NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

AUTHOR ATTRIBUTION ON A MOBILE PHONE

by

Jody Grady

March 2011

Thesis Advisor:	Robert Beverly
Second Reader:	Craig Martell

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
3–2–2011	Master's Thesis	2009-03-01—2011-03-25

4. TITLE AND SUBTITLE

Author Attribution on a Mobile Phone

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Jody Grady

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Reconnaissance Office

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited

13. SUPPLEMENTARY NOTES

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. **IRB Protocol Number: n/a**

14. ABSTRACT

Put something abstracty here.....

15. SUBJECT TERMS

Machine Learning,Author Attribution,Mobile Computing, Supervised Learning

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER *(include area code)*
Unclassified	Unclassified	Unclassified	UU	63	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

AUTHOR ATTRIBUTION ON A MOBILE PHONE

Jody Grady
Commander, United States Navy
B.E. Georgia Institute of Technology, 1994

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 2011

Author: Jody Grady

Approved by: Robert Beverly
Thesis Advisor

Craig Martell
Second Reader

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Put something abstracty here.....

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

Table of Contents

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgements

..... I would like to thank all the little transistors that computed the results I got here....

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2:
Prior and Related Work

2.1 Introduction

Author detection is the process of analyzing documents to determine if that document was created by a per-determined set of authors.Detecting authors on mobile devices requires selection of a feature set that distinguishes authors, selection of techniques that effectively uses these features to identify authors, selection of efficient machine learning

2.2 Author Detection

”Automated authorship attribution is the problem of identifying the author of an anonymous text, or text whose authorship is in doubt” [1]. For this thesis, author detection and authorship attribution are synonymous. The explosive growth of communications and document storage on the Internet provides a vast amount of data to draw on for author detection. Books, articles, blogs, tweets, and emails are posted for public viewing in an electronic format every day. Some of these postings have verifiable authors. Many Internet authors use nom de plumes or are posted anonymously. Matching verified authors to anonymous Internet authors or mobile phone texters has numerous practical applications. The increased speed and storage capacity of computing devices allow analysis of these corpora for author detection. The methods of author detection fall within the science of machine learning.

2.3 Machine Learning

”Machine learning is programming computers to optimize a performance criterion using example data or past experience” [2]. Machine learning has been used famously to determine the authors of the Federalist papers, allow computers to ”read” human handwriting, and to mine sales data for profitable trends. Two broad categories of machine learning are supervised learning and unsupervised learning. Supervised learning is ”learning with a teacher.” The teacher can show the learner what to do based on examples or experience. Unsupervised learning is ”learning with a critic” [2]. This thesis relies exclusively on supervised learning. Mobile device limitations demand author identification models be constructed on a platform more powerful than a mobile device. That model is then put on a device for ongoing author identification. The models require previous ”teaching” instead of predictive ”criticizing”.

Machine learning is comprised of a set of classes, a classifier, a feature set, and data. In supervised learning, the machine learner uses a data input comprised of features fitted to (or owned by) by a specific class. Based on creatively counting these features, the machine learner creates a model for each class based on the the behavior of the classifier. Finally, test data, consisting of sets of features, are processed by the classifier based on the previously built models. The classifier provides an output of the most likely class that fits the given features.

Machine learning is central to this thesis. Modeling corpora of emails and tweets from numerous authors on large computers, and, then, testing prediction capability on mobile devices requires not just accurate machine learning, but efficient machine learning. The efficiency is needed due to the limits of even the most advanced mobile devices. Hardware specifications are not the only limiting factor in machine learning. There are competing strengths and weaknesses in the techniques chosen, as well.

2.3.1 Machine Learning Techniques

The techniques in this thesis are all supervised machine learning techniques. Specifically, the two supervised techniques used are Naive Bayes and Support Vector Machine (SVM). Naive Bayes was chosen because it is computationally lightweight compared to many other methods. Support Vector Machine was chosen because data for SVM can be stored in "sparse format". Sparse means that every feature does not have to be represented in the stored data for a model or test case. Features with a zero count can simply be excluded. SVM has been successful in many other authorship attribution experiments [3].

Naive Bayes

Naive Bayes is a supervised learning method that uses Bayes Rule of probability chaining over a set of features (words in a document) to arrive at an overall probability that a specific a set of features (words in a document) belongs to a particular class (specific author). Naive Bayes uses a strong independence assumption among the various features. This means that the classifier assumes that the probability of one feature appearing in a data set is completely independent of another feature showing up in the data set. While this assumed independence of features is unlikely to be actually true, the independence keeps the calculation of probabilities simple. In the case of documents and authors, Naive Bayes represents a bag of words model of a document where word order is lost and only frequency or occurrence of words or word combination is captured. Based on a set of words, t, of size n, the probability that a document,

d, belongs to a given class, c, is given by:

$$P(c|d) \propto P(c) \prod_{i<k<n_d} P(t_k|c) \tag{2.1}$$

To specifically apply the above equation to author detection, the classifier returns the class with the highest probability after executing the above formula. This turns the above equation into a maximum a posteriori (MAP) class c_{map}:

$$c_{map} = arg\,max\ \hat{P}(c|d) = arg\,max\ [\hat{P}(c) \prod_{i<k<n_d} \hat{P}(t_k|c)] \tag{2.2}$$

Since underflow is an issue when numerous float values are multiplied together over a set of features, the practical application of the above formula is:

$$c_{map} = arg\,max\ \hat{P}(c|d) = arg\,max[log\ \hat{P}(c) + \sum_{i<k<n_d} log\ \hat{P}(t_k|c)] \tag{2.3}$$

Because the probability of each feature is multiplied by the probability of every other feature, a zero probability for any feature will make the overall probability zero. To handle this issue, a technique called smoothing is used. Smoothing is a method of adding some non-zero values to each feature to prevent zero values. The simplest form of smoothing is Laplace Smoothing (Plus One Smoothing). In this method, each feature in the feature set is initialized with a count of 1 instead of zero. The denominator in the probability equation is increased by $1 * numberoffeatures$ to account for all the added ones. This method, attractive in its simplicity, often produces undesirable results. For this thesis, the counts from words in the Google Web1T corpus are used to smooth word counts in Naive Bayes. For example, the word "dog" appears 3,450,297 time in the Web1T corpus, so the count for "dog" is initialized to 3,450,297. The denominator for a Google Web1T smooth Naive Bayes instance is 1,024,908,267 based on total count weight of all tokens in the corpus.The specific details of the Google Web1T corpus are covered in a later section of this chapter.

Support Vector Machine

A Support Vector Machine (SVM) is a supervised machine learning method that finds a separating line or shape through a set of data based on a selected feature set. This is based on

finding a boundary between two types of data in a dataset, then computing the largest boundary between closest data points and the boundary. In cases where a clear boundary between two data sets is not possible, a "slack variable" provides an allowance of data points to be on the wrong side of the boundary. To create the boundary, SVM "maps the input vectors into some high dimensional feature space, Z, through some non-linear mapping chosen a priori" [4]

For the two situations that a SVM can encounter: data can be separated without error and data cannot be separated without error, the same equation can be used. In the first situation, where data can be separated without error, the SVM optimizes the SVM base equation with $C = 0$. For the second situation, where the training data cannot be strictly separated, $C > 0$:

$$\min_{w,\alpha} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i}^{n} \xi (2.4)$$

where ξ is known as the slack variable, C is the error penalty, and the entire term $C\sum_i^n \xi$ is the soft margin. This is a quadratic programming problem to find ξ and C, often accomplished by a logarithmic grid search ($C = 2^-5, 2^-3, 2^-1, 2^1, 2^3, 2^5$ and $\xi = 2^-15, 2^-10, 2^-5, 2^0, 2^5$) with the best accuracy or F-Score determining where to continue refining the grid.

Historical Roots of Support Vector Machines SVM historical roots lie in the R.A. Fischer's pattern recognition work using a Variance/Covariance matrix [5]. Fisher's pattern recognition used the mean matrix (also know as the centroid of a matrix) and variance-covariance matrix (also known as the dispersion of a matrix) of two normal distributions, and found the optimal Bayesian solution was a non-linear function. Fisher simplified his non-linear function to a linear function for situations where the dispersion of both normal distributions are equal. He even found that his simplified linear equation worked satisfactorily when the distributions needing patterns recognized were not strictly normal.

From this basis, Fisher created a precedent of pattern recognition based on linear discriminating surfaces within a multi-dimensional space. Fisher's work was furthered by perceptron work in the 1960's. This work created multiple linear discriminating surfaces to find a matching pattern. However, there was no method to optimize the separation between data using perceptrons. From the need to optimize the separation, feedback mechanisms were developed to refine the perceptron weights. By further developing the idea of feeding back to the perceptron weights, SVMs were created.

Optimal Hyperplane in Feature Space

The core of SVM is finding an optimal hyperplane in the higher dimension space mapped from the original feature space. That hyperplane is defined as:

$$\mathbf{w_0} \cdot \mathbf{z} + b_0 = 0 \tag{2.5}$$

where w_0 are weights, z is the space, and b_0 is still a mystery to me. To that end, $\mathbf{w_0}$ "can be written as some linear combination of support vectors." This uses the following equation:

$$\mathbf{w_0} = \sum_{support\ vectors} \alpha_i \mathbf{z_i} \tag{2.6}$$

and the decision function using those weights is given by

$$I(z) = sign\left(\sum_{support\ vectors} \alpha_i \mathbf{z_i} \cdot \mathbf{z} + b_0\right) \tag{2.7}$$

meaning that $I(z) < 0$ for one class and $I(z) > 0$ for the other class.

For distance ρ between projections defined by the support vectors, ρ is defined as:

$$\rho(\mathbf{w}, b) = \min_{x:y=1} \frac{\mathbf{x} \cdot \mathbf{w}}{|\mathbf{w}|} - \max_{x:y=-1} \frac{\mathbf{x} \cdot \mathbf{w}}{|\mathbf{w}|} \tag{2.8}$$

given that (2.5) it follows that the weights needed to create the optimal hyperplane are given by

$$\rho(\mathbf{w_0}, b_0) = \frac{2}{|\mathbf{w_0}|} \tag{2.9}$$

The best solution maximizes the distance ρ. To maximize ρ, you must minimize the magnitude of $\mathbf{w_0}$. Find that minimum $\mathbf{w_0}$ is a quadratic programming issue.[4]

Procedure "Divide the training data into a number of portions with a reasonable small number of training vectors in each portion. Start out by solving the quadratic programming problem determined by the first portion of training data. For this problem there are two possible outcomes: either this portion of the data cannot be separated by a hyperplane (in which case the full set of data as well cannot be separated), or the optimal hyperplane for separating the first portion of the training data is found." If this first set is found to be linearly separable, then all

the non-support vector values are discarded, a new batch of values are put into this set (these values do not meet the constraint of $y_i(\mathbf{w} \cdot \mathbf{x_i} + b) \geq 1, i = 1, ..., l$)

Soft Margins In cases where the data is not linearly separable, the goal becomes to minimize the number of errors (the number of values on the wrong side of the hyperplane). Now a new variable $\xi \geq 0, i = 1, ..., l$ is introduced along with the function $\Phi(\xi) = \sum_{i=1}^{l} \xi_i^{\sigma}$. The constraints are that the value ξ_i does not push values in the non-negative quadrant out of the negative quadrant ($y_i(\mathbf{w} \cdot \mathbf{x_i} + b) \geq 1 - \xi_i, i = 1, ..., l$. Also, ξ_i is zero or a positive number ($\xi_i \geq 0$). ξ here represents "the sum of deviations of training errors" The central equation for minimizing the number of errors is:

$$\frac{1}{2}\mathbf{w^2} + CF(\sum_{i=1}^{l} \xi_i^{\sigma}) \tag{2.10}$$

In cases for ξ_i^{σ} where $\sigma = 1$, we are dealing with the soft margin hyperplane. Cases where $\sigma < 1$, there may not be a unique solution. For values of $\sigma > 1$, there are also unique solutions, but $\sigma = 1$ is the smallest value and that allows the term $CF(\sum_{i=1}^{l} \xi_i^{\sigma})$ from (2.10) to not overwhelm the $\frac{1}{2}\mathbf{w}^2$.[4]

Multi-Class SVM SVM is an inherently binary classifier. However, SVM can process multi-class data sets using SVM. There are two approaches to applying a binary classifier to a multi-class data set: one-versus-all and one-versus-one. In one-versus-all, each class in the training set is singled out against the conglomerated remaining classes in the training set. Whichever class achieves the best separation is labeled as the correct class for that data. In one-versus-one, the data classes in the training set are paired against each other and the best comparison among pairs is labeled as the correct class for that data.

It is important to define what is meant by "best" in the classification process. Best is defined as the class that nets the most positive results from individual data instances in the training set. Settling ties, should they occur is implementation dependent, sometimes as simple as making a random choice among the tied classes.[6].

2.3.2 Machine Learning Tools

There are many machine learning toolkits available. These tools come in both open source and proprietary forms. Tools are chosen based on techniques used, so, for this thesis, libSVM

and libLinear were examined as SVM tools. Naive Bayes was constructed from scratch for customization with Google Web1T.

LibSVM

LibSVM attempts to optimize the basic SVM equation:

$$\min_{\mathbf{w},b,\xi} \frac{1}{2}\mathbf{w}^{\mathbf{t}}\mathbf{w} + C\sum_{i=1}^{l}\xi_i \tag{2.11}$$

$$\text{subject to } y_i(\mathbf{w}^{\mathbf{t}}\phi(\mathbf{x}_i) + b) > 1 - \xi_i \tag{2.12}$$

$$\text{and } \xi_i > 0 \tag{2.13}$$

For all kernels used in SVM a variable that must be solved for prior to optimization, the penalty term, C. Other kernels have additional variables that must be solved for prior to optimization, such as γ in the RBF kernel. While there are sophisticated methods to find C and other required variables, LibSVM takes a simple, straightforward approach: grid search. The grid for this search is a log grid search. As the local minimum is found on each pass of the grid search, libSVM reduces the grid size to home in on the minimum C value.

To make libSVM more efficient and more likely to converge on a solution, data in the training set should be scale to either span 0 to +1 or -1 to +1. While test data may show up outside the original training data range, libSVM will extend the normalized range to accommodate. For example, if the range of the training data was -100 to +100, libSVM would scale that range to -1 to +1. If there was test datum with a value of -110, then libSVM would scale that datum to -1.1. While it is stated here that libSVM scales data, that function is not automatic within libSVM itself, but is rather part of the libSVM.

LibSVM was originally constructed in C and employed with python tools to support. LibSVM is not available in a wide array of languages, including Java. A Java version of libSVM makes libSVM functional on many of the mobile operating systems available today, including Android. For this reason, libSVM was originally chosen as the SVM tool for this thesis.

LibLinear

While libSVM has numerous kernels to improve results, the inclusion of code to accommodate these kernels slows libSVM down. To increase processing speed for libSVM for linear kernels, libLinear was created. LibLinear is heavily modeled on libSVM but without non-linear kernel

support. The kernels, represented within the ϕ function in SVM equations is not dealt with at all in libLinear, thus cutting down on checks and processing time. A linear kernel has been found to give as good or nearly as good a result as other kernels such as RBF, parabolic, and radial for text classification, especially when the corpus being used is large. The reduction in code can produce results 100-200 times faster that using LibSVM.

LibLinear has also been studied for large data sets that produces models which cannot be fit into memory. the application of "chunked" data on a mobile platform with very limited RAM, but significant storage (due to microSD cards) makes libLinear even more attractive for mobile device use.

2.4 Features

2.4.1 Feature Types

Feature types for natural language processing can be as simple a keeping counts of individual characters within a document to complex tracking of word combinations. There are three feature types used in this thesis, N-Grams, Gappy Bigrams, and Orthogonal Sparse Bigrams. These feature types vary in complexity and effectiveness for author detection.

N-Grams

N-grams are word groups or character groups of size N within a document. These word groups can include sentence boundaries, often denoted as $<$ S $>$ for sentence start and $<$ /S $>$ for sentence end. For instance, in the phrase the "the quick brown fox" the set of 2-grams (bigrams) are shown in Table 2.1:

Table 2.1: N-grams (N=2) of "the quick brown fox" with sentence boundaries

$<$ S $>$ the
the quick
quick brown
brown fox
fox $<$ /S $>$

To further illustrate, the 3-grams (N=3 N-grams) of the phrase "the quick brown fox" are show in Table 2.2:

Table 2.2: N-grams (N=3) of ”the quick brown fox” with sentence boundaries

< S > the quick
the quick brown
quick brown fox
brown fox < /S >

The larger the N-Gram, the lower the probability of finding that N-Gram in a document. A specific 5-Gram may be very rarely repeated, even by the same author. That makes a 5-gram distinctive, but unreliable for author detection. A 1-Gram like ”the”, ”of”, ”a”, etc occurs frequently across almost all authors, but is not discriminating. Finding discriminating words groupings without the unreliable low probability of large-N N-Grams drove the creation of a modified N-Gram grouping called a Gappy Bigram.

Gappy Bigrams

Gappy Bigram definitions vary between the sources cited in this thesis. For the purposes of this thesis, a Gappy Bigram will be composed of two tokens (words) found within a distance of words. A Gappy Bigram of distance 0 reduces to an identical set to 2-Grams (also know as bigrams). Just like N-Grams, Gappy Bigrams can extend beyond a sentence boundary, include punctuation, etc. However, for larger distances, the distinction between Gappy Bigrams and regular bigrams is clear. For instance, in the phrase ”the quick brown fox” and a Gappy Bigram distance of 2, the Gappy Bigrams are given in Table 2.3:

Table 2.3: Gappy Bigrams (of distance 2) of ”the quick brown fox” with sentence boundaries

< S > the
< S > quick
< S > brown
the quick
the brown
the fox
quick brown
quick fox
quick < /S > brown fox
brown < /S >
fox < /S >

To further illustrate, Gappy Bigrams of distance 1 are given in Table 2.4:

Table 2.4: Gappy Bigrams (of distance 1) of ”the quick brown fox” with sentence boundaries

< S > the
< S > quick
the quick
the brown
quick brown
quick fox
brown fox
brown < /S >
fox < /S >

The Gappy Bigram is able to preserve distinctive word groups for an author without the extremely low probability of occurrence. However, an author may distinctively used a two word group at exactly an interval of 3 words or 2 words or 1 word. That distinctiveness could be a key attribute for that grouping and is lost in Gappy Bigrams. To capture that distinctiveness, Orthogonal Sparse Bigrams are employed.

Orthogonal Sparse Bigrams

Orthogonal Sparse Bigrams (OSB) are similar to Gappy Bigrams in how there are constructed except that the distance between words in the OSB is included in the OSB. Just like N-Grams, Gappy Bigrams can extend beyond a sentence boundary, include punctuation, etc. For instance, in the phrase ”the quick brown fox” and a OSB distance of 2, the OSBs are given in Table 2.5:

Table 2.5: Orthogonal Sparse Bigrams (of distance 2) of ”the quick brown fox” with sentence boundaries

< S > the 0
< S > quick 1
< S > brown 2
the quick 0
the brown 1
the fox 2
quick brown 0
quick fox 1
quick < /S > 2
brown fox0
brown < /S > 1
fox < /S > 0

To further illustrate, OSBs of distance 1 are given in Table 2.6:

Table 2.6: Orthogonal Sparse Bigrams (of distance 1) of ”the quick brown fox” with sentence boundaries

< S > the 0
< S > quick 1
the quick 0
the brown 1
quick brown 0
quick fox 1
brown fox 0
brown < /S > 1
fox < /S > 0

It is important to note that in the cited references, the distance for OSBs is place between token 1 and token 2 instead of after token 1 and token 2 as shown in Tables 2.5 and 2.6. The distance is placed after the tokens in this thesis for more convenient parsing within reference files. Also, for OSBs, there is an issue of how to count OSBs. The two approaches are to strictly use on the distance that a token pair is found. In ”the quick brown fox”, the OSB of distance 2 of ”quick brown” has one instance, with a distance of 0. For the other approach, using lesser-included distances for OSB of distance 2 , ”quick brown” has three instances: quick brown 0, quick brown 1, and quick brown 2 because quick brown is a lesser included OSB of distance 2.

If file or database of OSBs is constructed, then a file or database of Gappy Bigrams also exists by default. The count of maximum distance OSBs equals the count of Gappy Bigrams, assuming the lesser included version of OSBs is used. This can be useful for conserving space in a system when both OSBs and Gappy Bigrams are needed.

2.4.2 Feature References

Once a scheme is determined for managing features, the features required must be selected. Feature selection is the process of deciding which features to include during classification. A set of features can be built from the training set, such as selecting N most used words in a training set. Features can be further refined by using outside references. For instance, a feature set could be built as the N most used words in a training set and filtered for stop words. In this case, stop words could be defined by other researchers work or some standard stop word set. Another option is to build all features from a reference set. This thesis made heavy use of the Google Web1T Corpus to act as a feature filter and a feature reference.

Google Web1T Corpus

The Google Web1T Corpus is a massive corpus of English language N-grams ranging from N=1 to N=5. The corpus was created from a snapshot of Google's search databases that took place during January 2006. The corpus consists of text files with the N-grams accompanied by a count of those N-grams. Each set of N-grams is stored in its own folder. The N-Grams are organized alphabetically by the first word in the N-Gram. For instance, "a cat" comes before "a dog" in the 2-Grams of the corpus.

The unique folder within the corpus is the 1-Gram folder. There are two files within the 1-Gram folder. One file is organized alphabetically like the rest of the corpus, but the other file is organize by count. The largest count comes first. This folder serves as both a 1-Gram source and an authoritative reference of all types within the Google Web1T corpus.

Punctuation is included in the corpus. Sentence boundaries are indicated by <S> and < \S >. To qualify for corpus inclusion, a 1-Gram needed to appear in the Google search databases at least 200 times. Additionally, to appear in a 2-Gram or greater, a gram had to appear in the database at lest 40 times. For 2-Grams and greater that appeared 40 times or more, but one of the words in the gram did not individually appear at least 200 times, the tag <UNK> is used to replace that word. The characters used in the corpus are UTF 8. Tokenization was "similar" to Penn Tree Bank except that hyphenated words were separated.[7] From working with the corpus, it becomes apparent that contraction within the corpus does not exactly match Penn Tree Bank. No "'t" contractions were kept intact during tokenization. The authors were contacted regarding this tokenization issue, to determine if this was intentional, but no reply has been received.

The Google Web1T is massive. This size makes Web1T both powerful to employ and cumbersome to use. The statistics for this corpus are listed in Table 2.7.

Table 2.7: Token and Type Counts in Google Web1T Corpus

Number of tokens:	1,024,908,267,229
Number of sentences:	95,119,665,584
Number of unigrams:	13,588,391
Number of bigrams:	314,843,401
Number of trigrams:	977,069,902
Number of fourgrams:	1,313,818,354
Number of fivegrams:	1,176,470,663

2.4.3 Feature Compression Techniques

Due to the large size of the corpora and feature reference used in this thesis, an efficient way to represent words and N-grams was needed. After surveying general literature on representing large data sets, the search for this thesis was narrowed. Two methods of efficiently representing large sets were investigated: bloom filters and minimal perfect hash functions. Minimal perfect hash functions were ultimately chosen as the tool for representing data in this thesis.

Bloom Filters

Representing a large dataset in a small memory space requires trading off between probability of a false positive, probability of a false negative, processing time, and size of representation. Bloom filters allow efficient storage of a list of values with zero probability of false negatives and a minimum probability of false positives. A Bloom filter consists of an array of m bits and k hash functions. Each hash function has an output range of 0 to $m - 1$. Each hash function must provide an equal probability distribution for each value 0 to $m - 1$. At the beginning of the construction of the Bloom filter, all m bits are set to 0. Each value to be a member of the Bloom filter is processed by each hash function. The output of the hash function corresponds to the array position of one of the m bits, which is then set to 1. If an output bit is already set to 1, that bit remains a 1. After all Bloom filter member values have been processed by the hash functions, the array of bits should be a mix of 0's and 1's.

To determine if a value belongs to the Bloom filter, that value is run through all k hash functions. If each array position output by the k hash function contains a bit set to 1, then the value probably belongs to the Bloom filter. If any of the m bits is a 0, that value does not belong.

There are variations on the Bloom filter that can use parallel architectures to advantage. For example, if the array of m bits is a multiple of k, then each hash function can have a range of 0 to $\frac{m}{k}$. Then each hash function can be run in parallel instead of in series. This scheme has no effect on the probability of a false positive, but can be appreciably faster to process in parallel processing platforms.

The work in a Bloom filter comes from determining the minimum values required for k and m to represent the expected set of values for a required false positive rate. The trade offs are, the larger the number of bits, the lower the probability of a false positive, but the larger the storage of the Bloom filter becomes. Likewise, an increased number of hash functions provides a lower probability of false positives, but larger numbers of hash functions increases

the computational cost of the Bloom filter. Given a required maximum false positive probability, p, and a maximum number of items, n, the minimum number of bits, m, is given by:

$$m = \frac{n \ln p}{(\ln 2)^2}. \tag{2.14}$$

Once the number of bits, m, is determined, the minimum number of hash functions, k, must be found. The required minimum of hash functions is given by:

$$k = \ln \frac{m}{n}. \tag{2.15}$$

Bloom filters are flexible and compressible. They are flexible because the number of bits, m, can be changed on the fly based on a changing number of items, n. Various compression techniques can be used to compress the bits, m, in the filters for transmission between computers. The filters can be processed in serial or parallel based on hardware architecture. Unfortunately, while flexible, Bloom filters are not as compact as their closely related cousin, the minimal perfect hash function.

Minimal Perfect Hash Functions

A minimal perfect hash fuctnion is the culmination of three concepts: a hash, a perfect hash, and a minimal hash. A hash function is a function that maps values from a set, U, with a number of values, k, to a range of values, m [8]. Hashes are normally associated with mapping a large universe to a small universe, but hashes can map between spaces of equal size. Hashes are often used in computer science for cryptography, efficiently mapping values, and myriad other tasks.

A hash function is a perfect hash function if there are no hashing collisions. A collision occurs when different values from U result in the same output value. More formally, in perfect hashes, there are m distinct values resulting from applying the hash function to all k values in U such that $k = m$. In short there must be a 1-1 mapping between each value in U to each resulting value in the range, m – no collisions to be handled (load factor $\alpha = 1$. A perfect hash function is called a k-perfect hash function if the ratio of possible values in the mapped space is not larger than k times the original space. This means the range, m, must be k time larger than U to ensure there are no collisions.

A perfect hash function is called a minimal perfect hash function if there are no ”blank” spaces in the hash table – meaning that no space is wasted in storing the hash. This is the same as a

k-perfect hash function where k=1. Less formally, the size of the range, m is equal to n, the size of the universe, U.

The time required to compute a value in m from a value in U is known as evaluation time. The required to construct the minimal perfect hash function is known as construction time. Along with representation space, evaluation time and construction time are the three performance parameters used to judge the efficiency of a minimal perfect hash function.

Minimal perfect hash functions (MPHF) are comprised of a set of hashing functions and a lookup data structure. The set of values (the universe, U)to be hashed must be known in advance. Those values are mapped, one-to-one to a unique range of numbers. At the end of the mapping, there is exactly one unique numerical hash for every provided input. The required number of bits for the hash is the minimum number of bits possible to uniquely identify all the items. The theoretical lower bound is 1.44n bits, where n is the number of elements in U.

A lower bound of 1.44n bits is the advantage of the MPHF, the data structure is extremely compact once created. The disadvantage is that any value submitted to the MPHF will result in a hash value. This requires a second discriminating function to determine member in the correct value set, such as a second, traditional, hash. This second hash undermines the compact size of the MPHF. However, combining a MPHF with a single traditional hash provides an extremely small probability of a false positive during a membership check and a fast lookup time.

In general, there are three stages of creating a minimal perfect hash function or any k-perfect hash function. These three stages are mapping, ordering, and searching. The mapping stage maps the set of keys in universe, U, to some other values. For example mapping a set of strings to an integer value or creating a set of vertices in a graph could serve as the mapping step. Ordering involves finding the buckets, vertices, etc that have been mapped with the most keys. These highly mapped entries become levels or child graphs in a further refined hashing scheme to develop into the final data structure. The final step, searching, involves assigning keys to positions within the mapping. The mapping is often multilevel allowing duplication from hashing to be ”backed off” and retried to continue building the hash.

There are many MPHF implementations available in the open source world. The implementation claiming to be the closest to the theoretical minimum for representation space is called the Compress, Hash, and Displace (CHD) algorithm[9]. CHD maps keys into buckets. Each bucket is assigned its own hash function, ϕ to create an index into the final data structures. The

buckets are ordered by magnitude (number of values in the bucket) for placement into the data structure. The theoretical lower bound of storage for a minimal perfect hash is 1.44n bits [8]. CHD's lower bound of storage is 2.07n to 3.56n bits depending on generation time allowed for the data structure.

2.5 Evaluation Criteria

Results from classifying data are computed from four basic categories of results: true positives, (tp), true negatives (tn), false positives (fp), and false negatives (fn). These four basic results are combined into accuracy, precision recall and F-Score for this thesis. While there are other evaluation criteria, these chosen criteria are clear enough and sufficient for measuring results.

2.5.1 Accuracy

Accuracy is a widely used and intuitive performance measure for classification. Accuracy, however, is flawed. Accuracy poorly represents the effectiveness of a classifier when the number of true negatives is large compared to the number of true positives. Missing all the true positives, but calling everything a negative, true or otherwise, yields a high accuracy without actually being effective at finding correctly labeled positives. Accuracy is defined as:

$$accuracy = \frac{tp + tn}{tp + fp + tn + fn} \tag{2.16}$$

[10]

2.5.2 Precision and Recall

Due to the weakness of accuracy as an evaluation criteria, precision and recall (also known as sensitivity) is used. Precision measures how often a document that belongs to the class being sought is actually labeled as that class. In other words, for all the actual documents written by the target author, how often are those documents labeled by the classifier as being written by the author. For all the documents said to be true by the classifier, what percentage are actually true.

$$precision = \frac{tp}{tp + fp} \tag{2.17}$$

Recall determines how well the classifier picks out true documents. In other words, for all the true documents in the set, how often does the classifier detect those true documents? Recall is given by:

$$recall = \frac{tp}{tp + fn} \tag{2.18}$$

[10]

2.5.3 F-Score

F-Score is the harmonic mean of precision and recall. It is a superior indicator to accuracy in evaluating a classifier. The definition of F-Score used in this thesis is:

$$F - Score = \frac{2}{\frac{1}{p} + \frac{1}{r}} \tag{2.19}$$

This definition is a variant of the standard definition of:

$$F - Score = \frac{(\beta^2 + 1) * 2pr}{\beta^2 * (p + r)} \tag{2.20}$$

The full definition of F-Score involves an additional term, β, which is a weighting value. A β value greater than one favors precision and a β value less than one favors recall. This thesis values precision and recall equally. This makes $\beta = 1$, thus the simpler equation for F-Score is used:

$$\frac{2pr}{p + r} = \frac{2}{\frac{1}{p} + \frac{1}{r}} \tag{2.21}$$

F-Score will be the primary evaluation criteria for this thesis.[10]

2.6 Mobile Device Platforms

There are numerous mobile device platforms ranging from the near ubiquitous mobile phones to tablets to personal digital assistants. Even within the category of mobile phones, there is a wide ranging array of capability and popularity. For newer mobile phones, capabilities often include access to storage, a network, phone services, GPS, and multimedia. Storage can be both onboard phone storage or removable storage such as a micro-SD card.

Often, there network access to more than just the mobile provider GSM or CDMA network. Modern phones often have WiFi access. GPS services provide position updates to the phone. Multimedia capability varies dependent on display size, resolution, battery consumption, processing speed, memory, and network availability. Mobile phones have not yet reached the level of commonality expected in desktop and laptop computing devices.

2.6.1 Mobile Devices by Popularity

To determine an effective development strategy for author detection on a mobile phone, it is sensible to determine what development language would support the largest number of mobile phones. By device popularity, the most dominant mobile operating systems, in order, are Symbian (Nokia phones), Research In Motion (Blackberry), iOS (Apple iPhone, iPad, iPod), and Android (Droid, Evo, Galaxy Tab). These four OS platforms constitute 88% of the mobile device market for first quarter of 2010.[11] Symbian, RIM, and Android all accept applications built on Java, or at least a variant of Java. Based on this vast market share, using Java as the development language for author detection on a mobile device has the largest potential for use.[12][13][14] Only iOS uses exclusively Objective C.[15]

2.6.2 Android Operating System

Based on its popularity and ease of installing test applications, Android is used as the development platform for this thesis. Android applications are not written, strictly speaking, in Java. Android applications are written in Dalvik which implements most of the syntax and structure of Java. Dalvik development is targeted at mimicking recent stable releases of the Java Development Kit (JDK). The core of the Android operating system is built on Linux, but is not built as a traditional Linux environment.[14]

Android applications consist of a combination of Activities, Services, Intents, and Content Providers. Activities are processes that users can see and interact with. Activities create the windows, tabs, and dialogs for user interaction.

Services run in the background with no user graphical user interface (GUI). Android Services are not equivalent to traditional Unix services. Unix services are, by nature, persistent process within the operating system. Android Services are just as prone to being killed by the operating system as an Activity.

Intents are messages passed around by processes and Java Virtual Machines within the Android operating System. Typical Intents are created by Content Providers for actions such as incoming calls, incoming Short Messaging Service (SMS) messages, GPS, etc. Other typical Intents are passed between Activities in an application or between Services and Activities in an application. Intents can start, stop, and pause Activities as well as just pass along data such as a String or integer. Applications use Activities, Services, and Intents in combination to provide functionality on an Android Mobile device.

The lifecycle of an application in Android varies from a standard PC application lifecycle. Activities and Services continue to run in Android while sufficient resources remain on the mobile device. When resources become exhausted, the Android operating system will shut down Activities and Services it deems as less important or less used. This is why Android applications often lack a ”Quit” or ”Exit” function in their menus – developers expect that the application can continue to run so long as the operating system has sufficient resources. Contents providers, on the other hand, are persistent processes driven by items such as GPS receivers, mobile networks, and WiFi networks. Content providers are accessed and listened to by applications. A Content Provider can also be built by a developer to act as a data provider for other application as an abstraction instead of an actual physical device like GPS or WiFi.[16]

2.7 Corpora

A major portion of validating a method of author attribution is securing a corpus of usable data. There are some tried and true corpora openly available, such as the ENRON Email Corpus, which are well know, well studied, and useful for comparison. With a focus on mobile devices, this thesis needed a more short text relevant corpus. For this need an in-house corpus of Twitter posts, known as Tweets, was used. Using these two corpora provides a standard corpus to judge effectiveness and a newer corpus to anticipate future capability in the evolving medium of mobile computing.

2.7.1 ENRON Email Corpus

The ENRON email corpus is a set of emails collected by the Cognitive Assistant that Learns and Organizes (CALO) Project. The original corpus contains 619,446 emails from 158 users. These emails were posted on the web by the Federal Energy regulatory Commission during the investigation of ENRON. Issues with the raw posting were corrected by several people at MIT and SRI to arrive at the form of the current corpus. The emails are organized in folders, by user. The folder organization used by the original user is kept mostly intact (Inbox, Sent Items, etc) except for some computer generated folders that were seldom used by the actual users. Each email is contained in its own text file. Each text file contains the full email header as well as any threaded conversation headers (replies and forwards).[17]

The ENRON corpus is a frequent target for natural language processing. Author detection performance for character and word N-grams, SVM, Naive Bayes and other classifiers on the ENRON corpus is well documented. For this reason, all methods used in this thesis were

attempted on the ENRON email corpus as a benchmark of performance, before moving on to the more mobile-centric corpus of Twitter.

2.7.2 Twitter

Twitter is a short message micro-blogging services that users can access from traditional computers as well as mobile devices. Originally designed for use over Short Message Service (SMS), Tweets (vernacular for message sent on Twitter) are limited to 140 characters. Unlike other social networking sites, Twitter has no requirement for users to post their real names. Author detection on a corpus of Tweets will be challenged by the short duration of each Tweet (Tweets would constitute a document in this case) and the non-standard use of language. Also, users do not have to formulate original content for their Tweets. Just like as email forward, users can re-Tweet a Tweet they have already received.

Tweets are formatted for use with a JavaScript Object Notation (JSON) format. The JSON formatting provides numerous fields containing language, Twitter id, geocode (latitude and longitude of sender). The Twitter API contains both streaming and RESTful methods. Using the Twitter API, Tweets can be pulled from the TwitterSphere using a free, rate limited service called Garden Hose or via a fee-based, rate unlimited service called Fire Hose. The rate limit for Garden Hose is 150 messages per hour. Those messages are randomly chosen from Twitter accounts that make themselves viewable by the public. The Twitter API allows for filters to affect the stream of Tweets to avoid getting Tweets that do not meet your needs and would otherwise impact your rate limit. The length limitation and mobile nature of Tweeting, makes Twitter a reasonable model of SMS behavior for testing purposes.[18]

2.8 Recent Work in Author Detection, Google Web1T, and Mobile Devices

:NOTE: I just found a slew of related work that is worth studying some more, but I don't want to hold up the process of getting feedback on the bulk of this chapter 2. The below incomplete sentences are placeholders for me on what I have for related work right now. There is a patent on author detection based on a compressed hash, there are articles solely on managing and querying the Web1T, and a paper on chunking data on memory constrained systems. :ENDNOTE:

Google Web1T has been used as a smoothing reference in other machine learning studies. XXXXXX used a backoff method based on Google Web1T XXXX counts to ..... Google

Web1T has also been a reference for spelling correction[] and semantic classification[]. There has even been a paper on just managing the Google Web1T corpus effectively.

Author detection across varied information sources using a normalized compressor distance has been patented. This method creates a bitwise compression of content from web pages, emails, texts, or any electronic document and uses clustering, based on this patented distance measure, to arrive at probability of various documents being from the same author.[**?**] Author detection on mobile devices has not shown up in patent or paper searches. There are author detection papers that reference the prevalence of text messages in author attribution, but none on using the mobile platform itself to conduct processing. Despite a breathtaking pace of application development on mobile device platforms such as iPhone and Android, using mobile computing capability for traditional machine learning has appears to be a wide open question.

Recent SVM work has included....

2.9 Conclusion

There is a rich body of work on author attribution, SVM, Naive Bayes, and on the ENRON Email corpus. Applying traditional document and email author attribution tools to the short message environment of mobile phones is an area ripe for exploration.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3:
Experimental Design

This chapter document the concepts and technical approaches used in this thesis, as well as procedural concepts for understanding the experiments of this thesis.

3.1 Experimental Design Overview

Thesis Goals The central goal of this thesis's experiments is to compare size and speed of different author detection methods against the effectiveness of those same author detection methods on a resource constrained device such as a mobile phone. Size and speed are critical to this thesis. This is due to the restrictive nature of mobile phones. However, the nature of these experiments allows the results to be applied to other computing platforms with limited resources such as nano-computers, mobile sensors, or yet unimagined devices.

Experimentation Phases To achieve the thesis goal, experimentation will be conducted in two phases: parameter evaluation and mobile phone performance evaluation. In parameter evaluation, the effectiveness of different combinations of classification methods, features sets, group sizes, and smoothing/filtering to compare prediction performance against model size and processing requirements. During the mobile phone performance evaluation, the combinations of classification methods that are both feasible and effective are used on mobile phones to determine the overall performance and impact of running author detection on an actual mobile phone.

3.2 Phase One: Parameter Evaluation

This phase will evaluate numerous combinations of two classification methods, five feature sets, six grouping sizes, three grouping methods, and two corpora to determine the computing requirements and effectiveness of these combinations. Preparing for these evaluations takes several steps including determining the required combinations, organizing and compressing the feature references, preparing the training and prediction data, building the models, and, finally, running the prediction tests. The results for all prediction test will be stored in a mySQL database which will also store the resulting f-score, precision, recall, and size of model for each test.

3.2.1 Creating the Testing Combinations

The classification methods to be compared are Naive Bayes and Support Vector machines (SVM). Naive Bayes is fast and uses a relatively small amount of RAM and disk storage. SVMs, are slower, use greater RAM and disk storage, but often yield higher f-scores. There are numerous feature sets that can be chosen. For this thesis, 1-grams, 2-grams, 5-grams, gappy bigrams, and orthogonal Sparse bigrams will be examined. The intuition is that 1-grams are simple and use less space, but will be less effective than bigger feature sets such as gappy bigrams or 5-grams.

For this thesis, two feature reference sets will be examined, a bootstrapped bag of words and the Google Web1T corpus. Bootstrapped bag of words simply means finding all the unique types within a training set and making each type a feature in the feature set. Since the Google Web1T corpus is huge, a parameter of that feature reference which can be adjusted is the percentage of a given feature set that might be used. These experiments will permute through these numerous options to determine size, speed, precision and f-score. The end result will be an analysis of the utility of these various approaches to author detection on a mobile phone. A graphic of the parameter combinations is given in Figure 3.1.

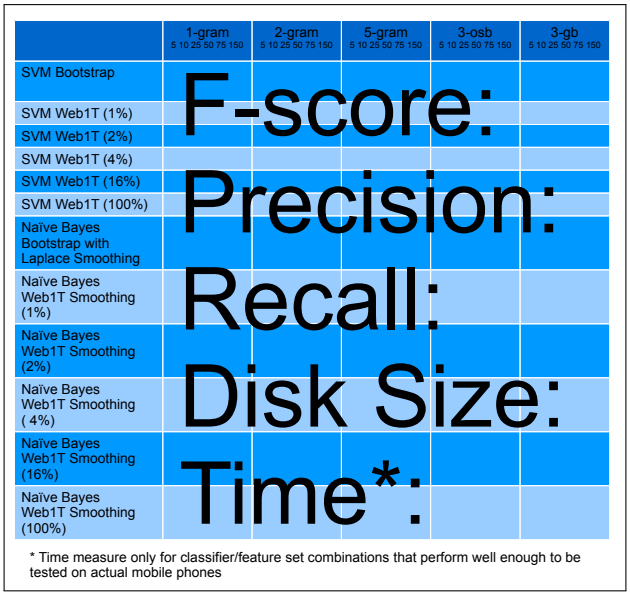

Figure 3.1: Parameter Combinations for Testing

The small numbers ”5 10 25 50 75 150” given under each column heading in Figure 3.1 indicate that all authors will be tested in groups of 5, 10, 25, 50, 75, and 150 using three different

grouping strategies: small-to-large, small-and-large, and random. In small-to-large, the authors with the smallest amount of training data are grouped together. In small-and-large, small authors and large authors are paired together. In the random grouping, the authors are grouped together by a pseudo-random selection. The reasoning for these three grouping strategies is to provide insight into the effect of prolific authors versus less prolific authors. If results are similar for the same author for each group, the prolific writing may not impact the outcome of author detection with these methods. This is needed information to rule out that the test author detection methods simply select the most prolific author instead of the actual author.

3.2.2 Organizing and Compressing Feature References

A key element to this testing is the use of the Google Web1T corpus. The Web1T corpus contains billions of types with a token mass of just over 1 trillion. The size and breadth of the Web1T corpus makes it appealing as a source for smoothing in Naive Bayes and a tool for creating models in SVM. However, due to the huge size of the Web1T corpus, the text files comprising the corpus must be compressed and managed for use on desktop workstations, servers, and especially mobile devices. Managing the corpus requires determining what portions of the Web1T corpus will be used. Using the choice of 5-grams as an example for illustration purposes, suppose only the 5-grams portion of the Web1T corpus might be used. The 5-grams constitutes 118 text files containing up to 10 million lines of text each. Each line of the Web1T 5-gram files contains space separated words (making up the type) followed by a count, separated from the words by a tab. The lines of text are organized alphabetically by token where uppercase letters are distinct from lowercase letters. Even using only one size of gram from Web1T, a reverence of this size is slow and bulky for machine learning use. Therefore, a subset of the reference is needed.

Sizing the Feature Reference Set To manage the size of Web1T, a small portion of the 5-grams could be chosen – 1%, 2%, 4%, etc. To choose which part of the reference to use (largest, smallest, random) this thesis takes advantage of Zipf's Law. Zipf's law states that the highest frequency word occurs approximately twice as often as the next most frequent word. By that reasoning, a list of the types with the highest counts is needed to capture the largest use of words in a natural language corpus. To get this count ordered list, the complete set of Web1T n-grams are recreated. The recreated files list each type organized by count instead of alphabetically. If two or more types have the same count, then those types are list alphabetically. The types are still listed first as a group of space separated words followed by a tab and ended with a count.

Three Tiered Hashing Scheme Even once the feature set of types to be used for classification has been determined, the smaller set of text is still too slow to process and very bulky to store. To further compress the data, a three tiered hashing scheme is used. The structure of the three tiered hashing scheme is shown in Figure 3.2.

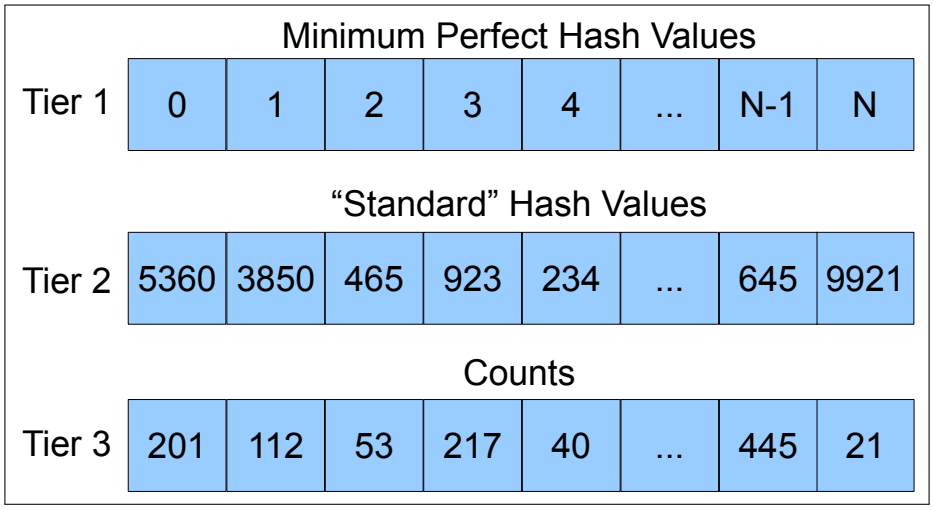

Figure 3.2: Three Tiered Hashing Scheme Structure

The first tier is comprised of minimal perfect hash (MPH) values of the selected feature set. The second tier of the scheme is comprised of a 64 bit hash of the original type. This second tier's job is to reduce the probability of a false positive in the fist tier. This issue arises because no matter what string is input to the MPH function, a valid MPH value will be produced. The second tier's traditional hash is accessed by mapping the MPH value to the index of an array that comprises the second tier. That array cell contains the 64 bit hash of the original text used to create the MPH value. This make the false positive rate for a given type $\frac{1}{2^{64}} * \frac{1}{\text{range of MPH values}}$ which is deemed an appropriate risk of collision in this hashing scheme. The third tier is simply an array of long values. The MPH value from tier 1 is used to access this array which hold the count value for a given type. An example of converting a phrase, ”the quick brown”, is shown in Figure 3.3.

These different tiers are not contained in a single data structures. The MPH data structure, tier 1, is contained in a file called ”keys.mph”. The array of hash values, tier 2, is contained in a file called ”signature”. The counts are contained in a Java object file call LongCountsArrayFile. The Naive Bayes experiments use all three tiers of this structure for smoothing values. The SVM experiments only use tier 1 and tier 2 to verify that a string encountered actually belongs to the

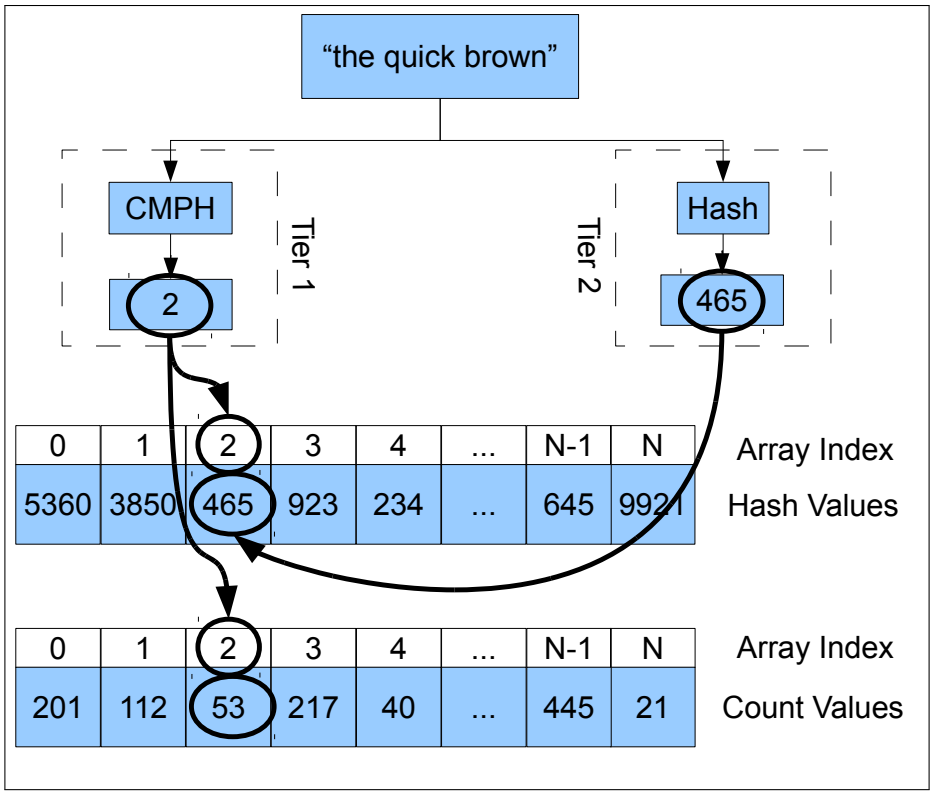

Figure 3.3: Three Tiered Hashing Scheme Example

feature set. These hefty data files comprise the bulk of storage required on the mobile device. Since these data files get loaded into RAM during the prediction process, the file sizes also impact RAM requirements. The impact on RAM and disk storage makes management of the size of keys.mph, signature, and LongCountsArrayFile an important aspect of the experiments.

Choosing Artifacts for the Three Tiered Hashing Scheme One impact of using MPH to reduce the size of storing types is a loss of flexibility with the text artifact selection process. Before the MPH data structure is created, the creator must determine if punctuation, capitalization, sentence boundaries, or ”unknown” words will be allowed. The omission of each of these artifact types brings its own unique challenges. A binary style number scheme was adopted for each of these features where capital letters hold the 1 position, punctuation the 2 position, unknown word tags the 4 position, and sentence boundaries hold the 8 position. The complete matrix of artifacts allowed in the MPH model is included in Figure 3.4.

MPH Label	Remove Sentence Boundary Tags	Remove Unknown Word Tags	Remove Punctuation	Remove Capital Letters
0	FALSE	FALSE	FALSE	FALSE
1	FALSE	FALSE	FALSE	TRUE
2	FALSE	FALSE	TRUE	FALSE
3	FALSE	FALSE	TRUE	TRUE
4	FALSE	TRUE	FALSE	FALSE
5	FALSE	TRUE	FALSE	TRUE
6	FALSE	TRUE	TRUE	FALSE
7	FALSE	TRUE	TRUE	TRUE
8	TRUE	FALSE	FALSE	FALSE
9	TRUE	FALSE	FALSE	TRUE
10	TRUE	FALSE	TRUE	FALSE
11	TRUE	FALSE	TRUE	TRUE
12	TRUE	TRUE	FALSE	FALSE
13	TRUE	TRUE	FALSE	TRUE
14	TRUE	TRUE	TRUE	FALSE
15	TRUE	TRUE	TRUE	TRUE

Figure 3.4: Matrix of CMPH Models by Artifacts Included

Omitting Punctuation Omitting punctuation provides two options for dealing with the corpus: replace punctuation with "$<$ UNK $>$" or drop the punctuation altogether. If punctuation is dropped, then any type containing a punctuation mark in the feature reference set must be completely ignored. If the punctuation is replaced with $<$ UNK $>$, then a search within the existing count structure must be conducted for a corresponding entry for $<$ UNK $>$ and any non-punctuation words in the type. While dropping punctuation is much simpler to implement than employing "$<$ UNK $>$" tags, however, Google did count punctuation as a word in type construction, so correlation between n-gram counts in the Web1T corpus and the trained/predicted documents is slightly affected. To maintain simplicity, the simple drop approach was used in these experiments.

Omitting Capitalization Omitting capitalization is straightforward for construction of tier 1 and tier 2, the inputted text for the type is converted to all lower case and a check is conducted to see if that type is already in the MPH data structure. For tier 3, which contains the counts, the lower case versions of the word must have its count mass added with its corresponding uppercase types. This adds complexity to the insertion process for MPH but is easily managed. Another option would be to simply drop all types that contained capitalization, but that would remove a large count mass from the Web1T corpus. Adding counts was the method used in this thesis to deal with omitting capitalization.

Omitting Sentence Boundaries Sentence boundaries are denoted in the Web1T corpus as $<$ S $>$ and $<$ \S $>$. Dropping sentence boundaries is straightforward since there is no replacement or count mass issues to deal with. Since the tools for locating sentence boundaries make use of their own machine learning processes, no sentence boundaries were used in these experiments.

Omitting Unknown Words In the Web1T corpus, "unknown" words have a specific meaning. To be included in any corpus n-gram set, a word must have appeared as a 1-gram at least 200 times in the Google database. By contrast, to be 2-gram, 3-gram, 4-gram, or 5-gram, that gram had to appear at least 40 times in the Google database. This created as situation where a word would need to appear in a 2-or-higher-gram, but was not allowed into the corpus because it did not appear 200 times in the overall database. Words that fall into that category are replaced with the tag $<$ UNK $>$ in the Web1T corpus. Removing $<$ UNK $>$ words from the MPH has no effect on the counts in tier3 and is a straightforward process.

Choosing N-Grams N-grams can be as small as a 1-gram and grow, theoretically, to any size N imaginable. The preferred reference set for this thesis, the Web1T corpus, uses 1, 2, 3, 4, and 5-grams. While it is tempting to test all 5 N-gram sizes available in the corpus, only three were used. 1-grams and 5-grams were chosen to represent opposite ends of the size N gram spectrum available. 2-grams were used as a strong comparison to gappy bigrams and orthogonal sparse bigrams discussed below. Future work could focus on 3 and 4-grams to determine if there is a performance to size advantage in using those size of N-grams.

Gappy Bigram and Orthogonal Sparse Bigram Construction Once the 3 tier structure is created and functional, there are still two type of features remaining to be created. The Web1T corpus only contains standard n-grams, not gappy bigrams or orthogonal sparse bigrams. To

create these more exotic types of bigrams, a rule for counting distance and a notation scheme was needed. It was decided to use ”lesser included counts” for both the gappy bigrams and the orthogonal sparse bigrams. This means that a word1 word2 pair would count for osb-0, osb-1, osb2, etc. While previous papers placed the distance for an OSB between word1 and word2 [19], this thesis constructed the OSBs with the distance after word2 for easier parsing. The gappy bigrams and OSBs were constructed from the 2, 3, 4, and 5-grams in the Web1T Corpus. Word pairs from a distance of 0 (a traditional bigram or an OSB-0) to a distance of 3 (an OSB-3 or the first and last word in a 5-gram) were built from the Web1T corpus. This process only looks at the first and last words in a 3-gram, 4-gram, or 5-gram since the inner words of this gram are already captured in the 2-gram. Using the inner 2-grams would double count 2-grams and throw off the count mass. The same is true for 3-grams inside of 4 and 5-grams as well as 4-grams inside of 5-grams.

Grouping By Size With references built and sized, an efficient structuring of the authors and documents needs to be devised. During data file construction, the grouping and conversion processes happened simultaneously. The grouping sets built were: small-to-large, small-and-large, and random.

Small-To-Large The small-to-large group matched the least prolific authors together with increasing size up to the most prolific authors. For example, of the 5 authors in the ENRON corpus with 5 total kilobytes worth of text are group together while the 5 authors with greater than 1 total megabyte of text are group together. No author is picked more than once. An example is shown in Figure 3.5.

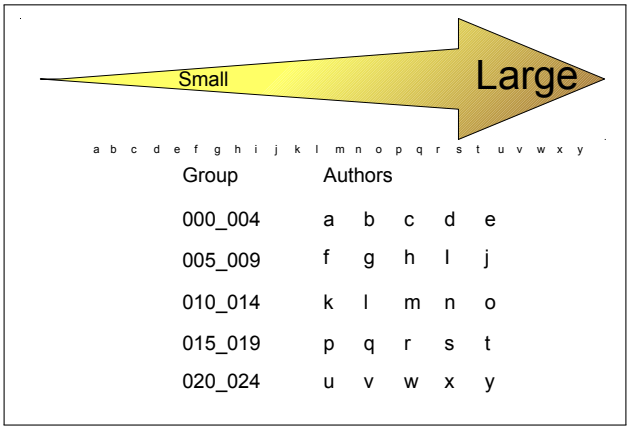

Figure 3.5: Small-To-Large Group for Group Size 5, 25 Authors

Small-And-Large The next group, small-and-large, is created by binning the authors by size. Then one author from each bin is picked to be group with one author from each other bin. For example the least prolific author is paired with one author from the most prolific bin and one author from each bin in between. In this situation, the selection from each bin is not random. The least prolific remaining author from each bin is picked for grouping. No author is picked more than once. An example is shown in Figure 3.6.

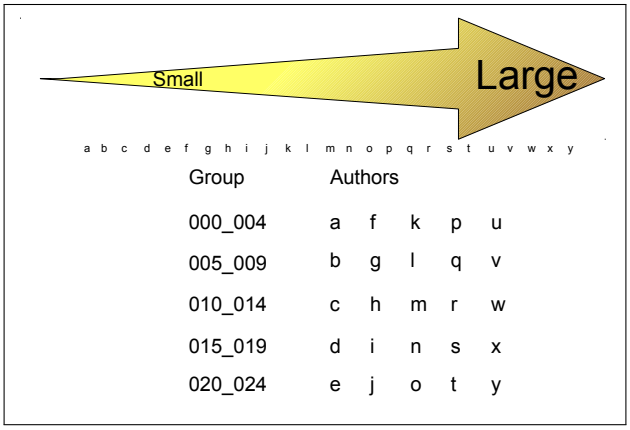

Figure 3.6: Small-And-Large Group for Group Size 5, 25 Authors

Random This grouping simply produces a random number in the range of available authors and places the selected author into a group until that group is full. Then the next group is filled the same way until no authors remain. No author is picked more than once. No author is picked more than once. An example is shown in Figure 3.7.

Group Sizes Based on having 150 authors in the ENRON Corpus, the six following group sizes were used: 5, 10, 25, 50, 75 150. These six group sizes coupled with the three grouping types, small-to-large, small-and-large, and random creates 18 grouping types. Examples of these grouping types are 5 small-to-large, 5 small-and-large, 5 random, 10 small-to-large, ..., 150 small-to-large, 150 random. Although using all 150 authors in a grouping set makes the procedure of how the 150 were grouped redundant, all three size 150 tests were conducted as a check on the experiments. If the 150 author grouping provides different reslts, then there may be an issue with the classifiers.

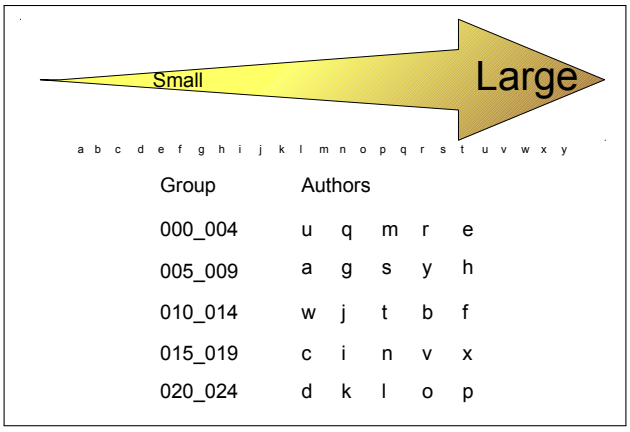

Figure 3.7: Random Group for Group Size 5, 25 Authors

After these grouping types were constructed, there were 171 totals sets (30 sets of 5 small-to-large, 15 sets of 10 small-to-large, ..., 1 set of 150 small-to-large, 1 set of 150 random.) Each of these sets were intended to be run through Bootstrapped SVM, Web1T SVM, Laplace Smoothed Naive Bayes, and Web1T Smoothed Naive Bayes. Assuming that only one MPH model is chosen to represent Google Web1T, that results in 684 experiments. Since there are 16 different MPH models based on the combinations punctuation, capitalization, sentence boundaries, and unknown words, the number of experiments could rise drastically. However, only two MPH models will be used during the experiments resulting in only 1,368 per feature type. Using 1-grams, 2-grams, 5-grams, 3-gb, and 3-osb results in 6,840 totals experiments.

Data File Format With combinations of features, artifacts, and group sizes chosen and the MPH data structures created, the actual documents must be converted into a format that can be used by the classifiers. The LibSVM file format was used since that it is the native format for LibLinear, the tool used for SVM in this thesis. The Naive Bayes classifier was built specifically for this thesis and was designed to use LibSVM format for convenience. The format of the data files consisted of an integer representing the author followed by a space, followed by a number representing the MPH value, followed by a colon, followed by another number representing the count. Each succeeding instance of a MPH value coupled with a count is separated by a space. Each document in the corpus is represented by a single line. Each line's mph number is in increasing order from left to right. The data files store the word/count pairs in a sparse fashion. This means that a zero count is not included in the data file. Absence of a word/count pair constitutes a zero count without needlessly using up space in the file. An example of this file format is provide in Figure 3.8.

```
83 362112:1 2216672:1 4609969:1 5582887:1 6141348:1 13588391:0
115 2334923:1 4077269:1 4759253:1 10878308:1 13069356:1 13588391:0
47 902626:1 1820755:1 10686459:1 12596717:1 13588391:0
80 1648944:1 1979998:1 2205090:1 2334923:1 2478205:2 13588391:0
```

Figure 3.8: LibSVM File Format

Running SVM With the data files created, the classifiers can be applied. The chosen tool for author detection using SVM is LibLinear. LibLinear was chosen for its speed compared to LibSVM. The LibLinear source code was slightly modified to allow training a model from a data set, then running prediction on a separate set without using the built-in cross validation function. During the training phase, each author has a SVM model built for it from a training file in a directory labeled ”train”. During the prediction phase, document contained in another file are used to predict the mostly likely author. That file is contained in a folder called ”predict”. The SVM author result is printed to a result file in a directory labeled ”result”. The f-score, precision, and recall for each file is recorded in a file inside a folder labeled ”analysis”. The analysis file also contains a full confusion matrix, time of prediction, size of original file, and other statistics. This file is finally pulled into a mySQL database for storage and calculation of precision, recall, and f-score..

The size of the author models impacts RAM usage and disk space. LibLinear stores SVM models as an array. RAM and storage are not the only limits. An array of integers representing token counts can be sizable, especially when token counts are long numbers (64 bits) instead of integers (32 bits).

RAM and disk storage are not the only limits. By specification, arrays in Java are limited to $2^{31} - 1$ entries. This means the model cannot contain more than $2^{31} - 1$ features. Also, the model must be loaded into RAM, so the number of authors coupled with the size of the author model must be weighed against the available RAM and disk storage.

Running Naive Bayes The Naive Bayes classifier has been specifically built for this thesis. The classifier reads in a pre-built array of long values from a file. The two types of arrays are a Laplace Smoothing array, which is comprised of all 1’s. the second type of array is the Google Smoothing array comprised of the count values from the Web1T corpus. Using an array to hold the smoothing values for Naive Bayes has an impact on RAM usage. There must

be enough available RAM to hold the smoothing array. To prevent having numerous copies of the smoothing array in memory (one for each author being trained) a hashmap is used to create the author models instead. The process for training put each encountered feature type into a hashmap along with a count of $1plus$the array smoothing value. If that feature type is encountered, the the count is simply incremented. Once all the training documents have been read and counted, the hashmaps of feature types and counts is converted into a hashmap of feature types and log of probability.

During the prediction process, each encountered feature type is queried against the author hashmap first. If the feature type is found in the hashmap, then the hashmap $\log probability$ is used. If not, then the smoothing array containing log of probabilities is used. An example of this hashmap/array process is shown in Figure 3.9. The result of the prediction process is outputted to a file in the corresponding results directory. Those results are then processed into a file in the corresponding analysis folder where all data is then read into a mySQL database for evaluation of precision, recall, and f-score.

3.3 Phase Two: Android Implementation

To manage files on the mobile device, a rudimentary file manager was built with a text viewer added. A button was also added to the File Manager to execute prediction against a document on the phone. An Android Service was also constructed that listens for incoming SMS messages. When an SMS Message is ”heard”, it is processed for author detection. The Service can be turned on and off using a button on the File Manager.

To measure CPU and RAM impact caused by the author detection processing, the third party applications, and Memory Usage, was installed on the phones. The method is to take a baseline of the phone’s CPU and RAM usage with no Widgets or Applications running, the phone is attached to a recharging device, and no calls or texts are being sent. The same phone conditions are being set for the processing tests where the only application that will run on the phone will be the SMS capture and author detection application for this thesis. This will yield some basic metrics of author detection impact on the phone’s capabilities.

3.4 Corpora

Two corpora are used for this thesis: the ENRON Email Corpus and the Naval Postgraduate School (NPS) Twitter Corpus. The aim of this thesis is to examine author detection using a mobile device. Two of the most common text communications on a mobile device are email

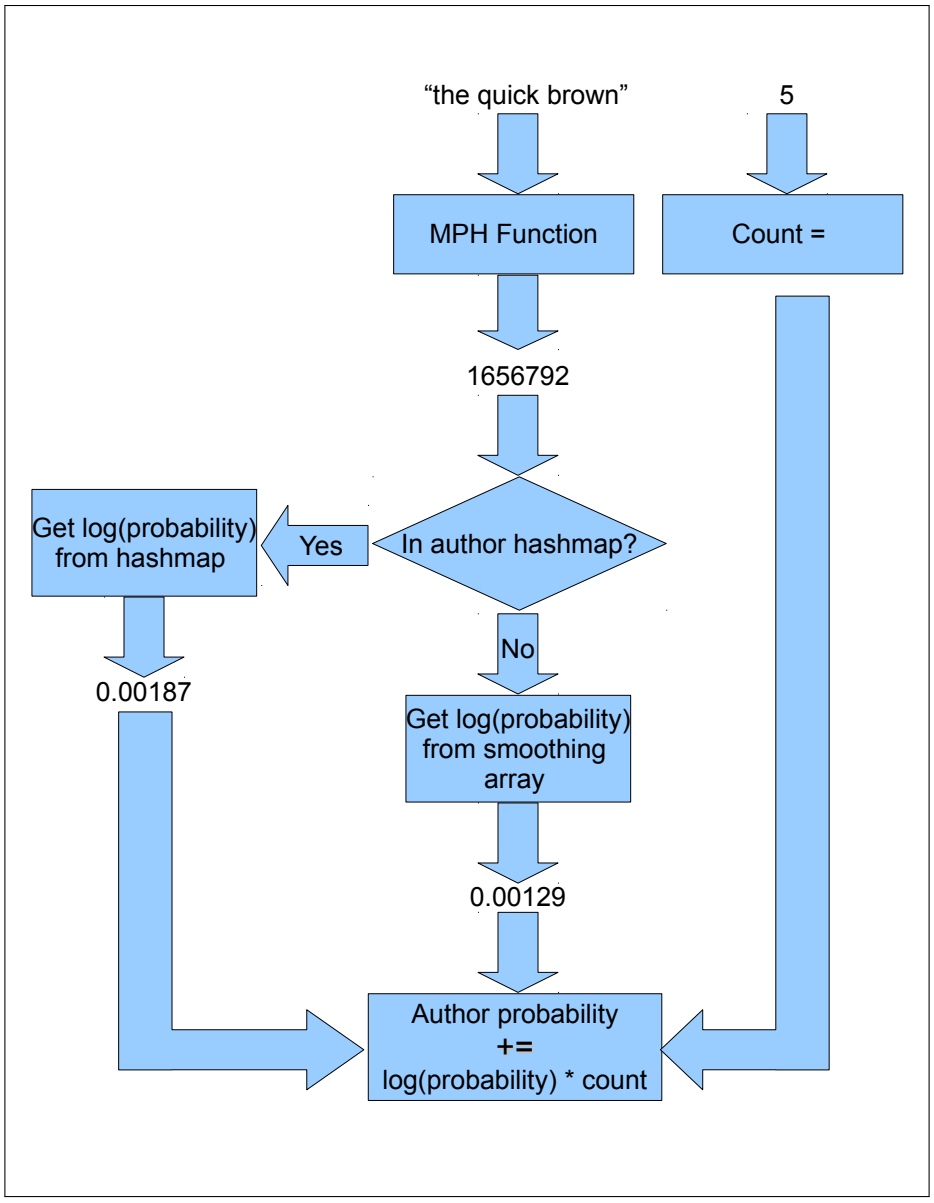

Figure 3.9: Naive Bayes Hashmap and Smoothing Array Flow Chart

and SMS (texting). The ENRON Email Corpus has been widely examined and has been used to author attribution in other studies. This makes the ENRON Corpus a suitable standard to measure the author detection techniques used in this thesis. The NPS Twitter Corpus is smaller and newer than the ENRON email corpus, but texting is extremely popular as a communications medium. Determining the effectiveness of author detection over this rapidly expanding text standard is important for analyzing the effectiveness of author detection on mobile devices.

ENRON Email Corpus Each ENRON email was stored in a single text file within a folder labeled with the author's first initial, second initial, and last name. Prior to processing each ENRON email, a systematic attempt was made to distill each email down into just the author's words. To support this distillation, the email header was stripped from each email. A search was conducted throughout the remaining text to find additional email headers. These are the embedded headers caused by email replies and forwards. Also to prevent biasing the author attribution, an attempt was made to systematically detect an email closing such as "Sincerely, Dave" or "Yours Truly, Jane".

Naval Postgraduate School Twitter Short Message Corpus All tweets from a single author were stored in a single text file. Each tweet from that author was contained on its own line. Each line begins with a date-time stamp with the content of the text following. Prior to constructing the corpus, all "re-tweets" were removed to ensure the text came from a single author, not just from a single Twitter account.

3.5 Intended Comparison

Once all tests are complete, performance of the different combinations of feature and classifiers will be compared for both the ENRON email corpus and the Twitter Corpus. This is to allow any differences in performance against the two primary media used on mobile phones. The completed test results should provide insight into the possibility of author detection on a mobile phone against both email and short messages.

CHAPTER 4:
Results and Analysis

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5:
Conclusions and Future Work

THIS PAGE INTENTIONALLY LEFT BLANK

REFERENCES

[1] Harold Love. *Attributing authorship: an introduction*. Cambridge University Press, June 2002. ISBN 9780521789486.

[2] Ethem Alpaydin. *Introduction to machine learning*. MIT Press, October 2004. ISBN 9780262012119.

[3] Daniel Jurafsky and James H. Martin. *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition*. Prentice Hall, 2009. ISBN 9780131873216.

[4] Vladimir Vapnik and Corinna Cortes. Support-Vector networks. *Machine Learning*, 20:273–297, 1995. ISSN 0885-6125. http://dx.doi.org/10.1023/A:1022627411411. 10.1023/A:1022627411411.

[5] RA Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7 (2):179–188, 1936.

[6] Multiclass SVMs. http://nlp.stanford.edu/IR-book/html/htmledition/multiclass-svms-1.html. http://nlp.stanford.edu/IR-book/html/htmledition/multiclass-svms-1.html.

[7] Thorsten Brants and Alex Franz. Web 1T 5-gram Version 1, 2006, Linguistic Data Consortium, Philadelphia.

[8] Djamal Belazzougui, Fabiano Botelho, and Martin Dietzfelbinger. Hash, displace, and compress. In *Algorithms - ESA 2009*, pp. 682–693. 2009. http://dx.doi.org/10.1007/978-3-642-04128-0_61.

[9] CMPH - c minimal perfect hashing library. http://cmph.sourceforge.net/. http://cmph.sourceforge.net/.

[10] M. Sokolova, N. Japkowicz, and S. Szpakowicz. Beyond accuracy, f-score and roc: a family of discriminant measures for performance evaluation. *AI 2006: Advances in Artificial Intelligence*, p. 10151021, 2006.

[11] Gartner says worldwide mobile phone sales grew 17 per cent in first quarter 2010. http://www.gartner.com/it/page.jsp?id=1372013. http://www.gartner.com/it/page.jsp?id=1372013.

[12] BlackBerry - BlackBerry developer zone. http://us.blackberry.com/developers/. http://us.blackberry.com/developers/.

[13] Symbian SDKs. http://www.forum.nokia.com/info/sw.nokia.com/id/ec866fab-4b76-49f6-b5a5-af0631419e9c/S60_All_in_One_SDKs.html. http://www.forum.nokia.com/info/sw.nokia.com/id/ec866fab-4b76-49f6-b5a5-af0631419e9c/S60_All_in_One_SDKs.html.

[14] Mark Lawrence Murphy. *Android Beyond Java*. CommonsWare, LLC, September 2010. ISBN 9780981678047.

[15] Creating an iPhone application. http://developer.apple.com/library/ios/#referencelibrary/GettingStarted/Cre an_iPhone_App/index.html. http://developer.apple.com/library/ios/#referencelibrary/GettingStarted/Creating_an_iPhone_App/index.html.

[16] Mark L. Murphy. *The Busy Coder's Guide to Android Development*. CommonsWare, October 2010. ISBN 9780981678009.

[17] Enron email dataset. http://www-2.cs.cmu.edu/%7Eenron/. http://www-2.cs.cmu.edu/%7Eenron/.

[18] Streaming API documentation | dev.twitter.com. http://dev.twitter.com/pages/streaming_api. http://dev.twitter.com/pages/streaming_api.

[19] Daniel M Bikel and Jeffrey Sorensen. If we want your opinion. In *Proceedings of the International Conference on Semantic Computing*, p. 493500. IEEE Computer Society, Washington, DC, USA, 2007. ISBN 0-7695-2997-6. http://portal.acm.org/citation.cfm?id=1304608.1306375. ACM ID: 1306375.

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California