Project 2: Geolocation of Twitter Users

October 11, 2014

1 Introduction

The objective of Project 2 was to build a geolocation classifier for Twitter users, based on the content of their Tweets (ASCII strings between 1 and 140 characters in length). The set of nominal classes (Twitter user locations) was restricted to Chicago, New York, San Francisco, Atlanta and Los Angeles. Feature vectors, containing 386 distinct attributes (words), and a large corpus of American Tweets was supplied as part of Project 2. Token-level and semantic generalisation feature engineering methods and Bayesian classification algorithms were explored in Project 2.

2 Preprocessing Input Data - Feature Engineering

Several heuristics were employed in the preprocessing of the the feature vector attributes and the training, development and test twitter files to improve the performance of the classification algorithms explored in Project 2. The preprocessing steps were first employed on the given feature vector attributes, before being employed on the Twitter data, to produce new training, development and test vector instances.

The reduction in the size of the feature vector instance files, due to the preprocessing steps outlined below, are summarised in Table 4.

2.1 Unimelb English Social Media Normalisation Lexicon - Twitter Text Normalisation

Normalising text into a canocial form increases the consistency of the text and reduces the problem sample space (by reducing the number of different tokens to consider). Good text normalisation techniques are highly document dependent, there is no all purpose normalisation procedure.

The supplied Twitter data was normalised using the University of Melbourne's (Unimelb)English Social Media Normalisation (ESMN) Lexicon. The Unimelb ESMN Lexicon mapped OOV words to IV words, therefore a simple (and quick) linear search and substitution through the vector attributes and twitter data was all that was required to (somewhat) normalise the vector attributes and twitter data.

The ESMN Lexicon was chosen due to it offering a fast, lightweight and easy-to-use solution that was suitable for high-volume microblog (Twitter) pre-processing. Further, the ESMN Lexicon had a large volume of normalisation, containing over 40,000 entries [1].

2.2 Word Stemming

Stemming is the process of reducing inflicted words to their stem (base form) [3]. A Porter Stemmer algorithm was used in the preprocessing of the location and tweet files.

The Porter Stemmer algorithm was found to further distort a subset of misspelled words within the tweet bodies, however the Porter Stemmer was also capable of (partially) correcting miss-spelled location words. Further, the Porter Stemmer reduced the sample space of the problem by introducing consistency in tweet words and reducing the number of words in the preprocessed tweets.

A sample of differences between tweet and feature vector strings with and without the Porter Stemmer preprocessing step can be seen in Table 1.

Table 1: Stemming Tweets

	Tweet Text	
No Porter Stemming	yelled egypt phone today true story	family friends atlanta motor speedway first time experience
Porter Stemming	yell egypt phone today true stori	famili friend atlanta motor speedway first time experi

2.3 Removal of Stop Words And Excess White Space

A large amount of tweets and given features contained words which were one character in length and words which were extremely common in the English language (especially after preprocessing stage 2.1 and 2.2). Common words and words of one-character length carried little entropy[4] in determining descriptive features for geolocating a Twitter user (Table 2), and therefore were removed from the vector attributes and corresponding twitter data.

It is worth noting that stop words can cause problems when a tweet contains informative 'geolocating' phrases that include them. However, as a 'bag of words' model was used for modelling the twitter document, information contained within phrases was already lost.

The list of stop words, contained within the NLPK library for python, was used to remove stop words from the locations and tweet texts.

Table 2: Undesirable Features Before Stop Word Removal

Removed Feature Token	**Example Tweets**
'i', 'n', 'so'	omg accid on i n so scari httpyfrogcomhrwadj
'my', 'me'	my mom just told me to updat my websit next i suppos my grandma will tell me my tweet are shit

2.4 Per User Feature Vectors

Initially, a feature vector instance was supplied for each individual tweet. However, as Project 2 involved the geolocation of Twitter USERS, the supplied training, development and test data was restructured to contain a feature vector for each individual Twitter user.

The restructuring of the model vectors to be per-user had the advantage of reducing the number of vectors that had to be processed. The potential disadvantage of the reduction in data-points in the new per-user model was offset by the information being retained in the now richer instance vectors (Table 3).

After restructuring the instances to be per-user, the number of training and development instances was 22,993 and 22,579 respectively. The similarity in the number of training and development instances was inappropriate as ideally the number of training instances should be maximised whilst still allowing enough development instances to sufficiently test the explored machine learning algorithms.

Originally, the ratio of training to development instances was approximately 4 : 1. To re-establish the original balance of training and development data 13,465 development instances were relocated into the training file resulting in 36,458 training instances and 9,114 development instances.

Table 3: Feature Vector Restructuring

	Training and Development Files	
	Per Tweet	**Per User**
Number of Instance Vectors	958,359	45,572
Vectors With At Least 1 Non-Zero Dimension	748,851	44,089
	78.14%	96.75%

Table 4: Preprocessing Input Data Size Reduction

	Attributes	**Training Instances**	**Development Instances**
Raw Feature Vectors	387	766,841	191,518
Processed Feature Vectors	352	36,458	9,114

3 Geolocation Classification Algorithms And Results

The following results were obtained using the preprocessed model space representation (feature vectors) of the Twitter users. The ML implementations available via the Weka GUI framework were explored. Supervised Machine Learning (ML) techniques were explored, that is ML techniques in which the class value of the training data is exposed to the ML algorithms.

A holdout strategy was used to evaluate Weka's ML algorithms. The holdout strategy involved separating the data used for training the ML algorithms, training instances, and the data used for testing the performance of the ML algorithms, development instances. The number of vector instances was considered large enough, and number of classes small enough, to render N-fold validation inappropriate.

The results of the different Weka ML algorithms are summarised in table ...

3.1 0-R and 1-R: Baseline Establishment

0-R and 1-R algorithms are popular rule based machine learning algorithms often used for establishing baselines in machine learning tasks. 0-R and 1-R algorithms have been proven to perform well on many datasets, achieving close to more complex solution accuracies, whilst having drastically reduced complexity [2]. The 0-R and 1-R baselines were used to determine the intrinsic difficulty of Project 1 and gauge the performance of more complex ML solutions.

3.1.1 0-R Decision Rule

The 0-R ML algorithm is (arguably) the most simplistic ML algorithm available in the Weka libraries. 0-R classifies every instance with the class value that occurs must frequently in the training data.

The Los Angeles, LA, class occurred most frequently in the training data, occurring in $\frac{11{,}126}{36{,}458} = 30.5\%$ of instances. Therefore, the Weka 0-R model classified each Twitter user test instance as having being Tweeted from within LA, resulting in a classification accuracy of 29.7%. Further, the 0-R ML model was built extremely quickly, taking only 0.02 seconds to build.

3.1.2 1-R Decision Rule

The 1-R ML algorithm is simplistic in nature and often performs very well on many datasets [2]. 1-R ML algorithm involves making a decision on which class a test instance corresponds to based on a single attribute value. The attribute which minimises the error rate of the decision is used as the splitting attribute.

The Weka 1-R ML algorithm split on the 'twistenfm' attribute, as the 'twistenfm attribute' had the smallest error rate producing $\frac{12{,}764}{36{,}458} = 35\%$ correctly classified instances on the training data. 1-R performed reasonably well correctly classifying 32.9% of development instances. The 1-R ML model was built in the very quick time of 1.31 seconds.

3.2 Naive Bayes

The Naive Bayes classifier is a simple probabilistic classifier based on Baye's theorem (1).

$$P(Class|Attributes) = \frac{P(Attributes|Class)P(Attributes)}{P(Class)} \tag{1}$$

Naive Bayes classifiers rests on two key assumptions:

1. Attributes are equally important, binary values are assigned to each attribute indicating whether they are present or not?????
2. Attributes are assumed to be conditionally independent, that is attributes are assumed to be independent of one another, given a class (2).

The assumption of conditional independence allows simple maximum likelihood estimations to be used to calculate the probabilities within equation. The approximation of attribute independence is often reasonable because the Naive BayesML classifier does not care about the actual probability of $P(Class|Attributes)$ rather it cares about the relative probabilities of $P(Class|Attributes)$.

$$\begin{aligned} P(Class|Attributes) &= \frac{P(A_1|Class)P(A_2|Class)...P(A_N|Class)P(Class)}{P(Class)} \\ &= P(A_1|Class)P(A_2|Class)...P(A_N|Class)P(Class) \end{aligned} \tag{2}$$

The multiplication of a zero probability attribute class conditional can degrade the performance of Naive Bayes, the other probabilities calculations are 'discarded' regardless of their strength. The Naive Bayes method implemented in Weka handles the zero-frequency problem by adding one to the count of an attribute given a class. The 'add-one' solution somewhat alleviates the zero-frequency problem. However, the classification of a class then becomes an add-one competition, the class with the least number of raw add-ones is likely to be assigned.

Weka's Naive Bayes algorithm performed poorly on the development data. Naive Bayes correctly classified fewer training instances than 0-R and 1-R, correctly classifying 28.29% training instances. The poor performance of Weka's Naive Bayes can be attributed to two factors:

1. The loss of information associated with Naive Bayes ignoring the number of times an attribute occurred within a given Twitter user instance.

2. The likely inclusion of (somewhat) redundant attributes in the training and development data. Redundant attributes cause problems with Naive Bayes due to their dependence, violating Naive Bayes key assumption of conditional independance between attributes. It is because of this reason that it is not a case of the more attributes the better for the performance of Naive Bayes.

The Naive Bayes ML model was built in 1.43 seconds.

3.3 Multinomial Naive Bayes

3.4 C4.5 Decision Tree

Table 5: Weka ML Model Build Times

	0-R	**1-R**	**Naive Bayes**	**Multinomial Naive Bayes**	**C4.5 Decision Tree**
Time (seconds)	0.02	1.31	1.43	0.20	226.01

Table 6: Weka ML Training Instance Classification Summary

	0-R	**1-R**	**Naive Bayes**	**Multinomial Naive Bayes**	**C4.5 Deci**
Correctly Classified Instances	29.71%	32.39%	28.28%	44.78%	40.8
Mean F-Score	0.136	0.188	0.259	0.436	0.4

4 Conclusion

X methods were considered for the task of geolocating Twitter users. By employing clever initialisation and exit parameters to the N-W algorithm, the Sub-Distance algorithm, gave the best overall results, .

All algorithms, explored as part of Project 1, often exhibited false positives when considering location names that could also be used in language when NOT referring to a location. Therefore, a future area of research would include algorithms that take into account the context in which words are used.

Further, the shortest runtime for a single tweet to be compared to all queries was 2.8 seconds (Sub-Distance). Therefore, to process all 3.6 million preprocessed tweets against all of the preprocessed locations would take over 116 days. If the whole Location-Tweet sample space had to be analysed faster methods of string approximation, such as the use of Tries or N-gram matching algorithms, are an area for further research.

References

[1] Bo Han, Paul Cook, and Timothy Baldwin. Automatically constructing a normalisation dictionary for microblogs. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 421–432, Jeju Island, Korea, July 2012. Association for Computational Linguistics.

[2] Robert C Holte. Very simple classification rules perform well on most commonly used datasets. *Machine learning*, 11(1):63–90, 1993.

[3] Mirko Popovic and Peter Willett. The effectiveness of stemming for natural-language access to slovene textual data. *Journal of the American Society for Information Science*, 43(5):384 – 390, 1992.

[4] Anand Rajaraman and Jeffrey David Ullman. *Mining of Massive Datasets [electronic resource]. / Anand Rajaraman, Jeffrey David Ullman.* Cambridge : Cambridge University Press, 2011., 2011.