Using the Complexity of the Distribution of Lexical Elements as a Feature in Authorship Attribution

L.M. Spracklin, D.Z. Inkpen, A. Nayak
University of Ottawa
Ottawa, Canada
lspra072@uottawa.ca, diana@site.uottawa.ca, anayak@site.uottawa.ca

Abstract

Traditional Authorship Attribution models extract normalized counts of lexical elements such as nouns, common words and punctuation and use these normalized counts or ratios as features for author fingerprinting. The text is viewed as a "bag-of-words" and the order of words and their position relative to other words is largely ignored. We propose a new method of feature extraction which quantifies the distribution of lexical elements within the text using Kolmogorov complexity estimates. Testing carried out on blog corpora indicates that such measures outperform ratios when used as features in an SVM authorship attribution model. Moreover, by adding complexity estimates to a model using ratios, we were able to increase the F-measure by 5.2-11.8%

1. Introduction

Determining the author of a text is an important problem in computational linguistics. It has applications to plagiarism, copyright infringement and the analysis of anonymously written texts. Normally a machine learning system for authorship attribution extracts features which represent the counts of a variety of lexical elements which are normalized over the length of the text. For example the number of nouns, verbs, common words or punctuation characters may be counted. This is done to fingerprint an author. A model is created based on these features and texts of unknown origin are classified using the model.

The problem with extracting counts of elements is that information is lost. The text is viewed as a "bag-of-words" and the distribution of elements within the text is ignored. It would be useful to have a method of quantifying the distribution of lexical elements within a text. For example, are most of the common words clustered in one part of the text or are they distributed in a near random fashion throughout the text? If one could quantify the distribution of lexical elements then the question still remains as to whether this is a useful feature for authorship attribution and if so is it as good as or better than using only the counts of lexical elements.

This paper proposes a method of quantifying the distribution of lexical elements by using compression. The set of tokens is represented by a binary string. The Kolmogorov complexity of a binary string is the length of the shortest program which can output the string on a universal Turing machine and then stop (Li, 1997). It can be approximated using any lossless compression algorithm. The degree of compression of the string gives an upper bound on the Kolmogorov complexity (Li, 1997).

In this paper, we use a set of blogs as a training/test corpus and a Support vector machine, or SVM, is used to develop models which predict the author. For each blog, nouns, verbs, pronouns, conjunctions, common words, unique words, internet slang words and punctuation characters were identified. Both normalized counts and complexity estimates were extracted for each lexical element. Thus we have 10 features which are based on traditional normalized counts and 10 features which are based on complexity.

We show that the complexity of lexical elements is a better indictor of style then the normalized count of such elements. In addition because ratio and complexity contain different information, both can be used in an authorship attribution model to boost performance without over fitting.

2. Authorship Attribution

Stylometry is concerned with analyzing the linguistic style of text to determine authorship or genre. If one assumes that an author has a consistent style, then one can assume that the author of a text can be identified by analyzing its style.

In order to analyze the style of text, various features are extracted and analyzed. The first attempts at feature extraction focused on treating text as sets of tokens arranged into sentences (Stamatatos, 2000). The extracted features included sentence count, words per sentence, and characters per word. Usually these features were also normalized to the length of the text. These features are still used for classification today.

One can also analyze syntactic features by using a Parts-of-Speech or POS tagger (Stamatatos, 2000). Then the average number of noun phrases, verb phrases, prepositional phrases etc. can then be extracted.

Measures of vocabulary richness are also very important features or style markers (Stamatatos, 2000). One can count the number of words which are very common or the number of words which occur only once or twice. It has been shown that the style of a text can be more easily defined using the presence of common features rather than uncommon features (Uzuner, 2005).

Most work done in authorship attribution and indeed text classification, treat the text as a "bag-of-words", that is order does not matter. Aside from research into n-gram models, the placing of words within the text and the location of some words relative to others is ignored. Obviously this results in a loss of information as we are throwing away structure. Traditionally, this has been considered of no importance.

3. Kolmogorov Complexity

3.1 Introduction to Kolmogorov Complexity

Kolmogorov complexity, also known as algorithmic entropy, stochastic complexity, descriptive complexity, Kolmogorov-Chaitin complexity and program-size complexity, is used to describe the complexity or degree of randomness of a binary string. It was independently developed by Andrey N. Kolmogorov, Ray Solomonoff and Gregory Chaitin in the late 1960's (Li, 1997).

In computer science, all objects can be viewed as binary strings. Thus we will refer to objects and strings interchangeably in this discussion. The Kolmogorov complexity of a binary string is the length of the shortest program which can output the string on a universal Turing machine and then stop (Li, 1997).

Turing showed in his famous work on the halting problem that it is impossible to write a computer program which is able to predict if some other program will halt (Li, 1997). It follows then that even if we find a short program which outputs a particular string there are always other shorter programs and we can never know if one of those programs will halt and if so whether or not they will output the string. Thus it is impossible to compute the Kolmogorov complexity of a binary string. However there have been methods developed to approximate it.

The Kolmogorov complexity of a string x, denoted as $K(x)$, can be approximated using any lossless compression algorithm (Li, 1997). A compression algorithm is one which transforms a string A, to another shorter string, B. The associated decompression algorithm transforms B back into A or a string very close to A. A lossless compression algorithm is one in which the decompression algorithm exactly computes A from B and a lossy compression algorithm is one in which A can be approximated given B. When Kolmogorov Complexity, or $K(x)$, is approximated, this approximation corresponds to an upper-bound of $K(x)$ (Li, 1997). Let C be any compression algorithm and let $C(x)$ be the results of compressing x using C. The approximate Kolmogorov complexity of x, using C as a compression algorithm, denoted $K_c(x)$, can be defined as follows:

$$K_C(x) = \frac{Length(C(x)))}{Length(x)} + q$$

where q is the length in bits of the program which implements C. In practice, q is usually ignored as it is not useful in comparing complexity approximations and it varies according to which programming language implements C. If C was able to compress x a great deal then $K_c(x)$ is low and thus x has low complexity. Likewise if C could not compress x very much then $K_c(x)$ is high and x has high complexity.

3.2 Using Kolmogorov Complexity Estimates to Classify Objects

Suppose an object can be viewed as an ordered collection of n items and each item belongs to a class {c_i | i = 1, 2, …k}. We can then map the object to a string representation which represents the distribution of item classes within the object. Kolmogorov complexity can be used to classify the object by compression this mapping then comparing the degree of compression with the expected compression of different categories of objects. This gives a quantitative measure of the complexity of the distribution if item classes within the object.

It seems intuitive that this can be used for text classification as we can tokenize a text sample and then assign classes to tokens. The Kolmogorov complexity estimate of that distribution can then be used as a feature in machine learning.

One must decide which classes will be used for the Kolmogorov complexity estimates. There is a great deal of flexibility in this but it seems clear that one should consider the meaningfulness of the class and whether or not the distribution of that class is likely to vary among object categories.

Any lossless compression algorithm will estimate Kolmogorov complexity. However, it seems intuitive that when the strings are short, simple compression algorithms will give the best estimate of the complexity. This is because if the string is short, a particular algorithm may compress it a great deal simply by chance. Larger data sets will likely benefit more from efficient compression algorithms as there is a much lower chance that the degree of compression may be an anomaly. Generally, it would not be useful to compute the Kolmogorov complexity of very short strings such as those with length less than 15.

3.3 Filtering Spam using Kolmogorov Complexity Measures

This method of object classification was originally developed for use in spam filters (Seaward, 2007). A common ploy of someone who sends spam or a spammer is to append a segment of text or a list of

keywords at the end of the spam to try to fool the filter into allowing it through. For example, a spammer might send the following message:

Buy your Rolexes here!!!!!!!!!!

Mary put her purse by the door because she knew she would be leaving again.
Suddenly the phone rang and she wondered if Carlos was calling.

A human can look at such an email and immediately discern the disparity in style and semantics and declare it spam. However, a filter which views an email as a "bags-of-words" can be easily fooled into thinking this is a legitimate email.

In order to use Kolmogorov complexity to classify email, we first train the filter to recognize which tokens or words are associated with spam and which are associated with non-spam or ham. Then each email to be classified is tokenized into an ordered set of tokens each of which belongs to the class spam or ham. If we represent ham as '0' and spam as '1' then we may map the email to a string of zero's and one's. The complexity measure is computed by compressing the string using run-length compression,

$$Complexity = \min(1, \frac{2 * runs}{length})$$

Each set of consecutive 1's is a run as is each consecutive set of 0's. The runs are counted to determine how much the string could be compressed if one was to encode it as runs of 0's and 1's. For example 11101000 could be encoded as a run of three 1's, followed by a run of one 0, followed by a run of one 1, followed by a run of three 0's.

The complexity of the string is compared to a threshold t and if it is greater than t then the email is classified as spam otherwise it is classified as ham. This method was used to classify email with accuracy of 81-96% (Seaward, 2007).

3.5 Attributing Authorship using Kolmogorov Complexity Measures

Authorship attribution is an interesting problem for Kolmogorov Complexity measures. Tokens or words in a text sample can be divided into many different meaningful classes such as lexical type (noun, verb, preposition etc.); length (short, medium or long); common or uncommon; slang word or proper word etc. It is obvious that the distribution of such measures within the text is a meaningful measure of the style of such a text.

Indeed, Zipf's law states that in any corpus, the frequency of any word is inversely proportional to its rank in the frequency table (Manning, 1999). Linguists agree that most language consists of a great many common and possibly ambiguous words with a small number of relatively uncommon and unambiguous words thrown in (Manning, 1999). This is to reduce the burden on the speaker and listener to know and understand many different words and ensure the listener understands the message clearly. However, it is intuitive that this mix of words would be a good way to fingerprint an author as it seems logical that the distribution of such common/uncommon words will vary by author.

The main question that arises is whether the complexity of a feature's distribution is more meaningful then the count of such a feature. For example is it more meaningful to say the complexity of common words was 0.32 or to say that out of 1000 words, 700 were common. Either measure loses information. Complexity measures lose magnitude. For example, consider the following two strings and suppose '1' represents a common word and '0' represents all other words. The second string has twice as many common words but they have the same complexity.

String	Complexity	Ratio
00001000010000	0.714	0.143
01111011110000	0.714	0.571

Likewise, if we use ratios, we lose information about how the common words were distributed. For example, the following two strings have the same ratio of 0.5 but very different distributions.

String	Complexity	Ratio
11111110000000	0.143	0.500
10101010101010	1.000	0.500

As we will see in the results section, both measures can be used without overfitting as the both contain information which is useful in fingerprinting an author. Surprisingly however, the complexity of a feature's distribution performs better than the ratio of that feature when performing authorship attribution. The best results are obtained when both ratio and complexity measures are used.

4. The Blog Corpus

All blogs were taken from Moshe Koppel's blog corpus which is a collection of 681,288 blogs from 19,320 authors or bloggers (Schler, 2006). Blog is a combination of the words "web" and "log" and is thus a weblog or internet diary. Generally blogs are posted frequently through a website which supports such postings. Koppel obtained all the files in the blog corpus from www.blogger.com. The files are annotated with the bloggers purported age, gender, industry and astrological sign.

Koppel et al. obtained good results using almost the entire corpus for author attribution (Koppel, 2006). Using only those posts with over 200 words, he obtained 18,000 blogs for testing/training. He extracted weighted measures of words that would be representative of topic, function words and unique non-numeric non-alphanumeric words (such as smileys). He used information retrieval techniques instead of a machine learning model and in 66.0% – 79.5% of cases his system could make no conclusions and returned "I don't know", otherwise his system was over 80% accurate.

Schler et al. also used this corpus to build a model which predicts gender and age with over 80% accuracy (Koppel, 2006).

Since the corpus contained far too many blogs and bloggers for a machine learning author attribution model, I selected a subset of bloggers to work with. By analyzing length of posts, I obtained a set of 19 authors each of which had over 37 blogs of length over 1000 words. Generally this is the minimum length required for accurate authorship attribution (Stamatatos, 2000). I divided these authors into 2 sets. Data set A is a balanced data set in which most authors have the same number of large (length > 1000) blogs. The mean is 43.40 blogs and the standard deviation is 3.31. Data set B is less balanced and the mean number of blogs is 60.56 and the standard deviation is 28.87. There were other blogs by each of these 19 authors but only posts of length greater than 1000 were used. Note that I refer to bloggers and authors interchangeably in this report.

Table 4.1 Details of Authors and their blogs in Data Set A.

Author	Gender	Age	Posts of Length > 1000
a1	male	24	46
a2	male	24	40
a3	male	47	44
a4	male	41	42
a5	male	17	36
a6	female	26	47
a7	male	36	45
a8	male	25	46
a9	female	47	44
a10	male	25	44

Table 4.2 Details of Authors and their blogs in Data Set B.

Author	Gender	Age	Posts of Length > 1000
b1	male	25	89
b2	male	27	62
b3	male	33	112
b4	female	25	38
b5	male	15	76
b6	male	44	54
b7	male	37	37
b8	female	43	39
b9	female	14	38

5. Weka

Weka is a collection of machine learning algorithms and data processing tools (Witten, 2005). It is available for free download and it is very easy to use. It was developed at the University of Waikato in New Zealand. There are a great deal of machine learning tools included in the Weka package such as trees, linear regression, neural networks, naïve Bayes and support vector machines. A support vector machine was used in our experiments.

6. Support Vector Machines

A support vector machine or SVM is a supervised learning method used to classify data (Witten, 2005). Each instance in the training set is represented as a set of n features which correlates with an n-dimensional data point. The entire set of training examples if viewed as a set of data points in n-dimensional space and the SVM attempts to find the hyperplanes which best divide the space between each pair of classes such that the largest possible number of data points are on the same side and the distance between each class and the hyperplane is maximized (Witten, 2005). The optimal hyperplane is one which minimizes the risk of misclassifying a data point. The data points which are closest to the maximum margin hyperplane are known as support vectors (Witten, 2005).

Thus for our problem there are 20 dimensions and 9 or 10 classes or authors thus the SVM will find a maximal margin hyperplane which separates each possible pair of classes of which there are 34 or 43. Suppose we label the features f_1, f_2, …f_{20}, then the SVM finds a vector of the form $x = w_0 + w_1 f_2 + w_2 f_2 + … + w_{20} f_{20}$ to divide each possible pair of classes. This will be done for each pair of classes and each fold in 10-cross validation.

For this project the SMO or sequential minimal optimization implementation of an SVM in Weka was used. By default the SMO uses polynomial of Gaussian kernels, transforms nominal vales into binary ones and normalizes attributes (Witten, 2005). Other machine learning tools in Weka were also tested but SVM proved to be very competitive and seemed a natural fit for the problem.

7. Methodology

7.1 Overview

In order to build an authorship attribution machine learning model there are several steps which must be followed. In the table below, we outline each step and detail how the step was completed

Table 7.1 Steps in building an SVM model for
Authorship Attribution

Step	Description	Details
1	Obtain corpus	Moshe Koppel's blog corpus which is available for free download was used.
2	Pre-process corpus	I reduced corpus to 19 authors and removed all blogs less than 1000 words.
3	Extract features	This was done using Python scripts. The main scripts are given in appendix B and C.
4	Place features in format for machine learning toolkit	An ARFF file was created for each data set.
5	Normalize features	Done automatically by Weka.
6	Create models	Done by selectively removing features from set of 20 in Weka.
7	Prune features	Weka's attribute evaluator function was used to identify the possible candidates for pruning. I tested each model for each data set with the two bottom ranked features removed. Those features were pruned if this improved performance.
8	Evaluate Results	Precision, recall, F-measure and confusion matrix are given for 10-fold cross validation by Weka. Full results are in appendix A.

7.2 Feature Extraction

We started with the 20 features listed below. In order to identify slang words, the internet dictionary from www.noslang.com was used (www.noslang.com). Whether or not a word was unique or common was based on token frequency analysis for the entire blog corpus.

Parts of Speech tagging was done using a simple python tagger developed by Jason Wiener (2006) and based on the work of Eric Brill (Brill, 1996). It is a rule-based system which uses transformations and is error-driven. The algorithm has been implemented in many platforms and is known as the Brill Tagger.

For the features below, all counts were normalized to the number of words/tags in the text or in the case of punctuation the number of characters in the text. The method of constructing the string mapping for complexity estimates is given below. After the string is constructed run-length compression was then applied to get an estimate of the Kolmogorov complexity of the mapping.

Table 7.2.1 Features and their Descriptions.

Attribute	Description
commoncount	Count all words which occur more than 1000 times in the entire blog corpus.
commoncomplexity	If a word occurs more than 1000 times in the entire blog corpus then count as '1' otherwise count as '0'.
uniquecount	Count all words which occur less than 3 times in the entire blog corpus.
uniquecomplexity	If a word occurs less than 3 times in the entire blog corpus then count as '1' otherwise count as '0'.
slangcount	Count all words which appear in dictionary from www.noslang.com and divide by the total number of tokens.
slangcomplexity	If word is in no-slang dictionary than count as '1' otherwise count as '0'.
nouncount	Count all tokens which are tagged as noun phrases and divide by the total number of tags.
nouncomplexity	If a token is a noun phrase then count as '1', otherwise count as '0'.
verbcount	Count all tokens which are tagged as verb phrases and divide by the total number of tags.
verbcomplexity	If a token is a verb phrase then count as '1', otherwise count as '0'.
adverbcount	Count all tokens which are tagged as adverbs and divide by the total number of tags.
adverbcomplexity	If a token is an adverb then count as '1', otherwise count as '0'.
adjectivecount	Count all tokens which

	are tagged as adjectives and divide by the total number of tags.
adjectivecomplexity	If a token is an adjective then count as '1', otherwise count as '0'.
conjunctioncount	Count all tokens which are tagged as conjunctions and divide by the total number of tags.
conjunctioncomplexity	If a token is a conjunction then count as '1', otherwise count as '0'.
punctuationcount	Count all characters which are punctuation and divide by the total number of characters.
puncuationcomplexity	If a token is a adjective then count as '1', otherwise count as '0'.
averagewordlength	Sum all word lengths and divide by the total number of words.
wordlengthcomplexity	If a word is less than or equal to 4 characters then count it as '0'. If it is greater than or equal to 6 characters then count it as '1'. Otherwise ignore the word.

7.3 Pruning Features

Using Weka's SVM attribute evaluator, the following rankings were obtained. In using an SVM, feature selection is very important. Since there are 20 features, there are $2^{20}-1$ or 1048575 possible combinations of features. Thus it is impossible to try all combinations of features to determine which combination results in the best performance. This tool was very useful for providing an idea as to which features may be likely candidates for pruning. The bottom two ranked features were pruned both individually and separately to determine if this increased accuracy.

Table 7.3.1 Features Pruned for each Model and Data Set.

Data Set	Model	Features Pruned
A	Complexity-Ratio	slangcount
A	Complexity	commoncomplexity
A	Ratio	-
B	Complexity-Ratio	-
B	Complexity	-
B	Ratio	-

8. Results

When evaluating a machine learning system there are three commonly used measures of performance. They are precision, recall and F-measure. Recall is the proportion of spam messages which were correctly identified as spam. Precision is the proportion of emails which are identified as spam and which are actually spam. The F-measure is a combined measure which equally weights recall and precision

FN = False negatives
FP = False positives
S = Total number of samples of the class or author

$$\mathrm{Re}\,call = \frac{S-FN}{S},$$

$$\mathrm{Pr}\,ecision = \frac{S-FN}{S-FN+FP},$$

$$F-measure = \frac{2(\mathrm{Pr}\,ecision * \mathrm{Re}\,call)}{\mathrm{Pr}\,ecision + \mathrm{Re}\,call},$$

Here are the results for all six experiments. All these results are based on 10-fold cross validation. Detailed data on each class and confusion matrices are in appendix A.

Table 8.1 Precision, Recall and F-measures for each Model and Data Set

Model	Data Set	Precision	Recall	F-measure
Ratio Model	A	0.651	0.651	0.651
Complexity Model	A	0.665	0.665	0.665
Complexity-Ratio Model	A	0.703	0.704	0.704
Ratio Model	B	0.741	0.741	0.741
Complexity Model	B	0.787	0.787	0.787
Complexity-Ratio Model	B	0.859	0.859	0.859

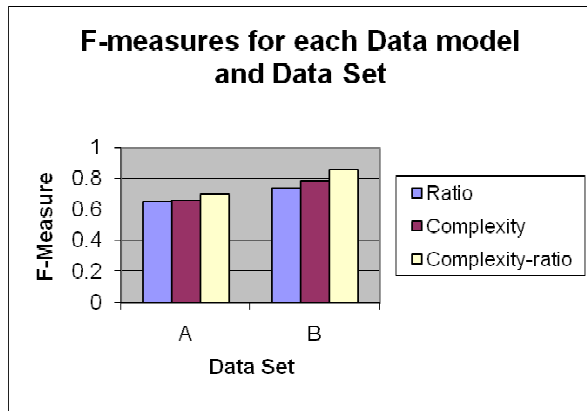**F-measures for each Data model and Data Set**

Figure 8.1 F-measures for each Model and Data Set.

The results are better for data set B then data set A for two reasons. Firstly B is bigger than A. This generally always increases accuracy. Also if you view the confusion matrices for data set A (table A2, A4 and A6 in appendix A), there was a great deal of confusion between authors a1 and a2. In fact a2 was never classified correctly for any of the instances for any of the models and in fact was always classified as a1. What is also interesting is that both a1 and a2 are 24 year old men. There are two possibilities. Schler et al. show that in blogging, gender and age have an affect on the use of lexical features (Koppel, 2006). It is also possible that they were both written by the same person. On the internet no one is ever who they say they are. I chose not to remove a2 from the training set as we did not want to run the risk of cherry-picking data.

9. Conclusion

It has been shown that the addition of Kolmogorov complexity measures of lexical features increases the accuracy of a feature-based author attribution SVM model by 5.2%-11.8%. In fact the Kolmogorov complexity estimate of a feature is a better measure of style than the normalized count of such a feature. This is an important result because this means we can boost accuracy with the addition of very little extra computation. If a system is already extracting features and counting them then it is trivial to also compute the complexity measure for that feature as well. Our results also show that some information about style is lost when we view text as a "bag-of-words" and it is possible to add some of this information back into an natural language processing model through the use of feature complexity estimates.

10. Future Work

There are several things that can be done to expand this method. Firstly, one could experiment with the Kolmogorov complexity estimates of different features then the ones used in this project. One could also experiment with the complexity estimates of non-binary classes. For example what about it we counted nouns as 0 and verbs as 1 and all others as three and then computed complexity. It would also be interesting to use other compression algorithms than run-length compression.

11. References

Brill E. (1995) "Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part-of-Speech Tagging" Computational Linguistics vol. 21 no. 4 pp 543-565, 1995.

Internet Slang Dictionary www.noslang.com

Joachims T. (1998). "Text categorization with Support Vector Machines: Learning with many relevant features" *In Machine Learning: ECML-98, Tenth European Conference on Machine Learning*, pp. 137—142, 1998.

Keselj V., Peng F., Cercone N., and Thomas C. (2003) "N-gram-based Author Profiles for Authorship Attribution" In *Proceedings of the Conference Pacific Association for Computational Linguistics, PACLING'03*, Halifax, Nova Scotia, Canada, pp. 255--264, August 2003.

Koppel M., Schler J., Argamon S. and Messeri E. (2006). "Authorship Attribution with Thousands of Candidate Authors" *(poster) in Proc. Of 29th Annual International ACM SIGIR Conference on Research & Development on Information Retrieval, August 2006*.

Li M. and Vitanyi P. (1997) "An Introduction to Kolmogorov Complexity and its Applications" Second Edition, *Springer Verlag*, Berlin, pages 1-188, 1997.

Manning C., Schütze H. (1999) "Foundations of Statistical Natural Language Processing" pp 23-35, MIT Press, 1999.

Schler J., Koppel M., Argamon S. and Pennebaker J. (2006) "Effects of Age and Gender on Blogging" in *Proceedings of 2006 AAAI Spring Symposium on Computational Approaches for Analyzing Weblogs*.

Seaward L. and Saxton L.V. (2007), "Filtering spam using Kolmogorov complexity measures", to appear in *The Proceedings of the 2007 IEEE International Symposium on Data Mining and Information Retrieval (DMIR-07)*, (Niagara Falls, May 21-23, 2007).

Stamatatos E., Fakotakis N., and Kokkinakis G. (2001). "Computer-Based Authorship Attribution without

Lexical Measures" *Computers and the Humanities*, 35(2), pp. 193-214, Kluwer, 2001.

Stamatatos E., Fakotakis N., and Kokkinakis G. (2000). "Automatic Text Categorization in Terms of Genre and Author" *Computational Linguistics*, 26:4, pp. 461-485, 2000.

Uzuner O. and Katz B. (2005) "A comparative study of language models for book and author recognition" *In Proceedings of the 2nd International Joint Conference on Natural Language Processing* (IJCNLP-05), 2005.

Weka Project http://www.cs.waikato.ac.nz/ml/weka/

Wiener J. (2006) NLP Parts of Speech Tagger http://jcay.com/python/scripts-and-programs/development-tools/nlp-part-of-speech-tagger.html

Witten I.H., and Frank E. (2005) "Data Mining: Practical machine learning tools and techniques", pp. 341-410 2nd Edition, Morgan Kaufmann, San Francisco, 2005.