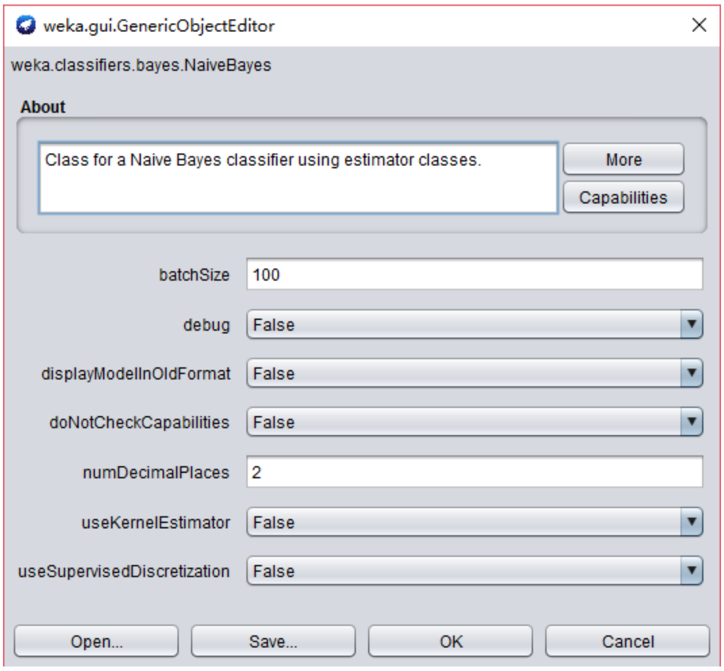
batchSize 100

debug False

doNotCheckCapabilities False

numDecimalPlaces 2

Figure 1-Zero R setting



weka.gui.GenericObjectEditor

weka.classifiers.bayes.NaiveBayes

About

Class for a Naive Bayes classifier using estimator classes.

batchSize 100

debug False

displayModelInOldFormat False

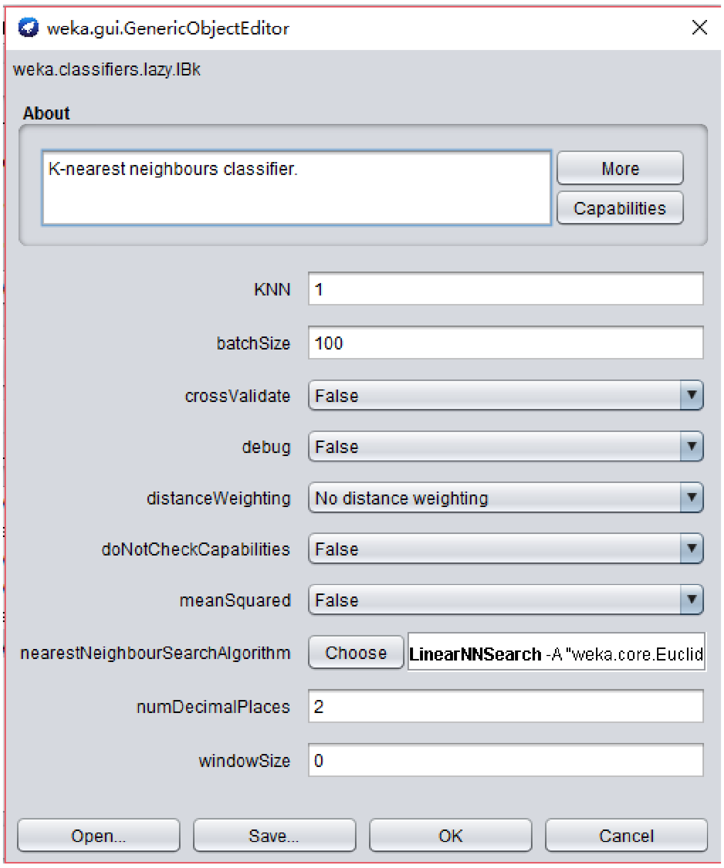
doNotCheckCapabilities False

numDecimalPlaces 2

useKernelEstimator False

useSupervisedDiscretization False

Figure 2-Naive Bayes setting



weka.gui.GenericObjectEditor

weka.classifiers.lazy.IBk

About

K-nearest neighbours classifier.

KNN 1

batchSize 100

crossValidate False

debug False

distanceWeighting No distance weighting

doNotCheckCapabilities False

meanSquared False

nearestNeighbourSearchAlgorithm Choose LinearNNSearch -A "weka.core.Euclid

numDecimalPlaces 2

windowSize 0

Figure 3-1NN setting

3 Evaluation of 0-R, Naïve Bayes and 1-Nearst Neighbour

3.1 Accuracy analysis

Here is the accuracy for the classifiers:

	BEST 35	APPROX17	ADDAPPROX44
ZERO-R	26.1615%	26.1615%	26.1615%
NAÏVE BAYES	28.9266%	21.7563%	**46.7986%**
1-NEARST NEIGHBOUR	--	--	**47.5442 %**

Table 3- Accuracy for classifier

From this table, we have following findings:

a) Naïve Bayes should usually have better accuracy than Zero-R. However, with only 17 features, the accuracy dropped below the baseline. It indicates that Naïve Bayes may not efficient to use if there is small number of features

b) After add 9 more approximate match words as new features, accuracy would be obviously better than original 35 features when using Naïve Bayes classifier, and that is the second best classifier found (marked in green).

c) 1NN took nearly 8000 seconds (while it took Naïve Bayes 10 seconds) to finish testing and it got the best as expected (marked in yellow).

It proved that approximate match words are helpful to identify user's location as it could bring a huge jump for accuracy. In this project, with only 9 features has been added and it improve the accuracy 17.872% compared to the origin best 35 features, it is reasonable to believe that it could have better performance with more training data and more meaningful approximate matched words involved.

As Zero-R method will classify all instance according to the most common class, which is "SD" in this project, three classifiers get the same results as they are using the same training and development data. It is a feature-independent classifier.

As Naïve Bayes using probability distribution to classify, it will be reliable based on the assumption that people from same city will tend to post similar content in tweets. This assumption has been proved to be reasonable by an estimation framework proposed in 2010 (Cheng, et al., 2010). So it could provide better performance than the other two classifiers when there are more than 30 features to calculate possibilities. So the number and the quality of features would determine the performance of this classifier.

For 1NN, it should be suitable to deal with location problems because it views each instance as a node in N dimension space. However, as it need to compare between instances, it took much longer time than Naïve Bayes to get result with similar accuracy. So if consider execution time, it is not the best option to solve the problem.

3.2 Advantages and disadvantages

As the best results came from the Naïve Bayes, the advantages and disadvantages will be discussed in centre of Naïve Bayes classifier.

Generally, Naïve Bayes classifier is easy and quick to build without constraints on input values or data size. Especially compared to the decision tree classifiers usually suitable for binary values and need relatively more resource in time and space to build.

However, it heavily relies on the feature engineering and feature selection to generate more accurate output unlike Zero-R.

Another disadvantage is that, since Naïve Bayes classifier is built based on the assumption that each features are independent with each other, it may not be true for this problem as we used the city name and their approximate matched name both as features.

3.3 Feature Optimization

Based on observation on the tweets, there are tweets from the same user that be marked in the same city. So the user id may also helpful for analysing the data sets in this project. Therefore, the model built using another feature set that add user id to "approx44" with Naïve Bayes classifier was attempt. The result was **47.7952%**, which means 0.9966% improvement has been made only by adding the user id.

3.4 Future improvements

Removing data noise and engineering better features will improve classifier development.

There are data noises observed from the raw training tweets. For example, high possible given a user an unlike city class: For tweet start with id "*1553278271 Just arrived in the Boston area (Westford), MA. Sound check in 15 mins concert starts@7PM. Does anyone know a great place to eat in Boston? SD*" The "Boston" has shown twice and the context showed obviously the user should in Boston while it was classified in San Diego. These data noises from training data should be removed.

As quality of features has been proved to be important for building classifiers (except the baseline classifier), if more relevant features are being used and remove unreverent features should also benefit for classifier development.

4 Conclusion

To classify the location of tweeter user based on tweets need three key aspects to be accurate: high quality (and maybe large quantity) data, efficient features and using suitable classifier according to the raw data and the features. Although Naïve Bayes generated the best model in the scope of this small project, there should be better solution for this problem.

References

Cheng, Z., Caverlee, J. & Lee, K., 2010. *You Are Where You Tweet: A Content-Based Approach to Geo-locating Twitter Users.* New York, ACM New York, NY, USA.

Scott, S. & Matwin, S., 1999. Feature Engineering for Text Classification. *ICML,* Volume 99, pp. 379-388.