Feature Selection for Text Classification

George Forman
Information Services and Process Innovation Laboratory
HP Laboratories Palo Alto

HPL-2007-16(R.1)
May 3, 2007*

text classification,
document
categorization,
unstructured data,
feature generation,
feature selection,
overview

Chapter: Feature Selection for Text Classification
Book: Computational Methods of Feature Selection
Chapman and Hall/CRC Press, 2007

George Forman
Hewlett-Packard Labs

April 26, 2007

Contents

1 Introduction

Applications of text classification technology are becoming widespread. In the defense against spam email, suspect messages are flagged as potential spam and set aside to facilitate batch deletion. News articles are automatically sorted into topic channels, and conditionally routed to individuals based on learned profiles of user interest. In content management, documents are categorized into multi-faceted topic hierarchies for easier searching and browsing. Shopping and auction web sites do the same with short textual item descriptions. In customer support, the text notes of call logs are categorized with respect to known issues in order to quantify trends over time. These are but a few examples of how text classification is finding its way into applications. Readers are referred to the excellent survey by Sebastiani [14].

All these applications are enabled by standard machine learning algorithms—such as Support Vector Machines (SVMs) and Naïve Bayes variants[12]—coupled with a pre-processing step that transforms the text string representation into a numeric feature vector. By far, the most common transformation is the 'bag of words,' in which each column of a case's feature vector corresponds to the number of times it contains a specific word of the training corpus. Strikingly, although this representation is oblivious to the order of the words in the document, it achieves satisfactory accuracy in most topic-classification applications. For intuition behind this: if the word 'viagra' appears anywhere in an email message, regardless of its position, the probability that it is spam is much greater than if it had not appeared at all.

Rather than allocate every unique word in the training corpus to a distinct feature column, one can optionally perform feature selection to be more discriminating about which words to provide as input to the learning algorithm. This has two major motivations:

1. Accuracy (error rate, F-measure, ROC area, etc.): The accuracy of many learning algorithms can be improved by selecting the most predictive features. For example, Naïve Bayes tends to perform poorly without feature selection in text classification settings. The purpose of feature selection is sometimes described as a need to eliminate useless noise words, but a study showed that even the lower ranked words continue to have predictive value[9]—only a small set of words are truly equally likely to occur in each class. Thus, feature selection may be viewed as selecting those words with the strongest signal-to-noise

ratio. Pragmatically, the goal is to select whatever subset of features yields a highly accurate classifier.

2. Scalability: A large text corpus can easily have tens to hundreds of thousands of distinct words. By selecting only a fraction of the vocabulary as input, the induction algorithm may require a great deal less computation. This may also yield savings in storage or network bandwidth. These benefits could be an enabling factor in some applications, e.g. involving large numbers of classifiers to train or large numbers of cases.

Even so, the need for feature selection has been somewhat lessened by continuing advances in the accuracy and scalability of core machine learning algorithms. For example, Joachims recently demonstrated a new linear SVM classifier that can be trained on over 800,000 text cases with nearly 50,000 word features in less than 3 minutes on a 3.6GHz PC processor [10]. What is more, for some training sets, feature selection provides no improvement in accuracy. Hence, the additional complexity of feature selection can be omitted for many researchers who are not interested in feature selection, but simply need a fixed and easily replicable input representation.

Nevertheless, a data-mining practitioner faced with a given training set from which to produce the best possible classifier should not ignore feature selection. It can significantly boost accuracy for some datasets, and may at least produce modest improvements on average. Thus, feature selection still has a role to play for those who seek to maximize accuracy, e.g. industrial practitioners, application programmers, and contestants in data-mining competitions.

Moreover, the accuracy and scalability benefits accrue more substantially when one considers other possibilities for feature terms besides just individual words. For example, having a single feature representing the occurrence of the phrase ‘John Denver’ can be far more predictive for some classification tasks than just having one feature for the word ‘John’ and another for the word ‘Denver.’ Other potentially useful features include any consecutive sequence of characters (n-grams) and, for domains that include multiple text fields (e.g. title, authors, abstract, keywords, body, and references), separate feature sets may be generated for each field or any combination of concatenated fields. It can be prohibitive simply to extend the bag of terms to include every potential feature that occurs in the training corpus. Thus, feature selection is also needed for scalability into large feature spaces. One can then search via cross-validation to improve the input representation to the core induction algorithm. That is, different choices for feature generators can be tried, as well as different choices for feature selectors. In this way, the scalability improvements of feature selection can also benefit accuracy by extending the space that may be searched in a reasonable time.

In an ideal world, we might know, for any task domain, the best feature generator and feature selection method that dominates all others. However, in the research literature, no single dominant method appears. We must either choose one ourselves from among many reasonable options, or use cross-validation to select one of many. If the latter, then our role becomes one of providing a sufficiently large (but not intractable) search space to cover good possibilities. This changes the game somewhat—we can propose features that might be useful, without having to assure their usefulness.

Section 2 describes a variety of common feature generators, which may be used to produce many potentially useful features. Section 3 describes the details of feature selection for binary and multi-class settings. Section 4 discusses the efficient evaluation of feature selection, and the computational corners that may be cut for repeated evaluations. Section 5 illustrates the gains that the described methods can provide, both in selecting a subset of words and in selecting a good combination of feature generators. The remainder of this introduction describes the three major paradigms of feature selection, and the common characteristics of the text domain.

1.1 Feature Selection Phyla

There are three major paradigms of feature selection: *Filter methods* evaluate each feature independently with respect to the class labels in the training set, and determine a ranking of all features, from which the top ranked features are selected [1]. *Wrapper methods* use classic AI search methods—such as greedy hill-climbing or simulated-annealing—to search for the 'best' subset of features, repeatedly evaluating different feature subsets via cross-validation with a particular induction algorithm. *Embedded methods* build a usually linear prediction model that simultaneously tries to maximize the goodness-of-fit of the model and minimize the number of input features [7]. Some variants build a classifier on the full dataset, and then iteratively remove features the classifier depends on least [8]. By beginning with the full dataset, they qualify as the least scalable. Given large feature spaces, memory may be exceeded simply to realize the full feature vectors with all potential features. We will not consider such methods further. Filter methods are the simplest to implement and the most scalable. Hence, they are appropriate to treat very large feature spaces and are the focus here. They can also be used as a pre-processing step to reduce the feature dimensionality sufficiently to enable other, less scalable methods. Wrapper methods have traditionally sought specific combinations of individual features from the power set of features, but this approach scales poorly for the large number of features inherent with classifying text. Using cross-validation to select among feature generators and optimize other parameters is somewhat like a wrapper method, but one that involves far fewer runs of the induction algorithm than typical wrapper feature selection.

1.2 Characteristic Difficulties of Text Classification Tasks

Besides the high dimensionality of the feature space, text classification problems are also characterized as frequently having a high degree of class imbalance. Consider training a text classifier to identify pages on any particular topic across the entire Web. High class skew is problematic for induction algorithms. If only 1% of the training cases are in the positive class, then the classifier can obtain 99% accuracy simply by predicting the negative class for all cases. Often the classifier must have its decision threshold adjusted in a separate post-processing phase, or else it must explicitly optimize for F-measure—which pressures it to increase recall of the positive class, without sacrificing too much precision.

One complication of high class skew is that even large training sets can end up having very few positive examples from which to characterize the positive class. Given 1% positives, a training set with 5000 randomly selected, manually labeled examples ends up with only 50 positives on average. This leads to significantly more uncertainty in the frequency estimates of words in the positive class than in the negative class. And if a predictive word is spelled 'color' in half the positive cases and 'colour' in the other half, then this *dispersion* of information into separate features yields more uncertainty. In technical notes or web text, we often encounter misspellings, which may yield other variants, such as 'collor.' This problem is exacerbated by the fact that natural language provides many ways to express the same idea, e.g. hue, tint, shade, dye or paint.

Another common aspect of text classification is that the large feature space typically follows a Zipf-like distribution [11]. That is, there are a few very common words, and very many words that rarely appear. By contrast, the most predictive features would be those that appear nearly always in one class, but not in the other.

Finally, text classification problems sometimes have only small amounts of training data available, perhaps more often than in other domains. This may partly be because a person's judgment is often needed to determine the topic label or interest level. By contrast, non-text classification problems may have their training sets labeled by machines sometimes, e.g. classifying which inkjet pens during manufacture ultimately fail their final quality test.

2 Text Feature Generators

Before we address the question of how to discard words, we must first determine what shall count as a word. For example, is 'HP-UX' one word, or is it two words? What about '650-857-1501'? When it comes to programming, a simple solution is to take any contiguous sequence of alphabetic characters; or alphanumeric characters to include identifiers such as 'ioctl32', which may sometimes be useful. By using the Posix regular expression `\p{L&}+` we avoid breaking 'naïve' in two, as well as many accented words in French, German, etc. But what about 'win_32', 'can't' or words that may be hyphenated over a line break? Like most data cleaning endeavors, the list of exceptions is endless, and one must simply draw a line somewhere and hope for an 80%-20% tradeoff. Fortunately, semantic errors in word parsing are usually only seen by the core learning algorithm, and it is their statistical properties that matter, not its readability or intuitiveness to people. Our purpose is to offer a range of feature generators so that the feature selector may discover the strongly predictive features. The most beneficial feature generators will vary according to the characteristics of the domain text.

2.1 Word Merging

One method of reducing the size of the feature space somewhat is to merge word variants together, and treat them as a single feature. More importantly, this can also improve the predictive value of some features.

Forcing all letters to lowercase is a nearly ubiquitous practice. It normalizes for capitalization at the beginning of a sentence, which does not otherwise affect the word's meaning, and helps reduce the dispersion issue mentioned in the introduction. For proper nouns, it occasionally conflates other word meanings, e.g. 'Bush' or 'LaTeX.'

Likewise, various word stemming algorithms can be used to merge multiple related word forms. For example, 'cat,' 'cats,' 'catlike' and 'catty' may all be merged into a common feature. Stemming typically benefits recall but at a cost of precision. If one is searching for 'catty' and the word is treated the same as 'cat,' then a certain amount of precision is necessarily lost. For extremely skewed class distributions, this loss may be unsupportable.

Stemming algorithms make both over-stemming errors and under-stemming errors, but again, the semantics are less important than the feature's statistical properties. Unfortunately, stemmers must be separately designed for each natural language, and while many good stemmers are available for Romance languages, other languages such as Hebrew and Arabic continue to be quite difficult to stem well. Another difficulty is that in some text classification applications, multiple natural languages are mixed together, sometimes even within a single training case. This would require a language recognizer to identify which stemming algorithm should be used on each case or each sentence. This level of complexity and slowdown is unwelcome. Simply taking the first few characters of each word may yield equivalent classification accuracy for many classification problems.

For classifying technical texts or blogs, misspellings may be common or rampant. Inserting an automatic spelling correction step into the processing pipeline is sometimes proposed, but the mistakes introduced may outweigh the purported benefit. One common problem is that out-of-vocabulary (OOV) words of the spell checker may be forced to the nearest known word, which may have quite a different meaning. This often happens with technical terms, which may be essential predictors. For misspellings that are common, the misspelled form may occur frequently enough to pose a useful feature, e.g. 'volcanoe.'

A common source of OOV words is abbreviations and acronyms, especially in governmental or technical texts. Where glossaries are available, the short and long forms may be merged into a single term. Although various acronym dictionaries are available online, there are many collisions for short acronyms, and they tend to be very domain-specific and even document-specific. Some research has shown success recognizing acronym definitions in text, such as '(OOV)' above, which provides a locally unambiguous definition for the term.

Online thesauruses can also be used to merge different words together, e.g. to resolve the 'color' vs. 'hue' problem mentioned in the introduction. Unfortunately, this approach rarely helps, as many words have multiple meanings, and so their meanings become distorted. To disambiguate word meanings correctly would require a much deeper understanding of the text than is needed for text classification. However, there are domain-specific situations where thesauruses of synonyms can be helpful. For example, if there is a large set of part numbers that correspond to a common product line, it could be very beneficial to have a single feature to represent this.

2.2 Word Phrases

Whereas merging related words together can produce features with more frequent occurrence (typically with greater recall and lower precision), identifying multiple word phrases as a single term can produce rarer, highly specific features (which typically aid precision and have lower recall), e.g. 'John Denver' or 'user interface.' Rather than require a dictionary of phrases as above, a simple approach is to treat all consecutive pairs of words as a phrase term, and let feature selection determine which are useful for prediction. The recent trend to remove spaces in proper names, e.g. 'SourceForge,' provides the specificity of phrases without any special software consideration—perhaps motivated by the modern world of online searching.

This can be extended for phrases of three or more words with occasionally more specificity, but with strictly decreasing frequency. Most of the benefit is obtained by two-word phrases [13]. This is in part because portions of the phrase may already have the same statistical properties, e.g. the four-word phrase 'United States of America' is covered already by the two-word phrase 'United States.' In addition, the reach of a two-word phrase can be extended by eliminating common stopwords, e.g. 'head of the household' becomes 'head household.' Stopword lists are language specific, unfortunately. Their primary benefit to classification is in extending the reach of phrases, rather than eliminating commonly useless words, which most feature selection methods can already remove in a language-independent fashion.

2.3 Character N-grams

The word identification methods above fail in some situations, and can miss some good opportunities for features. For example, languages such as Chinese and Japanese do not use a space character. Segmenting such text into words is complex, whereas nearly equivalent accuracy may be obtained by simply using every pair of adjacent Unicode characters as features—*n-grams.* Certainly many of the combinations will be meaningless, but feature selection can identify the most predictive ones. For languages that use the Latin character set, 3-grams or 6-grams may be appropriate. For example, n-grams would capture the essence of common technical text patterns such as 'HP-UX 11.0', '`while (<>) {`', '`#!/bin/`', and '` :)`'. Phrases of two adjacent n-grams simply correspond to (2n)-grams. Note that while the number of potential n-grams grows exponentially with n, in practice only a small fraction of the possibilities occur in actual training examples, and only a fraction of those will be found predictive.

Interestingly, the common Adobe PDF document format records the position of each character on the page, but does not explicitly represent spaces. Software libraries to extract the text from PDF use heuristics to decide where to output a space character. That is why text extracts are sometimes missing spaces between words, or have a space character inserted between every pair of letters. Clearly, these types of errors would wreak havoc with a classifier that depends on spaces to identify words. A more robust approach is for the feature generator to strip all whitespace, and generate n-grams from the resulting sequence.

2.4 Multi-Field Records

Although most research deals with training cases as a single string, many applications have multiple text (and non-text) fields associated with each record. In document management, these may be *title, author, abstract, keywords, body*, and *references.* In technical support, they may be *title, product, keywords, engineer, customer, symptoms, problem description*, and *solution.* Multi-field records are common in applications, even though the bulk of text classification research treats only a single string. Furthermore, when classifying long strings, e.g. arbitrary file contents, the first few kilobytes may be treated as a separate field and may often prove sufficient for generating adequate features, avoiding the overhead of processing huge files, such as tar or zip archives.

The simplest approach is to concatenate all strings together. However, supposing the classification goal is to separate technical support cases by product type and model, then the most informative features may be generated from the product description field alone, and concatenating all fields will tend to water down the specificity of the features.

Another simple approach is to give each field its own separate bag-of-words feature space. That is, the word ‘OfficeJet’ in the title field would be treated as though it were unrelated to a feature for the same word in the product field. Sometimes multiple fields need to be combined, while others are kept separate, and still others are ignored. These decisions are usually made manually today. Here again an automated search can be useful to determine an effective choice. This increases computation time for the search, but more importantly, it saves the expert’s time, and it may discover better choices than would have been explored manually.

2.5 Other Properties

For some classification tasks, other text properties besides words or n-grams can provide the key predictors to enable high accuracy. Some types of spam use deceptions such as '4ree v!@gr@ 4 u!' to thwart word-based features, but these might easily be recognized by features revealing their abnormal word lengths and the density symbols. Likewise, to recognize Perl or awk code, the specific alphanumeric identifiers that appear are less specific than the distribution of particular keywords and special characters. Formatting information, such as the amount of whitespace, the word count, or the average number of words per line can be key features for particular tasks.

Where task-specific features are constructed, they are often highly valuable, e.g. parsing particular XML structures that contain name-value pairs. By being task-specific, it is naturally difficult to make generally useful comments about their generation or selection. The little that is said about task-specific features in the text classification literature belies their true importance in many practical applications.

2.6 Feature Values

Once a decision has been made about what to consider as a feature term, the meaning of the numerical feature must be determined. For some purposes, a binary value is sufficient, indicating whether the term appears at all. This representation is used by the Bernoulli formulation of the Naïve Bayes classifier[12]. Many other classifiers use the term frequency $tf_{t,k}$ (the word count in document k) directly as the feature value, e.g. the Multinomial Naïve Bayes classifier[12].

The support vector machine (SVM) has proven highly successful in text classification. For such kernel methods, the distance between two feature vectors is typically computed as their dot product (cosine similarity), which is dominated by the dimensions with larger values. To avoid the situation where the highly frequent but non-discriminative words (such as stopwords) dominate the distance function, one can either use binary features, or else weight the term frequency value $tf_{t,k}$ inversely to the feature's document frequency df_t in the corpus (the number of documents in which the word appears one or more times). In this way, very common words are downplayed. This idea, widely known as 'TF.IDF,' has a number of variants, one form being $tf_{t,k} \times log(\frac{M+1}{df_t+1})$, where M is the number of documents. While this representation requires more computation and more storage per feature than simply using binary features, it can often lead to better accuracy for kernel methods.

If the document lengths vary widely, then a long document will exhibit larger word counts than a short document on the same topic. To make these feature vectors more similar, the $tf_{t,k}$ values may be normalized so that the length (Euclidean norm) of each feature vector equals 1.

3 Feature Filtering for Classification

With a panoply of choices for feature generation laid out, we now turn to feature filtering, which independently scores each feature with respect to the training class labels. The subsections below describe how this can be done for different classification settings. After the scoring is completed, the final issue is determining how many of the best features to select for the best performance. Unfortunately, the answer varies widely from task to task, so several values should be tried, including the option of using all features. This parameter can be tuned automatically via cross-validation on the training data.

Cross-validation may also be needed to select which feature generators to use, as well as selecting parameters for the induction algorithm, such as the well known complexity constant C in the SVM model. The simplest to program is to optimize each parameter in its own nested loop. However, with each successive nesting a smaller fraction of the training data is being given to the induction algorithm. For example, if nested 5-fold cross-validation is being used to select the feature generator, the number of features, and the complexity constant, then the inner-most loop trains with only half of the training set: $\frac{4}{5} \times \frac{4}{5} \times \frac{4}{5} = 51\%$. Unfortunately, the optimal parameter values found for this small training set may be a poor choice for the full size training set. (This is one reason why 10-fold cross-validation, despite its computational cost, is usually preferred to 2-fold cross-validation, which trains on nearly half as much of the data.)

Instead, a single loop of cross-validation should be married with a multi-parameter search strategy. The simplest to program is to measure the cross-validation accuracy (or F-measure) at each point on a simple grid, and then select the best parameters. There is a large literature in multi-parameter optimization that has yielded methods that are typically much more efficient, although more complex to program.

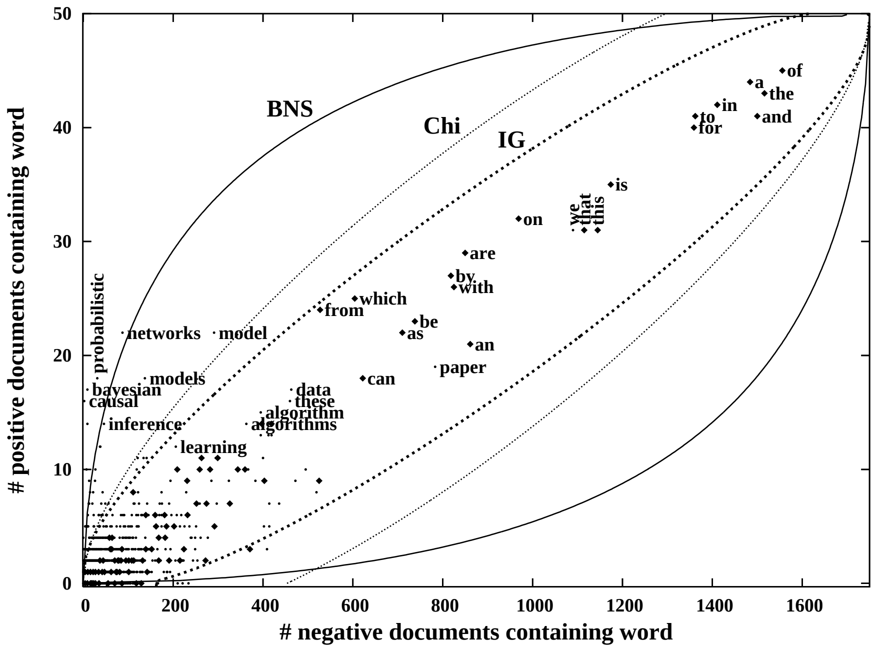

Figure 1: Word document frequencies in the positive and negative classes for a typical problem with class imbalance.

3.1 Binary Classification

We first discuss the setting where there are two-classes. Binary classification is a fundamental case, because (1) binary domain tasks are common, e.g. identifying spam email from good email, and (2) it is used as a subroutine to solve most types of multi-class tasks.

To clarify the nature of the problem, we demonstrate with an exemplary binary task: identifying papers about probabilistic machine learning methods among a collection of other computer science papers. The dataset has 1800 papers altogether, with only 50 of them in the positive class—2.8% positive. Each is represented by its title and abstract, which generate 12,500 alphanumeric words when treated as a single text string. Figure 1 shows, for each word feature t, the document frequency count tp_t with respect to the 50 documents in the positive class (y-axis) and separately fp_t for the 1750 documents in the negative class (x-axis), similar to an ROC graph. A feature in the topmost left corner would be perfectly predictive of the positive class, and would aid the induction algorithm a great deal. Unfortunately, this region is typically devoid of features.

Table 1: Three common feature selection formulae, computed from document frequency counts in the positive and negative classes.

Name	**Formula**
Information Gain (IG)	$e(pos, neg) - [P_{word} e(tp, fp) + (1 - P_{word}) e(fn, tn)]$ where $e(x, y) = -\text{xlx}(\frac{x}{x+y}) - \text{xlx}(\frac{y}{x+y})$, and $\text{xlx}(x) = x \log_2(x)$
Chi-Squared, χ^2 (Chi)	$g(tp, (tp + fp)P_{pos}) + g(fn, (fn + tn)P_{pos}) +$ $g(fp, (tp + fp)P_{neg}) + g(tn, (fn + tn)P_{neg})$ where $g(count, expect) = \frac{(count - expect)^2}{expect}$
Bi-Normal Separation (BNS)	$\lvert F^{-1}(tpr) - F^{-1}(fpr) \rvert$ where F^{-1} is the inverse of the Normal CDF

Notation:

pos:	number of positive cases $= tp + fn$
neg:	number of negative cases $= fp + tn$
tp:	true positives = number of positive cases containing the word
fp:	false positives = number of negative cases containing the word
fn:	false negatives
tn:	true negatives
tpr:	true positive rate $= tp/pos$
fpr:	false positive rate $= fp/neg$
P_{pos}:	percentage of positive cases $= pos/all$
P_{neg}:	percentage of negative cases $= neg/all$
P_{word}:	percentage of cases containing word $= (tp + fp)/all$

Common stopwords such as ‘of’ and ‘the’ occur frequently in both classes, and approximate the diagonal. These have no predictive value. The slightly larger points indicate which words appear on a generic list of 570 common English stopwords. Observe that the non-stopwords ‘paper’ and ‘algorithm’ behave like stopwords in this dataset, unsurprisingly. This illustrates that stopwords are not only language-specific, but also domain-specific.

Because of the Zipf-like distribution of words, most words occur rarely in each class. In fact, the majority of the points are plotted atop one another near the origin, belying their overwhelming density in this region. Over half of the words appear only once in the dataset and are plotted at just two points—(1,0) and (0,1). One often removes such extremely rare words via a minimum count threshold df_{min}; in this case, $df_{min} = 2$ removes about half the features. Whether this is a good idea depends on the induction algorithm and the character of the dataset. Raising df_{min} typically hurts precision.

Filter methods evaluate each feature term t according to a function of its document frequency counts in the positive and negative classes. Three commonly used feature selection formulae are given in Table 3.1: Information Gain (IG), Chi-Squared (Chi), and Bi-Normal Separation (BNS).

Returning to Figure 1, the contour lines on the graph show the decision boundaries that these three feature selection methods would make for this dataset when the top 100 features are requested. That is, for each method, the points along its contour lines all evaluate to the same ‘goodness’ value according to its function, and there are 100 features with greater values. Naturally, each method devalues and removes the stopwords near the diagonal, without being language- or domain-specific, as stopword lists would be.

The features that are selected lie above the upper contour as well as below the matching lower contour; the contours are rotationally symmetric about the center point. Despite this symmetry, these two areas differ in character because the Zipfian word distribution focuses the selection decisions to be near the origin, and the feature selection methods each have an asymmetry. The most noticeable asymmetry is that the chi-squared method results in a strong bias towards positive features; there are no features under its lower contour. Information gain selects some negative features, but still has a bias for positive features. This is more easily seen at the top, where there are no word points obscuring the place where the contour meets the top of the graph.

By contrast, the BNS decision boundary comes much closer to the origin on the x-axis. Compared to the other methods, BNS prefers many more of the negative features—in this case, only those occurring more than 50 times among the negatives and not once among the 50 positives. It is for this reason that BNS excels in improving recall, usually at a minor cost to precision. This tradeoff often yields an overall improvement in F-measure compared to other methods.

Why is BNS asymmetric, given that its formula is symmetric? It stems from the class skew. Since the inverse Normal cumulative distribution function (CDF) is undefined at zero, whenever there are zero occurrences of a word in a class, we must substitute a small constant, e.g. $\xi = 0.1$ occurrences. Since there are typically more negatives than positives, the minimum false positive rate $fpr = \frac{\xi}{neg}$ is smaller than the minimum true positive rate $tpr = \frac{\xi}{pos}$. In this way, a feature that occurs x times in only the minority class is correctly preferred to one that occurs x times in only the majority class.

Likewise, to avoid the undefined value $F^{-1}(1.0)$, if ever a feature occurs in every single positive (negative) case, we back off the tp (fp) count by ξ. This does not occur naturally with language texts, but text classification techniques are regularly used to treat string features of all sorts of classification problems. In industrial settings with many classes to process, it sometimes happens that there is a perfect indicator in the texts, e.g. `<meta name="Novell_ID" val="Win32">`, which may be discovered by long n-grams or phrases of alphanumeric words. Note that without feature selection, an SVM classifier will not make effective use of a few excellent features [5].

As a side note, sometimes the purpose of feature selection is just to characterize a class for user understanding rather than machine classification. In this case, ordinarily one wants to see only the positive words and phrases.

3.2 Multi-Class Classification

There are two major forms of multi-class classification: single-label (1-of-n) classification, where each case is known to belong in exactly one of the n classes, and multi-label (m-of-n) classification, where each case may belong to several, none, or even all classes.

In the multi-label case, the problem is naturally decomposed into n binary classification tasks: class_i vs. not class_i. Each of these binary tasks is solved independently, and each can have its own feature selection to maximize its accuracy. In the single-label case, many induction algorithms operate by decomposing the problem into n binary tasks as above, and then making a final decision by some form of voting. Here also, feature selection can be optimized independently for each binary subtask. However, some 1-of-n induction algorithms do not perform binary decompositions, and need *multi-class feature selection* to select a single set of features that work well for the many classes. Other 1-of-n induction algorithms perform very many binary decompositions, e.g. those that search for optimal splitting hierarchies, or error-correcting code classifiers that consider $O(n^2)$ dichotomies. For these situations, it may be preferable to perform one multi-class feature selection than a separate binary feature selection for each dichotomy.

All multi-class tasks could be dealt with by binary decompositions in theory, and so there would be no need for multi-class feature selection. However, practice often recants theory. The APIs for many good software products and libraries expect the transformation of text into numerical feature vectors to be performed as a pre-processing step, and there is no facility for injecting feature selection into the inner loops, where the decompositions occur. Even some m-of-n applications that can be programmed *de novo* demand multi-class feature selection for performance and scalability reasons. For example, where a centralized server must classify millions of objects on the network into multiple, orthogonal taxonomies, it can be much more efficient to determine a single, reasonably sized feature vector to send across the network than to send all the large documents themselves. In another example application[4], a large database of unstructured, multi-field (technical support) cases is represented in memory by a cached, limited size feature vector representation. This is used for quick interactive exploration, classification, and labeling into multiple 1-of-n and m-of-n taxonomies, where the classifiers are periodically retrained in real time. It would be impractical to re-extract features for each binary decomposition, or to union all the features into a very long feature vector that would be requested by all the binary feature selection subtasks.

Many multi-class feature selection schemes have been devised, and some methods such as Chi-squared naturally extend to multiple classes. However, most of them suffer from the following liability: consider a multi-class topic recognition problem, where one of the classes happens to contain all German texts. The German class will generate many extremely predictive words. Nearly all feature selection schemes will prefer the stronger features, and myopically starve the other classes for features. Likewise, if one class is particularly difficult, multi-class feature selectors will tend to ignore it, since it offers no strong features. If anything, such difficult classes need more features, not fewer.

A solution to this problem is to perform feature selection for each class separately via binary decompositions, and then to determine the final ranking of features by a round-robin algorithm where each class gets to nominate its most desired features in turn [2]. This scheme was devised to improve robustness for unusual situations that arise in practice only occasionally. Usually efforts to improve robustness come at some loss in average performance. Remarkably, this improves performance even for well-balanced research benchmarks. Why? Inevitably, some classes are easier to recognize than others, and this disparity causes most feature selection methods to slight the very classes that need more help.

3.3 Hierarchical Classification

Hierarchy is among the most powerful of organizing abstractions. *Hierarchical classification* includes a variety of tasks where the goal is to classify items into a set of classes that are arranged into a tree or directed acyclic graph, such as the Yahoo web directory. In some settings, the task is a single-label problem to select 1-of-n nodes—or even restricted to the leaf classes in the case of a 'virtual hierarchy.' In other settings, the problem is cast as a multi-label task to select multiple interior nodes, optionally including all super-classes along the paths to the root.

Despite the offered hierarchy of the classes, these problems are sometimes treated simply as flat multi-class tasks, aggregating training examples up the tree structure for each class. Alternately, a top-down hierarchy of classifiers can be generated to match the class hierarchy. The training set for each step down the tree is composed of all the training examples under each child subtree, optionally including a set of items positioned at the interior node itself, which terminates the recursion. Although this decomposition of classes is different from a flat treatment of the problem, in either decomposition, the same single-label or multi-label feature selection methods apply to the many sub-problems. It has been suggested that each internal hierarchical classifier may be faster because each may depend on only a few features (selected by feature selection), and may be more accurate because it only consider cases within a limited context. For example, an interior node about recycling that has subtopics for glass recycling and can recycling would have a classifier under it that need only consider cases that have to do with recycling. In this way, the training sets for each of the interior classifiers may be more balanced than with a flat treatment of the problem.

4 Practical and Scalable Computation

We discuss briefly the matter of programming software for feature selection, with some practical pointers for efficient implementation. These issues are usually not recorded in the research literature or in product documentation. Since feature selection is often accompanied by multiple runs of cross-validation to select the best number of features, it makes sense to save computation where possible, rather than run each fold from scratch.

We begin with the binary case. By dividing the training cases into F folds in advance, the true positive and false positive counts can be kept track of separately for each fold. It then becomes very fast to determine the tp, fp counts for any subset of folds using just 2F integers per feature. This makes feature-filtering methods extremely scalable, and requires only one pass through the dataset.

Furthermore, for M training cases, each fold has only M/F cases, and an 8- or 16-bit counter will often suffice. For the 1800 paper dataset—which altogether can generate over 300,000 word, phrase, 3-gram, 4-gram, and 5-gram features—we can efficiently support feature selection for 10-fold cross-validation with less than 6MB of memory. This is an insignificant amount of memory nowadays and its computation takes only seconds on a PC. For 10-folds on $M \leq 640,000$ cases, 100MB of memory is sufficient for 2.6 million features. Moreover, likely half or more of the features will occur only once, and they can be discarded after one pass through the dataset, freeing up memory for inductions that follow in cross-validation.

Large generated feature spaces need not be stored on the first pass through the dataset. Once the counts are made—possibly on a subset of the training cases—the best, say, 100K features are determined (overall, and for each fold). Then a second feature-generation pass through the dataset stores only features that are actually needed. The subsequent cross-validation inductions then work with ever decreasing subsets of the realized dataset.

In the multi-class setting, further optimization is possible. If the total number of distinct classes is C, then we can efficiently determine the tp, fp counts for 'class vs. not class' binary subtasks using C+1 integers per feature (for this exposition, we ignore the orthogonal issue of the F-folds). The number of occurrences of the feature is tracked separately for each class, plus one additional integer tracks the total number of occurrences in the dataset. This missing fp counter is determined from the total minus the tp counter for the class. Further, if we know the classes are mutually exclusive (1-of-n classification), then we can efficiently determine the tp and fp counts for any dichotomy between any subsets of classes.

It is fortunate that feature selection for cross-validation can be so efficient. The bottleneck is then the induction algorithm. Reducing the number of folds from 10-fold to 2-fold cuts the workload substantially, but the smaller training sets can yield different behavior that misleads the search for optimum parameters. Specifically, using smaller 2-fold training sets may prefer substantially fewer features than is optimal for the full training set.

Rather than reduce the data folds, early termination can be used. With only a few of the fold measurements completed, the current parameter settings may be deemed inferior to the best result found so far. For example, suppose the best parameters made 80 misclassifications on all 10 folds, and the current parameters have already committed 80 mistakes halfway through the third fold. Early termination can be done even more aggressively with various statistical methods, and by being less conservative. After all, even with exhaustive evaluation of the folds, it is only a somewhat arbitrary subset of possibilities that are explored on a sample of the data.

Concerns about computational workload for practical text applications may gradually become insignificant, considering that 80-core CPUs are reportedly within a five-year horizon and that algorithmic breakthroughs often yield super-linear improvements.

5 A Case Study

The overall benefit of feature selection can vary to the extremes for different datasets. For some, the accuracy can be greatly improved by selecting ~1000 features, or for others, by selecting only a few strongly predictive features. For still others, the performance is substantially worse with anything fewer than all words. In some cases, including 5-grams among the features may make all the difference. Because large gains are sometimes possible, text feature selection will never become obsolete—although it would be welcome if it were hidden under layers of software the way SVM solvers are today.

Nonetheless, the chapter would not be complete without an example. Figure 2 shows the improvement in F-measure for including feature selection vs. just giving all word features to the state-of-the-art SVM-Perf classifier [10]. The improvement is shown for 6 different (mutually exclusive) classes, corresponding to different computer science topics in machine learning (each with 2.8% positives). Half the dataset was used for training, and the other half was used for testing; five such splits were evaluated and their results are shown as five separate whiskers for each class. The main point is how large a gain is *sometimes* made by considering feature selection. In every case, the

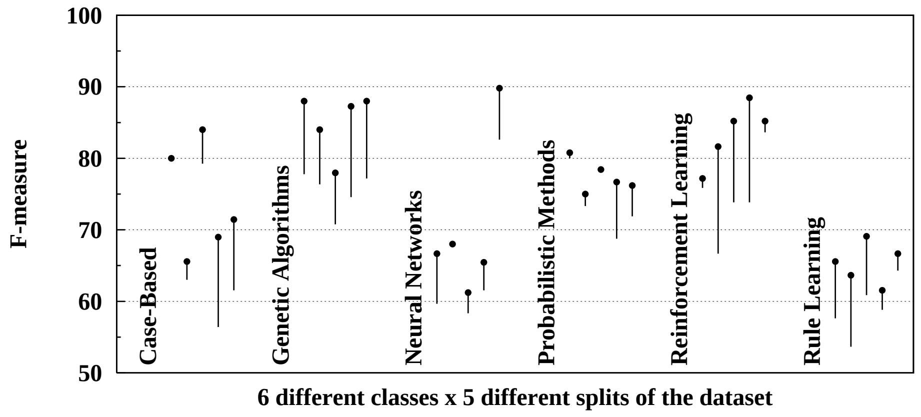

Figure 2: F-measure improvement via feature selection. The dot shows the best performance with feature selection; the other end of the whisker shows the best performance without feature selection, i.e. simply using all words.

SVM complexity constant C was optimized from a set of five values ranging from 0.01 to 1. The number of features was optimized from values of 100 features and up, in steps of 1.5×, including using all features. The optimal parameters were chosen according to which showed the best F-measure on the test set. Certainly, this is not a large enough study or dataset to draw general conclusions, but the potential benefit of feature selection is clearly illustrated. A full-scale study would also need to optimize its parameters via cross-validation on the training set, rather that taking the omniscient view we have here for expediency.

Once we have paid the software complexity price to have the cross-validation framework in place, we can also use it to try different feature generators. Figure 3 shows the further improvement in F-measure over the previous figure that is available by trying different combinations of feature generators. The three settings tried were: (1) words, (2) words plus 2-word phrases, and (3) words plus 2-word phrases, plus 3-grams, 4-grams and 5-grams. (The maximum performance without the n-grams is shown by the cross-hair, revealing that most of the performance gain is often captured by 2-word phrases. Nonetheless, n-grams do sometimes improve performance.)

(For the record, the example used: a rare word cutoff of 2, which eliminated any word that appeared only once in the given training set; a list of 571 English stopwords; BNS feature scoring with the minimum count $\xi = .1$; TF.IDF feature scaling without length normalization, since the ab-

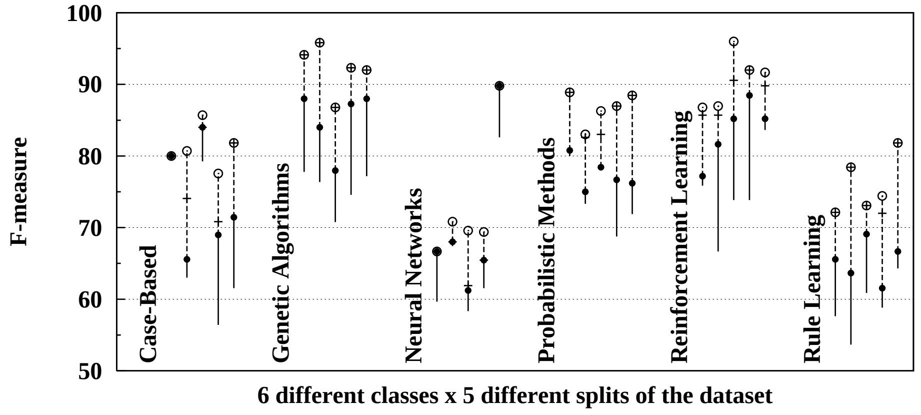

Figure 3: Further F-measure improvement for trying different feature generators. The plot from Figure 2 is reproduced, and the whiskers are extended up to the maximum performance for using words, + 2-word phrases, + 3-, 4-, and 5-grams.

stracts have near uniform length; SVM-Perf with its default settings, except the flag (`-l 1`), which tells it to optimize for F-measure rather than accuracy; and words consisting of alphanumeric characters plus underscore, and forced to lowercase.)

6 Conclusion and Future Work

Text classification is an elephant among blind researchers. As we approach it from different sides, we inevitably find that different strategies are called for in feature generation and feature selection. Unfortunately for the practitioner, there is much sound advice that is conflicting. A challenge for research in this decade is to develop methods and convenient software packages that consistently generate feature sets leading to good accuracy on most any training set, without requiring the user to spend their time trying different modules and parameter settings. Today, when faced with lackluster text classification performance on a new domain problem, one has to wonder whether it could be greatly improved by 'tweaking the many knobs,' or whether the poor performance is inherent to the data.

Cross-validation for model selection and parameter tuning appears to be the straightforward solution. However, proposing a large number of potential features for a class that has few training cases can lead to overfitting the training data—generating features that are only predictive for the particular training cases studied. Small training sets are a common problem, since obtaining correct topic labels requires people's time and concentration. Even a seemingly large training set can be meager if it is divided into many classes, if the class sizes are highly imbalanced, or if the words used for a single topic are diverse. Examples of the latter include multilingual texts, many creative authors, or topics that consist of many implicit subtopics, such as sports. These are common situations in real-world datasets and pose worthy research challenges, since obtaining additional training data usually comes with a cost. One direction may be to develop priors that leverage world knowledge, e.g. gathered from many other available training sets [3][6].

Other open problems arise in generating and selecting useful features for classes that are not topic-based. For example, one may need to classify texts as business vs. personal, by author, or by genre (e.g. news, scientific literature, or recipes). In these situations, the specific topic words used are less predictive, and instead one may need features that represent the verb tenses used, complexity of sentences, or pertinence to company products. While there is a healthy and growing literature in authorship, genre and sentiment classification, there are many other types of desirable and challenging classifications that have not been addressed, for example, determining the writing quality of an article containing figures, or classifying company web sites into a multi-faceted yellow pages, such as UDDI.org. There is certainly no shortage of research opportunities.

References

[1] G. Forman. An extensive empirical study of feature selection metrics for text classification. *J. of Machine Learning Research*, 3:1289–1305, 2003.

[2] G. Forman. A pitfall and solution in multi-class feature selection for text classification. In *ICML '04: Proc. of the 21st Int'l Conf. on Machine learning*, pages 297–304. ACM Press, 2004.

[3] G. Forman. Tackling concept drift by temporal inductive transfer. In *SIGIR '06: Proc. of the 29th int'l ACM SIGIR conf. on research and development in information retrieval*, pages 252–259. ACM Press, 2006.

[4] G. Forman, E. Kirshenbaum, and J. Suermondt. Pragmatic text mining: minimizing human effort to quantify many issues in call logs. In *KDD '06: Proc. of the 12th ACM SIGKDD int'l conf. on Knowledge discovery and data mining*, pages 852–861. ACM Press, 2006.

[5] E. Gabrilovich and S. Markovitch. Text categorization with many redundant features: using aggressive feature selection to make SVMs competitive with C4.5. In *ICML '04: Proc. of the 21st Int'l Conf. on Machine learning*, pages 321–328, 2004.

[6] E. Gabrilovich and S. Markovitch. Feature generation for text categorization using world knowledge. In *Proc. of The 19th Int'l Joint Conf. for Artificial Intelligence*, pages 1048–1053, Edinburgh, Scotland, 2005.

[7] I. Guyon and E. Elisseef, A. Special issue on variable and feature selection. *J. of Machine Learning Research*, 3:1157–1461, 2003.

[8] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1-3):389–422, 2002.

[9] T. Joachims. Text categorization with suport vector machines: Learning with many relevant features. In *ECML'98: Proc. of the European Conf. on Machine Learning*, pages 137–142. Springer-Verlag, 1998.

[10] T. Joachims. Training linear SVMs in linear time. In *Proc. of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 217–226, 2006.

[11] C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing.* The MIT Press, Cambridge, Massachusetts, 1999.

[12] A. McCallum and K. Nigam. A comparison of event models for naive bayes text classification. In *AAAI/ICML-98 Workshop on Learning for Text Categorization*, TR WS-98-05, pages 41–48. AAAI Press, 1998.

[13] D. Mladenic and M. Globelnik. Word sequences as features in text learning. In *Proceedings of the 17th Electrotechnical and Computer Science Conference (ERK98), Ljubljana, Slovenia*, pages 145–148, 1998.

[14] F. Sebastiani. Machine learning in automated text categorization. *ACM Comput. Surveys*, 34(1):1–47, 2002.