Reducing the Human Overhead in Text Categorization

Arnd Christian König
Microsoft Research
One Microsoft Way
Redmond, WA 98052
chrisko@microsoft.com

Eric Brill
Microsoft Research
One Microsoft Way
Redmond, WA 98052
brill@microsoft.com

ABSTRACT

Many applications in text processing require significant human effort for either labeling large document collections (when learning statistical models) or extrapolating rules from them (when using knowledge engineering). In this work, we describe a way to reduce this effort, while retaining the methods' accuracy, by constructing a hybrid classifier that utilizes human reasoning over automatically discovered text patterns to complement machine learning. Using a standard sentiment-classification dataset and real customer feedback data, we demonstrate that the resulting technique results in significant reduction of the human effort required to obtain a given classification accuracy. Moreover, the hybrid text classifier also results in a significant boost in accuracy over machine-learning based classifiers when a comparable amount of labeled data is used.

Categories and Subject Descriptors

H.1.2 [**Models and Principles**]: User/Machine Systems—*Human information processing*; H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing—*Indexing Methods*; I.5.4 [**Pattern Recognition**]: Applications—*Text processing*

General Terms

Algorithms, Experimentation

Keywords

active learning, classification, machine learning, supervised learning, support vector machines, text classification, text mining

1. INTRODUCTION

The automated categorization/classification of natural-language text into a number of predefined categories is the basis for many tasks in empirical text processing. The predominant approaches to this problem have been two-fold: (a) *knowledge-engineering* techniques, which involve manually building a set of rules encoding expert knowledge on how to classify documents and (b) *machine learning* techniques that use statistical models (such as Support-Vector Machines [19, 6] or Bayesian Networks [16]) which are trained using previously classified text corpora. The specific choice of knowledge-engineering framework or statistical model has been studied extensively in the research literature; however, from the perspective of an end-user looking to deploy a system, the true cost of setting up a specific NLP task does not lie in the overhead of developing the right classification algorithm, but rather in the overhead of having a human manually examine significant amounts of text to either infer classification rules or assign training labels.

This paper studies ways to reduce the amount of human effort required to build a text classifier. In our approach, we seek to leverage human input beyond individual labels for text documents, by exploiting two salient characteristics of text classification tasks: (1) A key property of text classification is that often knowledge of only a small subset of a documents' text is sufficient to make a correct classification decision. In fact, in some classification scenarios only a small subset of text is relevant – e.g. consider the example of sentiment detection in reviews where typically only a small fraction of a document conveys sentiment information, while the remainder can be ignored [12]. (2) The other property we leverage is the fact that humans are typically proficient at reasoning over classification rules specified in terms of text fragments, i.e. when presented with a set of possible rules that hold for a small training set of documents, humans can pick rules that generalize well without knowing the text corpus itself, by relying on knowledge of the relevant domain only. Variations of this property have been leveraged in a number of scenarios, e.g. [8, 15].

In a classification scenario we now leverage these properties as follows: initially, we run a pattern discovery algorithm over a *small* set of labeled training data to compute text patterns that are highly correlated with the occurrence of a specific label (i.e. if the pattern occurs, then – with high probability – so does the label). These patterns are then presented to a human user, who now selects among them text patterns that carry sufficient information to assign a document's label based on the pattern itself. We refer to such patterns as *discriminating* patterns. Here, special care must be taken to ensure that the patterns in question are of such type that a human can reason well over them.

Now, having obtained a set of discriminating patterns, we construct a 2-stage classifier as follows: An unclassified document is first checked if it matches any of the discriminating patterns. In this case, the document is assigned a label based on the pattern; if not, then the label is assigned by a machine-learning based classifier, that was trained on the same training documents used to discover the discriminating patterns (Figure 1).

To give an example of discriminating and non-discriminating patterns, consider the task of classifying user-feedback emails sent to a large company into mail expressing positive (+) and negative (-) sentiment. In this context, a frequent text pattern that has high

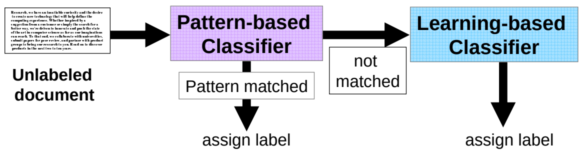


Figure 1: A hybrid classifier

correlation to the label (-) might be "*I will . . . switch to . . . XYZCorp*", where XYZCorp is the name of a competitor and the symbol '. . . ' accounts for gaps between the matching words in the sentence. A human annotator may now conclude that this pattern is discriminating and sufficient to identify negative customer feedback. In contrast, the text pattern "*XYZCorp*" may be as highly correlated and even more frequent in the training corpus, however, it does not convey sufficient information to make a classification decision based on the pattern itself, and hence is not discriminating.

This framework has two advantages over pure machine-learning: (1) Using a pattern-based classifier enables us to capture more complex and specific patterns than we would typically be able to learn (due to sparsity of the training data) using machine-learning based approaches alone. (2) We do get some of the benefits associated with knowledge-engineering approaches (transparency of the discriminating text patterns, ability to leverage domain knowledge) without having a human read through and abstract from a large collection of documents manually.

The main advantage over pure knowledge engineering is efficiency: coming up with discriminating rules manually requires significant human effort for examining the document collection for variations of and counterexamples to each rule. Moreover, the rules a human designer intuitively comes up with may not correspond to the most frequent discriminating patterns in the text collection; thus additional human cycles are required to search the document collection for additional patterns.

Our work makes the following contributions: (1) A framework that combines text mining with manual annotation of text patterns to construct a hybrid classifier that significantly increases the classifier accuracy relative to the amount of human effort. (2) Scalable algorithms to discover the most frequent text patterns satisfying a threshold on their correlation to a specific label. (3) A pattern matching framework that is sensitive to the contexts in which the meaning of a pattern may be changed (e.g. if the pattern is in the scope of a negation); as we will show later, this component is crucial in enabling our approach of having humans reason over text patterns without knowing all contextual information, which is the basis for our scheme's efficiency.

The remainder of this paper is organized as follows. In Section 2, we will briefly review related work. In Section 3, we define the class of text patterns used in our approach and describe scalable algorithms to extract such patterns from text corpora. In Section 4 we describe scenarios in which human reasoning is unreliable and augment our pattern-matching scheme to address this. In Section 5 we evaluate the efficiency of the proposed framework using multiple real-life datasets. We conclude in Section 6.

2. RELATED WORK

The value of human feedback for feature selection was recently studied in [15]. Here, the authors show that feature selection solely based on the labeled training corpus only has little effect, whereas human feedback on feature relevance can identify a large fraction of the most relevant features. Similar to our findings, this work shows that by leveraging the fact that features can be annotated much more rapidly than training documents, one can significantly reduce the overall overhead in classifier construction. Our work differs from this paper in that we use significantly more complex patterns and consider the patterns' context when making matching decisions. Also, we use text patterns in isolation of the actual machine-learned text classifier itself, allowing us to be agnostic to the specific classification technique and feature set.

On the surface, our method has some similarity with *active learning* techniques (e.g., [17, 18]). While both methods can be used to ultimately reduce the human effort in categorization, their approach is different: active learning is concerned with using humans to maximize the efficiency of labeling additional documents, whereas our approach leverages text mining to come up with what amounts to complementary rules among which humans are then asked to select. Recent results [15] indicate that these approaches may be combined effectively and can complement each other.

In the context of machine learning, a large number of different approaches exist that leverage bootstrapping (or co-learning, transduction. . .) schemes to construct a machine-learning classifier using little initial training data combined with human feedback and/or large amounts of unlabeled data (e.g. [4, 8, 11]). Because our approach isolates the text patterns from the machine learning classifier, all of these approaches can be combined with our work, potentially reducing the human overhead even further.

Regarding the data structures used to obtain the text patterns from the labeled documents themselves, our work is closely related to other approaches in text mining via suffix arrays [2, 3]. The key difference in our approach is that we seek to maximize a very different objective function, enabling pruning of the search space ([2, 3] require exhaustive traversal).

3. SCALABLE PATTERN DISCOVERY

3.1 Pattern Definition

Given a vocabulary $\mathcal{V}$ over words, we define a *text pattern* as an ordered sequence of elements from $\mathcal{V} \cup \{\ldots\}$, where the symbol '. . . ' denotes gaps in the text. We say that a sentence *matches* a pattern if it contains the symbols of the pattern in identical order while gaps may be filled with arbitrary sequences of words within a sentence (including the empty sequence) i.e. the pattern "*a . . . good product*" would match both "*This is a good product.*" and "*a certainly very good product.*". Patterns do not match across sentence boundaries. A document d matches a pattern p, if a sentence in d matches p. For a set of documents $\mathcal{D}$, we define the *frequency* of a pattern p as the number of distinct documents in $\mathcal{D}$ that match p.

We refer to a pattern p as *maximal* for a document collection if no pattern p' exists for which it holds that (a) p is a subsequence of p' and (b) every sentence in $\mathcal{D}$ that matches p also matches p'. We refer to a pattern as *simple* if it doesn't contain the gap symbol.
Notation: In the remainder of this paper, we will use the following notation: given a set of documents $\mathcal{D} = \{d_1, \ldots d_n\}$, and a set of labels $\mathcal{L} = \{l_1, \ldots, l_h\}$, we will denote the label of a document $d \in \mathcal{D}$ by $Label(d) \in \mathcal{L}$. For a given document set $\mathcal{D}$ and a pattern p, we use the term $freq(p)$ to denote the number of documents from $\mathcal{D}$ that match p. Similarly, we use the term $freq(l, p)$ to denote the the number of documents from $\mathcal{D}$ that match p *and* are labeled with the label $l \in \mathcal{L}$. We use the symbol $\circ$ to denote the concatination of strings, i.e. "A " $\circ$ "string" = "A string".

3.2 Pattern Discovery

Given this pattern definition, we want to discover patterns that are highly indicative of one label; to measure this, we simply use the probability of a label l given a pattern p in the training data:

$$Pr(l,p) := \frac{freq(l,p)}{freq(p)}.$$

We also can use any other type of *impurity function* here without changing the resulting algorithm. For a pattern to be displayed to the human annotator, we require a value of $Pr(l,p)$ greater than a threshold α, chosen to be larger than the desired classification accuracy. Among all patterns satisfying this condition, we now want to display the ones which occur in the largest number of documents, as they are assumed to occur more frequently in the unlabeled data as well. Thus, the search problem becomes:

Problem Statement: *Given a set of documents $\mathcal{D} = \{d_1, \ldots d_n\}$ and a threshold α on the required minimum probability induced by a pattern, compute the k most frequent patterns $p_1, \ldots, p_k$, such that for each p_i it holds that*

$$\max_{l \in \mathcal{L}} Pr(l, p_i) > \alpha. \quad (1)$$

For very unbalanced corpora, we may want to extract the most frequent patterns indicative of *each* label separately; our overall algorithm remains the same in either case.

This text mining problem differs from many data-mining scenarios studied over in the context of itemset data in two respects: first, the text data itself (when using a dictionary-based encoding of each word as an integer) typically fits into main memory – given that vocabulary sizes for natural language text typically are in the range of 100K-200K, one billion words can be represented in 3GB of memory. Second, mining approaches that explore the search-space bottom-up by computing the number of occurrences and $Pr(l,p)$-values for short word-combinations (e.g. bi- or trigrams) do not scale due to the large vocabulary size and the fact that the patterns allow gaps; e.g. a vocabulary of $|\mathcal{V}| = 25$K words can yield up to $|\mathcal{V}|^2 = 625 \cdot 10^6$ different bigrams. While the actual numbers encountered are smaller, they still grow fast enough to rule out bottom-up pruning. Therefore, we use an algorithm that is based on multiple *suffix arrays* [9] to compute these patterns.

3.2.1 Suffix Arrays

Given a vocabulary $\mathcal{V}$ and a text corpus $\mathcal{T}$ (of words from $\mathcal{V}$) made up of $\mathcal{D}$ documents, we represent $\mathcal{T}$ by encoding each word as an integer in $\{1, \ldots, |\mathcal{V}|\}$, marking the end of each sentence by a special symbol '#' . We denote the string resulting from concatinating all symbols in the corpus by T, using the notation $T[j]$ to denote the j-th symbol in T, and N to denote the length of T.

Now, a suffix array is a compact representation of a suffix tree [10] over T. The key advantage of suffix arrays over suffix trees is that their space requirements do not grow proportional to the product of the vocabulary size and N, but only proportional to N. Conceptually, a suffix array SA is an array of all N suffixes in T, sorted lexicographically. Each suffix is represented as a pointer into the corpus only, i.e. if $SA[j] = i$ then the j-th (in lexicographical order) suffix contained in T is the word sequence $T[i] \ldots T[N]$. We will refer to this number j as the *rank* of a the suffix starting at $T[i]$.

In addition to the suffix-array, we maintain an auxiliary array for storing LCPs (longest common prefixes), where each element, $LCP[j]$, indicates the length of the common prefix between the suffixes denotes by $SA[j]$ and $SA[j-1]$. Finally, we also maintain an inverted suffix array SA^{-1} defined via $SA^{-1}[i] = j \Leftrightarrow SA[j] = i$. Given that all of these arrays have N fields, the total structure requires $O(N)$ space, which in practice means less than 3 times the space required to store T, as all pointers and members of $\mathcal{V}$ are stored using 4 bytes, and the LCP array typically only requires single-byte counters (for natural text, patterns longer than 256 words are typically irrelevant). Construction of Suffix Arrays and the LCP array requires overhead linear in N (e.g. [7]).

3.2.2 Discovering Patterns without gaps

Once a suffix array over T has been constructed, we can compute the $Pr(l,p)$ for all maximal patterns that do not contain the gap symbol '...' using a modification of the algorithm proposed in [21] to compute the *term frequency* (= total number of occurrences) and *document frequency* (= $freq(p)$) of text patterns. This algorithm computes the document frequency for any maximal n-gram occurring in the text (and therefore any maximal pattern without gaps), with $n \in \{1, \ldots N\}$ in $O(N \log N)$ time; the modification we make is that we keep track of the $|\mathcal{L}|$ different document frequencies – corresponding to the sets of documents of each label. This modification is trivial, so we omit a detailed description.

The algorithm traverses all maximal patterns contained in T and also outputs for each pattern p the lowest and largest rank of suffixes starting with p (which means that all suffixes with ranks between these boundaries must also start with p), and the length of each pattern. In the following, we will denote the set of ranks for a pattern p as $rank(p)$ and its length by $length(p)$. Note that for our purposes it suffices to consider only the maximal patterns, as any non-maximal patterns have identical correlation, but contain less information for the human annotator.

3.2.3 Discovering Patterns with gaps

In the following, we will describe how to compute – for a given pattern p – all patterns of the form $p \circ \text{'...'} \circ p_1 \circ \ldots \circ p_m$ in T, with each p_i being a maximal simple pattern, as well as the corresponding suffix-ranks and $Pr(l, p \circ \text{'...'} \circ \ldots \circ p_m)$. We will refer to this as *expanding* the pattern.

First, consider the case of p being a simple pattern. Now, using the algorithm described above, we compute $rank(p)$; consequently, we can compute the starting positions of all suffixes after occurrences of p as $\{SA[r] + length(p) | r \in rank(p)\}$. Now, given that patterns cannot cross sentence-boundaries, we can compute the set of all positions $pos(p)$ in T that can potentially contain patterns of the form $p \circ \text{'...'} \circ p'$ using the set of suffixes of the occurrences of p in T. If we define the next sentence boundary after a position k in T as $bound(k) := \min\{l \in \mathbb{N} | l > k \text{ and } T[l] = \text{'\#'}\}$, then we can compute the set of positions as

$$pos(p) := \bigcup_{r \in rank(p)} \{SA[r] + length(p), \ldots, bound(SA[r])\}.$$

Now define the subset T_p as the subset of the text corpus T consisting of the items $T[i]$ where $i \in pos(p)$. We compute a suffix array SA_p on T_p by sorting the ranks corresponding to members of $pos(p)$ (which is typically much more efficient than building it by comparing the corresponding suffixes), including the corresponding LCP information and inverse suffix array SA_p^{-1}. Now, we can run the algorithm of [21] on this suffix array. Any patterns p' found in T_p correspond to occurrences of $p \circ \text{'...'} \circ p'$ in T, with the value of $Pr(l, p')$ with respect to T_p being the value of $Pr(l, p \circ \text{'...'} \circ p')$ with respect to T.

The construction for cases where p is not a simple pattern is almost identical. Let p be of the form $p_1 \circ \ldots \circ p_m$, where each p_i is a simple pattern. Then we compute the positions in $pos(p)$ as

$$pos(p) := \bigcup_{r \in rank(p)} \{SA_p[r] + length(p_m), \ldots, bound(SA_p[r])\}.$$

By repeating this construction, we can find patterns with arbitrary numbers of gaps. Overall, this construction results in a search tree

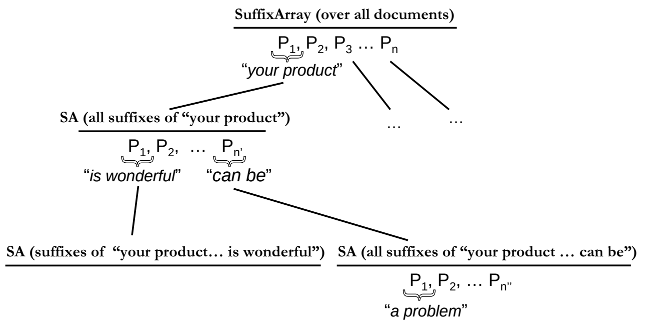

SuffixArray (over all documents)
P1, P2, P3 … Pn
"your product"
…
…
SA (all suffixes of "your product")
P1, P2, … Pn'
"is wonderful"
"can be"
SA (suffixes of "your product… is wonderful")
SA (all suffixes of "your product … can be")
P1, P2, … Pn''
"a problem"

Figure 2: Exploration of the search tree

over suffix arrays containing smaller and smaller subsets of the corpus, corresponding to the suffixes of the patterns found along the paths in the tree. We illustrate this in Figure 2.

One crucial aspect of this search scheme is that – when memory is sparse – we can traverse the search space in a depth-first manner, which in turn requires very little memory. This is due to the large vocabulary size (and the short length of the patterns we use), which results in the suffix-arrays further down the search tree only requiring a small fraction of the size of the initial suffix array.

3.2.4 Exploration of the Search Tree

The approach we describe above is very similar to the one taken in [3], with the key difference being the objective function used to evaluate patterns. While we search for the most frequent patterns satisfying a minimum level of correlation, the work of [3] tries to find the pattern p that (in the case of two labels $\{0, 1\}$) optimizes an evaluation function of the form:

$$G^{\psi}_{\mathcal{D},\mathcal{L}}(p) := \psi\left(\frac{freq(l,p)}{freq(p)}\right)\cdot freq(p) + \psi\left(\frac{|\mathcal{D}|-freq(l,p)}{|\mathcal{D}|-freq(p)}\right)\cdot(|\mathcal{D}|-freq(p)),$$

with ψ denoting any convex *impurity function*, such as *information entropy* or the *gini index*. The key difference between these formulations is that the latter function does not lend itself easily to pruning of the search space when exploring the search tree over suffix arrays described above. Instead, the algorithm of [3] traverses the entire search space.

In contrast, we are only interested in the k most frequent patterns, allowing us to abort any search path that cannot produce patterns of sufficiently high frequency. Due to this, the issue of the order in which we explore the search tree becomes important, since the faster we obtain high-frequency patterns, the smaller the subsequent search space. To implement this pruning functionality, we maintain a list P_found of the k most frequent patterns found thus far that satisfy the condition (1) given in the problem statement. Now, we only need to expand a pattern p if

$$\exists l \in \mathcal{L} : \frac{1}{\alpha} \cdot freq(l,p) > \min_{pf \in P\_found} freq(pf), \qquad (2)$$

for otherwise the expanded pattern cannot satisfy condition (1) and be among the top k most frequent patterns. Due to space-constraints, we omit a further description of the search algorithm.

4. ENABLING ACCURATE REASONING OVER TEXT PATTERNS

When presenting the resulting candidate patterns to a human annotator, one key to the efficiency of this approach is that the annotator is presented with patterns only, but not their sentential context (which would take orders of magnitude longer to review). However, this can be problematic in cases in which the context of a pattern invalidates the way the pattern is interpreted in isolation: consider the the text pattern "*I like your product*". In our sentiment-classifier example this phrase may induce the user to assign a (+) label to the feedback mail; however, if the phrase is in the scope of a negation, such as "*It's not true that I like your product*.", this conclusion would be invalid, even though the negation does not show up as part of the pattern itself. Natural language contains a number of constructs that have this type of effect:

Negation – these may occur either as *external negations* that occur outside a pattern and invalidate an entire pattern (such as in the example above), or as *internal negations* where the word triggering the negation is "hidden" in gap within the pattern itself (e.g. "*this is not great at all*" matching the pattern "*this is . . . great*").

Conditionals – If a pattern occurs as part of a conditional, then the statement implied by the pattern typically does not hold. For example, consider the pattern "*this is . . . a good product*" and the sentence "*If this is such a good product, then why does it crash*?".

Subjunctive – These affect patterns similar to conditionals. Example: the phrase "*were it the case that*" preceding a pattern.

Factives and Speech – These affect patterns similar to conditionals. Example: the phrase "*He claims that*" preceding a pattern.

We refer to the four types of constructs above as *invalidating constructs*. Fortunately, in English the words associated with these types of constructs are known and are independent of a specific training corpus. Thus by using a simple keyword (or key-phrase) matching algorithm, we can identify most sentences containing such constructs, if we are willing to allow false positives.

So, in order to deal with invalidating constructs, we constructed sets of words that can trigger such a construct (e.g. *not, never, cannot*, etc. in case of negation) and use a simple keyword-matching algorithm to identify sentences that potentially may contain an invalidating construct. Now, we extend the definition of *matching* a pattern as follows: a sentence containing an invalidating construct matches a text pattern only if the invalidating construct is part of the pattern itself. For example, the sentence "*don't buy this product*." matches the pattern "*don't buy this*", but not "*buy this product*".

This may result in false positives (i.e. we might flag an invalidating construct either where none exists, or when it does not affect the pattern in question), but since the pattern-based classifier is backed by a machine-learning one, we can afford to be overly conservative.

5. EXPERIMENTAL EVALUATION

In this section we describe experiments which evaluate both the overall efficiency of the hybrid classifiers as well as the reduction of human effort required to obtain a target accuracy.

Datasets: We used two different datasets in our experiments: the first is the *polarity dataset V2.0* available from [1], which consists of 2000 movie-reviews from `rec.arts.movies.reviews`, with 1000 reviews being positive and 1000 negative. This dataset has become the *de facto* standard dataset for sentiment-classification and has been used in over 15 research papers. The second dataset we used is real customer feedback data that is internal to Microsoft. The data set contains more than 1.8 million pieces of individual customer feedback and 118MB of raw text data. The individual messages contain comments on web help documents and are on average significantly shorter and vary more in content than the reviews in the first dataset. Each feedback item is tagged with one of the labels ('*useful*', '*not useful*', '*maybe*'), reflecting the customer assessment of the help document in question.

5.1 Classification Experiments

In these experiments we compare the accuracy of the hybrid classifier to a machine-learning based one given the required amount of human effort (i.e. the number of labeled training documents and text patterns examined by a human). The machine learning method we use is a *Support Vector Machine* (SVM) [19]. SVMs have consistently been shown to outperform other classification algorithms for text classification in general [6, 5], and for sentiment classification in particular [13, 12]. The training algorithm we used is Sequential Minimal Optimization [14]. When training the SVMs, each document is represented as a feature vector, where we varied the feature sets to include (a) all unigrams, or (b) all unigrams, bigrams and trigrams found in the training document set. All features were binary, i.e. only the absence or presence of a feature in a document is indicated, but not its frequency, which is consistent with results in research literature in which binary features outperform frequency features for text classification (e.g. [6, 12]).

5.1.1 Experiments on the Movie Review Dataset

Setup of the Text-Pattern based Classifier: To construct the text-pattern based classifier we first used the algorithm of Section 3.2 to discover the most frequent text patterns satisfying condition (1) (for $\alpha = 0.99$) in the full training data. The 300 most frequent patterns[1] were then presented to 15 different human annotators. Each annotator was given only the patterns, but no information about their frequency or examples of sentences containing the patterns, as we were interested in validating our hypothesis that humans can reason sufficiently well about the patterns themselves. Depending on the annotator, identifying discriminating patterns required 20-40 minutes, with the average being about 30.

Obviously, the quality of the resulting pattern-based classifier varies with the annotator – consequently, all experiments on classification accuracy using such classifiers are plotted with error-bars, with the high/low bar corresponding to the best/worst annotator and the plotted point corresponding to the average over all annotators.

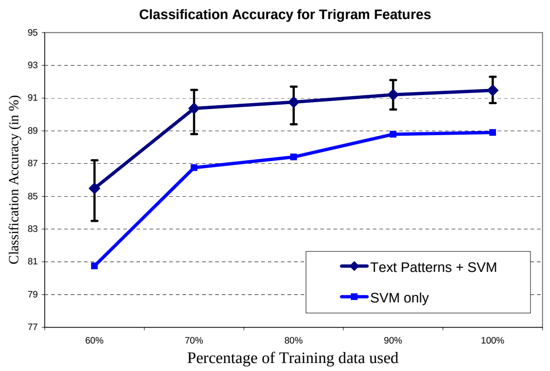
Classification Results: The classification accuracy of the resulting machine-learning based and hybrid classifiers is shown in Figure 3 for varying sizes of the training data. Note that the accuracies of the SVM-baseline compares nicely with other results published on this dataset. For example, [12] reports 87.15% accuracy for a SVM trained on unigrams when using the full reviews as training data, which matches our results almost exactly. However, in addition to the unigram-experiments we also use bi- and trigrams, which increases the classification accuracy significantly, so our baseline without text patterns already outperforms the results given in other papers. All experimental results are based on 5-fold cross-validation.

All experiments show clearly that the hybrid classifier results in a significant boost of the overall classification accuracy when compared to the machine-learning one. This holds true for *all* 15 annotators, indicating that humans can reason accurately over the patterns we provide.

Moreover, in nearly all cases the hybrid classifier also outperformed the machine-learning one using an additional 10% (or – in many cases – even more) training data. Given that 10% of the training data corresponds to 120 documents (of 600-700 words, as opposed to patterns which are about two orders of magnitude shorter), whose annotation almost certainly requires more time than the 30 minutes used for discriminating pattern selection on average, these results also show that using our framework can yield significant reduction in the amount of human effort required.

In addition, we ran some initial experiments where we used the discriminating patterns themselves as binary features in the SVM classifier. Unfortunately, when simply adding these to the feature

[1]We used a minimal amount of pruning here, excluding all patterns that consist to more than 50% of stop-words such as ’*and*’, *the*’, ’*in*’...



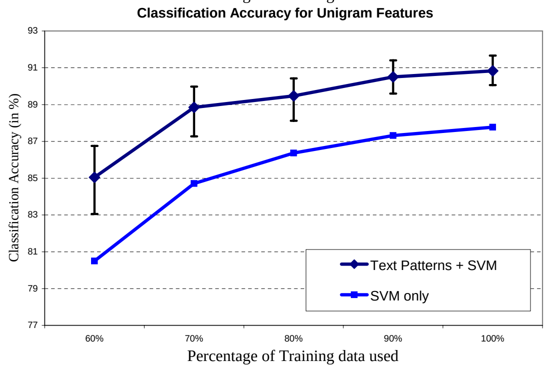


Figure 3: Classification accuracy for the Movie Dataset

set, they are “drowned out” by the other features, since the number of such patterns is extremely small in comparison to the number of n-grams. The alternative of using only the discriminating patterns as features is also not practical, as these only cover a fraction of the data set, i.e. many documents do not contain a single discriminating pattern. None of these findings rule out the use of discriminating patterns as classifier features, or as input to the classifier-construction. For example, techniques such as [20] which incorporate prior knowledge into weighted margin SVMs could potentially leverage discriminating patterns to come up with a better classifier, and we will investigate their use in the future.

5.1.2 Comparison with active learning

Given that our experiments require human interaction beyond labeling, a natural comparison was to *active learning* techniques. Here, we use an active learning approach specifically aimed at SVM-based classification of text documents, using the *Simple Margin* method described in [18] to chose additional documents to label. In order to achieve an apples-to-apples comparison we need to compare techniques that require the same amount of human effort; however, we can only guess at the time required at labeling a movie review. Hence, we conducted multiple experiments, one assuming that a human annotator would require 1 minute to label a review (the reviews average 600-700 words) and another one under the optimistic assumption that the annotator only requires 30 seconds per document. Since labeling the discriminating patterns required 30 minutes of time on average, we thus compare our approach to a machine-learning classifier which uses active learning to add 30 or 60 documents to its training data. The results are shown in Figure 4 – here the x-axis corresponds to the amount of training data used *before* the active learning sets in. Active learning performs comparably to the hybrid classifier for small training data sizes (outperforming SVM-classifiers without active learning using significantly more training data), but does not achieve the performance of the hybrid classifier for larger training sets.

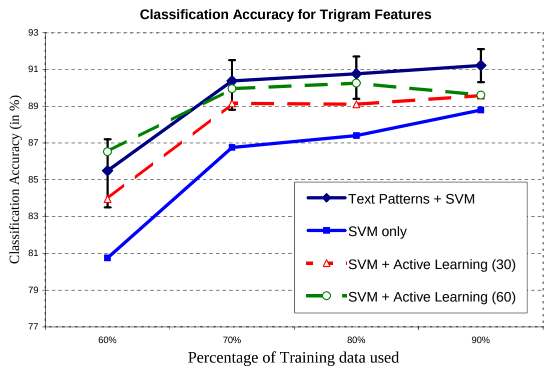


Figure 4: Classification accuracy for the Movie Dataset

5.1.3 *Experiments on the User Feedback Dataset*

To study the effects of text patterns in scenarios where training data is more abundant, we ran similar experiments on the user feedback dataset, with the classification task being to assign the labels '*useful*' and '*not useful*' to the comments. Here, we used between 23K and 32K comments as training data and a test-set of 10K comments. Note that this data is much more heterogeneous than the movie data, and individual items also much shorter, making this classification task somewhat more difficult.

To construct the machine-learning classifier, we again used an SVM model, trained on all uni-, bi- and trigrams occurring in the training data. To construct the hybrid classifier, we presented a human annotator with the 300 most frequent patterns, using a threshold of $\alpha = 0.95$; we only used a single annotator in this experiment. Finally, we also ran active-learning experiments similar to the above, allowing active learning to select 300 comments, to account for these being much shorter than the movie reviews.

The experimental results are shown in Figure 5. Again, investing a short amount of time into pattern annotation gives a boost to classification accuracy, which would have required a large number of additional labeled training data otherwise. Here, the overall improvement (and the classifier accuracy) is lower, as the comments vary more (many of them express specific issues relevant to only a subset of the help pages). As before, the hybrid classifier performs better in relation to active learning as the size of the initial training data becomes larger.

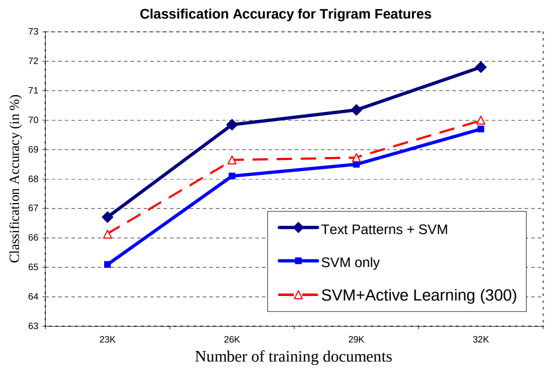


Figure 5: Accuracy for the User Feedback Dataset

6. CONCLUSIONS AND OUTLOOK

In this paper we describe a novel framework to extracts text patterns that are highly correlated to a specific label from an annotated text corpus, among which discriminating patterns are then selected by a human annotator, resulting in text patterns that are highly predictive and hence very valuable in classifier construction.

Our experiments suggest that these text patterns result in significant improvements in accuracy over pure machine-learning approaches. This is in part due to the fact that the text patterns we use are often more complex than the individual features used in text classification (due to the sparseness of text data) and in part due to our combination of text mining and human domain knowledge being able to pick up highly predictive patterns/features, which are hard to properly weigh by machine-learning approaches (if the patterns are part of the feature space), as they are statistically indistinguishable from other, less predictive features in the training data.

7. REFERENCES

[1] Polarity dataset v2.0. http://www.cs.cornell.edu/people/pabo/movie-review-data/.

[2] H. Arimura, H. Asaka, H. Sakamoto, and S. Arikawa. Efficient Discovery of Proximity Patterns with Suffix Arrays. In *Proceedings of Combinatorial Pattern Matching: 12th Annual Symposium*, 2001.

[3] H. Arimura, H. Sakamoto, and S. Arikawa. Efficient Data Mining from Large Text Databases. In *Progress in Discovery Science*, 2002.

[4] P. Beineke, T. Hastie, and S. Vaithyanathan. The Sentimental Factor: Improving Review Classification via Human-Provided Information. In *Proceedings of the 42nd ACL Conference*, 2004.

[5] S. Dumais, J. Platt, D. Heckerman, and M. Sahami. Inductive Learning Algorithms and Representations for Text Categorization. In *Proceedings of CIKM*, 1998.

[6] T. Joachims. Text Categorization with Support Vector Machines: Learning with many Relevant Features. In *Proceedings of the EMNLP Conference*, 1998.

[7] J. Kärkkäinen and P. Sanders. Simple Linear Work Suffix Array Construction. In *Proceedings of 13th International Conference on Automata, Languages and Programming*, 2003.

[8] B. Lui, X. Li, W. S. Lee, and P. S. Yu. Text Classification by Labeling Words. In *Proceedings of the 19th National Conference on Artificial Intelligence*, 2004.

[9] U. Manber and G. Myers. Suffix Arrays: A new Method for On-Line String Searches. In *Proceedings of the First Annual ACM-SIAM Symposion on Discrete Algorithms*, 1990.

[10] E. M. McCreight. A Space-Economical Suffix Tree Construction Algorithm. In *Journal of the ACM, 23*, pages 262–272, 1976.

[11] K. Nigam, A. McCallum, S. Thrun, and T. Mitchell. Learning to Classify Text from Labeled and Unlabeled Documents. In *Proceedings of AAAI*, 1998.

[12] B. Pang and L. Lee. A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts. In *Proceedings of the 42nd ACL Conference*, 2004.

[13] B. Pang, L. Lee, and S. Vaithyanathan. Thumbs Up? Sentiment Classification using Machine Learning Techniques. In *Proceedings of EMNLP*, pages 79–86, 2002.

[14] J. Platt. Fast Training of SVM's Using Sequential Minimal Optimization. In *Advances in Kernel Methods: Support Vector Machine Learning*, pages 185–209. MIT Press, 1999.

[15] H. Raghavan, O. Madani, and R. Jones. InterActive Feature Selection. In *Proceedings of IJCAI-05*, pages 841–846, 2005.

[16] M. Sahami. *Using Machine Learning to Improve Information Access*. PhD thesis, Stanford University, 1998.

[17] H. Seung, M. Opper, and H. Sompolinsky. Query by Committee. In *Proceedings of Computational Learning Theory*, 1992.

[18] S. Tong and D. Koller. Support Vector Machine Active Learning with Applications to Text Classification. *Journal of Machine Learning Research*, 2001.

[19] V. Vapnik. *Statistical Learning Theory*. Whiley, 2000.

[20] X. Wu and R. Srihari. Incorporating Prior Knowledge with Weighted Margin Support Vector Machines. In *Proceedings of KDD'04*, pages 326–333, 2004.

[21] M. Yamamoto and K. W. Church. Using Suffix Arrays to Compute Term Frequency and Document Frequency for All Substrings in a Corpus. In *Computational Linguistics, volume 27 (1)*, 2001.