Using Feature Construction to Avoid Large Feature Spaces in Text Classification

Elijah Mayfield
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA 15213
elijah@cmu.edu

Carolyn Penstein-Rosé
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA 15213
cprose@cs.cmu.edu

ABSTRACT

Feature space design is a critical part of machine learning. This is an especially difficult challenge in the field of text classification, where an arbitrary number of features of varying complexity can be extracted from documents as a preprocessing step. A challenge for researchers has consistently been to balance expressiveness of features with the size of the corresponding feature space, due to issues with data sparsity that arise as feature spaces grow larger. Drawing on past successes utilizing genetic programming in similar problems outside of text classification, we propose and implement a technique for constructing complex features from simpler features, and adding these more complex features into a combined feature space which can then be utilized by more sophisticated machine learning classifiers. Applying this technique to a sentiment analysis problem, we show encouraging improvement in classification accuracy, with a small and constant increase in feature space size. We also show that the features we generate carry far more predictive power than any of the simple features they contain.

Categories and Subject Descriptors

I.2.7 [**Natural Language Processing**]: Text analysis

General Terms

Algorithms

Keywords

feature space design, text classification, sentiment analysis

1. INTRODUCTION

Supervised learning algorithms are able to learn to predict class values based on stable patterns in vector representations of labeled examples [24]. However, they are only able to do this if there are stable patterns to be found. Thus, the representational power of the features that make up the feature space is a major limiting factor in their effectiveness. Since the mid-90s, there has been work in the Genetic Programming community on using genetic search to evolve powerful features to be used in supervised learning approaches. The complex features that are evolved have the potential to have far more predictive value than the primitives that make them up. An excellent review of recent work on this problem is presented in [21]. In this paper, we present a novel approach to evolutionary feature creation and selection for text classification problems.

Text classification has been an active research area in the field of natural language processing for decades [25]. Typically, supervised learning approaches are used. Very common approaches include Naive Bayes, Decision Trees, and Support Vector Machines (SVM) [9]. Normally, a large number of labeled training example documents are collected into a set (termed a corpus). In order to apply typical supervised learning approaches to these corpora, each document must first be transformed into a vector representation, which is an ordered set of feature-value pairs. Early work in text classification utilized very simplistic vector representations where the features each corresponded to individual words, termed unigrams. These simple vector representations were effective for simple, topically oriented text classification tasks, such as identifying which newsgroup a post was from [18] or classifying webpages as being faculty or student pages [7].

Recent work in text classification has focused on more sophisticated problems, such as sentiment analysis, where the goal of the classification task is to assess the attitude that is conveyed by a text. For problems like these, low level features like unigrams are notorious for not generalizing well, and thus the models are often evaluated with very poor performance. On the other hand, if even moderately more complex features are added to the feature space, such as the class of features that describe grammatical relationships that might hold between pairs of words within a sentence, the size of the feature space drastically expands, which frequently reduces rather than improves the effectiveness of the feature space representation [10]. At the same time, moderately more complex features such as these do very little to improve the representational power of the feature space. What is needed is dramatically more complex features. However, if even expanding to moderately more complex features is on the edge of computational feasibility, then expanding to this greater level of sophistication is clearly beyond reach.

We argue in this paper that Genetic Programming offers a unique and powerful solution to this problem. The genetic programming paradigm allows us to search a very large space of possible new features very efficiently, strategically choosing a few very powerful features from this space. Genetic search offers us control over how much exploration we do in order to find these choice features. We then have control over how many features we add to our baseline feature space. Thus, we are able to search for very sophisticated features without being committed to a computational explosion either in terms of time to search for potential features or in the size of the feature space we ultimately use in our supervised learning text classification approach.

In the remainder of the paper, we first review related work in the genetic programming community on feature creation. We then detail our technical approach to feature evolution and our experimental framework. Next we describe how we have explored the space of potential fitness functions in order to optimize the effectiveness of our feature evolution approach. Finally we discuss our promising results and directions for continued research.

2. RELATED WORK

We situate our work most closely with the extensive body of prior work related to evolutionary approaches to feature creation and selection [21]. An application of this approach to computer vision was discussed in [4], and another application to chromatography was shown to be effective in [19]. Genetic algorithms are commonly used for the feature selection process when using an evolutionary approach to feature construction. The results from [11] and [15] are most similar to our approach in that they use genetic programming to build trees as feature representations.

The key difference in our work from this past work is the nature of the data being classified. We are applying Genetic Programming to a language processing task. Attempts to use genetic programming for natural language processing are uncommon, although early applications to language processing tasks can be found as early as our work using GP to aid in recovery from parser failure in speech-to-speech machine translation in the late 90s [20]. Lately, there has been a large body of work for applying genetic programming to query generation for information retrieval [6], in uses related to ours, such as query expansion [1] and tree-based genetic programming for query generation [22].

To our knowledge, the most similar work to ours in language processing is [8], which attempted to classify newswire documents based on their topic using complex boolean search queries. There are several notable differences between their task and ours, the most important being that their system was designed to act as a fully-fledged classifier. In our work, on the other hand, we will be building features in a similar style to those used in their work, but with much shallower trees, where each evolved tree is just one feature, in other words one piece of evidence a trained classifier will use. Another difference between their work and ours is that we use whole word n-grams, i.e., sequences of whole words, as terminals, while the prior work uses character n-grams, i.e., sequences of characters.

Much of the prior work in feature construction makes use of purely numeric data; most (though not all) optimize their performance for use in Decision Tree learning. Decision trees typically perform best in tasks where there are a few very strongly predictive features within the feature space, which can be combined in a powerful way within the learned trees. Text classification differs from these tasks in key respects. Normally, text classification tasks involve very large feature spaces made up almost entirely of very weak predictors. Because of this, Decision Tree learning algorithms tend not to perform nearly as well as SVM, which has been designed specifically to be able to combine large spaces of weak predictors without overfitting. The main disadvantage of standard SVM, with a linear kernel, is that it is not able to represent interactions between features, which Decision Trees are able to represent very naturally, since each decision is conditioned upon the series of decisions leading up to it.

Support vector machines have been shown to excel at the problem of text classification based on their ability to aggregate evidence from numerous weakly predictive features. In order to process a document using SVM, that document must be transformed into a vector of numeric attributes. The most common way to do this is by way of a “bag of words” approach. In this representation, a vector is created with one attribute for each word that could possibly occur in a document. For each word that actually does occur in that document, the corresponding attribute is set to a value of 1; all other attributes are set to a value of 0. This is also known as a *unigram model*[1].

Interactions between simple features are useful to capture in language processing tasks. For example, consider that if each word in a sentence is treated as an independent piece of evidence, sentences like “The dog bit John.” and “John bit the dog.” would be treated as identical. What distinguishes them is the grammatical role the dog and John each play. Grammatical relations are a specific type of interaction between features, although this type of interaction is more powerful than what you get with higher order polynomial kernels in SVM. These kernels are only able to capture the fact that pairs or sets of features occurring together means something particular, which is distinct from the occurrence of each feature on its own. In the absence of grammatical features, grammatical analysis can be approximated using what are known as bigram features, which are pairs of words that appear next to one another in the text, for example “dog bit” in the example above. This simple way of approximating grammatical analysis can only go so far, however, For example, “The dog bit the other dog that John pet.” could not be distinguished from ”The other dog bit the dog that John pet.” using these features. Furthermore, bigram and higher order ngram features cannot be captured directly in standard higher order polynomial kernels. Instead, either the feature space must be elaborated with more powerful features, or specialized structural kernels must be used that are capable of capturing these richer interactions. Structural kernels such as those proposed in early work [5] and elaborated on in recent years [14] have shown promise, and such feature space enhancement is feasible in an SVM based approach, and is in fact common in many language processing applications. However, either approach can easily lead to a computational explosion, as we will illustrate below.

Some text classification tasks can benefit from the simpler types of interactions between unigram features that higher order polynomial kernels can capture. For example, consider a newsgroup classification task. A post that mentions

[1]It is important to make the distinction here that unlike many tasks where an n-gram refers to a series of n characters in sequence, the meaning in the natural language processing community generally refers to a series of n consecutive words in a document.

dolphins could be either for a newsgroup about animals or a newsgroup about football. And a post that mentions football may be from a newsgroup related to any team. However, a post that mentions dolphins and football is likely to be for the Miami Dolphins newsgroup. In this case, the simple interaction between features is sufficient. However, sentiment analysis requires more sophistication. Research in the field of Systemic Functional Linguistics [13] illustrates how attitude is encoded in language using syntax and cue phrases in strategic combination. For example, in the sentence, "Naturally I recommend sweetening." as a comment on a cooking blog makes a much stronger recommendation than "I recommend sweetening naturally." In the first case, "naturally" is a sentential modifier that serves as a persuasive device, whereas in the second sentence it only describes a manner of sweetening.

From this line of reasoning, we can draw a number of conclusions. First, increasing the complexity of the kernel for SVMs is not a viable solution for achieving the level of representational power sophisticated languague processing tasks require. The feature space for non-trivial text classification tasks is already too large even with just unigrams to feasibly apply a Decision Tree learning approach. And even there, the representational power is not sufficient for making some of the types of distinctions that are desired, especially at the sentene level. Thus, adding more sophisticated features to make desired distinctions is what is required.

Initially, this added level of detail is very attractive. However, it comes at a high price. For example, consider the corpus from [16], a collection of 2,000 relatively short documents (several hundred words each, on average). This small collection contains 65,044 unique unigrams, and 500,137 unique bigrams[2]. This order of magnitude difference in feature space size has a huge impact on the effectiveness of machine learning classifiers, almost always detrimental. Because of the sparsity of useful predictive words for any given classification task, the signal is drowned out in noise. Adding even more complex features quickly leads to an undesirable computational blow up.

3. FEATURE CONSTRUCTION

Our goal is to construct features which can capture information that a unigram model cannot, without a significant change to the size of the feature space. We do this by combining unigrams into boolean query-like statements. These combinations allow already-present information to be formulated and recombined in ways that bring to light clues that unigrams alone would not have uncovered. In this section, we first detail the task that we are using as an example application of our GP features. We then describe the structure of individual features as we are generating them. We describe our full process for generating features and combining them with a unigram model for our classification task, and then we conclude this section with a high-level description of our fitness function and the choices that were made in its design.

3.1 Task description

Our experiments deal with the problem of sentiment analysis, which is a broad class of problems related to detection of subtle shades of meaning such as attitude, emotion, perspective, or even personal identification. A nice review of work related to sentiment analysis in the computational linguistics community can be found elsewhere [17]. Our experiments in this paper focus on the task of determining whether the author of a movie review recommends a movie or not. Other example tasks include opinion mining of product reviews or trend analysis in on-line political discussions. We used the corpus from [16] to test our system's ability to perform this task. This corpus contains 2,000 movie reviews collected from an online review aggregator, comprised of 1,000 positive reviews and 1,000 negative reviews.

To provide an understanding of the body of work using this corpus, we present below results that others have presented using different approaches on this data set. A direct comparison between results reported in other work on this data is not possible since no standard evaluation methodology has been used, and thus the performance numbers are not directly comparable. For example, there is a discrepency between authors on the size of the training set used (70%, 80%, or 90%) and method of evaluation (using a held out set, or using cross validation). Therefore, we do not report performance improvements, only methods used, to give context to our work.

What sets our approach apart from prior work on sentiment analysis with this corpus is that our approach is fully automatic, inducing the features empirically from the data itself, whereas the other approaches require human annotation and a pre-existing lexicon of polarity words. Thus, it is not surprising that our results do not rise to the level reported in those papers. Nevertheless, we see our work as one important step towards replacing that manual effort.

An attempt to perform sentiment analysis using advanced features crafted by expert linguists is presented in [2]. This paper drew on theoretical concepts in the field of systemic functional linguistics, a theoretical framework which is designed to model author intent for communication and discourse between speakers [13]. Using this framework, an expert built a hierarchical lexicon of words specifically representative of the rhetorical devices formalized in that theory, and scored documents using a formula based on that lexicon.

A more traditional machine learning approach was taken by [23]. This work developed a voted combination of several machine learning classifiers, including SVM, a maximum entropy model, and a polarity scoring model based on a lexicon of positive and negative words. While it is unclear from their work whether this scoring method was fully automated, it appears that it required at least some human intervention to build the polarity lexicon.

Incorporation of even more explicit human effort into the machine learning process was taken by [26]. This work asked human annotators to read reviews and mark the "rationales" for a review's polarity - segments of the reviews that they judged important in determining the polarity of the document. Machine learning was then performed using only features extracted from the highlighted segments. This could be considered a manual form of feature selection.

While there have been efforts to reduce annotation costs, such as the work presented in [3], this is still impractical for large-scale systems. We consider our work a preliminary step towards building fully automated systems which can extract more detailed knowledge from texts.

[2]Incidentally, this increase in feature space size does not stop at bigrams. The same corpus contains 1,063,137 unique 3-grams and 1,347,584 unique 4-grams.

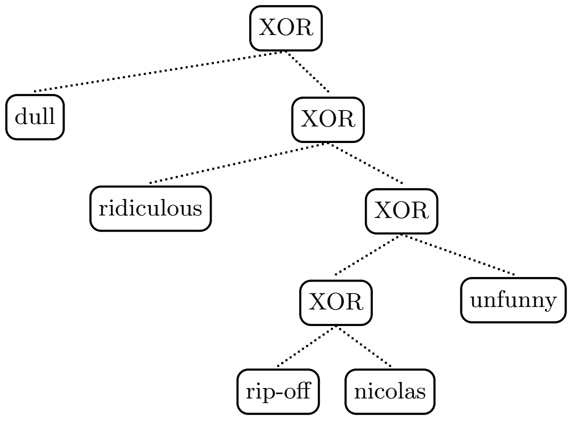

Figure 1: A simple example of a generated feature.

3.2 Design of Constructed Features

Our features are equivalent to boolean statements describing the presence or absence of unigrams in a document. We use two operators, AND and XOR. Experiments with a larger selection of boolean queries, such as NOT AND or an inclusive OR, did not appear to be more effective, so for simplicity we constrict our options to these two. For some insight into the process, it is best to look at example features. Consider a feature like:

(AND best (XOR thrilling subtle))

This feature is going to appear in two sets of movies which will rarely overlap - subtlety is rare in a thrilling movie, and vice versa - but either may still be describing a positive opinion. Alternatively, consider a feature such as:

(XOR diminished ambitious)

This is a subtree from a positive feature. The term "ambitious" is highly predictive when it occurs - it is a positive adjective. On the other hand, "diminished" is rarer, and is not strongly predictive of either positive or negative reviews. However, consider this portion of a review:

> this is a promising premise, and mr. taylor's film could have gone any number of ambitious ways from this point [...however] this is not a film which has the wherewithal to kill off its leading star in the opening ten minutes. the entire sequence is, then, clearly an exercise for character exposition, with attempts at humour terribly diminished by utter predictability .

This is a negative review which is ruled out by this feature's subtree. This style of feature occurs occasionally in our generated features - a common and predictive feature is paired with a rarer and less predictive feature, which further refines the predictive feature.

These features do occur in our system and appear to be effective, but they are rare. Frequently, we see features similar to the one shown in Figure 1. These features often come down to being nearly equivalent to simple lists of positive or negative words, with little interaction between them. However, even this carries descriptive content that is more concise and powerful than any one of the words alone.

3.3 Feature Construction Workflow

Our process of feature construction and evaluation is shown in Figure 2. The high-level process can be understood by first breaking documents down into unigram models, stripping out infrequent words to make the feature space more manageable and removing noise. From these models we can build a list of all possible candidate words for terminal nodes in GP trees. We construct features based on these lists using boolean operators, and build a set of features to be added onto our unigram space. Each of these features is then also converted to a binary form (equaling 1 if the boolean feature is true of the corresponding document). This combined feature space is then passed to an SVM classifier, which is trained on both the unigrams and GP features.

For our genetic programming module we use the ECJ toolkit [12]. Our population is comprised of 2,048 individuals, initiated using ramped half-and-half. For a selection function we use 7-individual tournament selection, with subtree crossover and subtree mutation equally likely to occur during breeding. As a simplifying assumption, we assert that alterations to the selection, mutation, and crossover settings are not likely to affect the performance outcome of our approach. Each run ends after 15 generations. This GP stage is repeated 15 times, with a different seed for random number generation in each run, in order to develop heterogenous features. Each GP run is repeated once for each possible value of the class we are trying to classify; the top individual from each variant of each run is selected as a feature to be appended to our baseline model. For a binary classification problem, therefore, our approach generates 30 new features. Once the combined feature space is built, we use SVM_{light} [9] for all experiments.

A systemic problem in genetic programming is increasing solution complexity over time. Often, we see in evaluation that the features that we generate reached a maximum performance at an early generation and then overfit, potentially decreasing performance on held-out data. Therefore, we must determine some way of choosing the best generation stopping point for evaluating our features. In this paper, our overall results are given by selecting, for each fold in cross-validation, the generation that provides the most improvement to our unigram model on the data that the features were trained on - note that this still makes no use of held-out data, to ensure that we maintain the complete separation between data used for feature generation and classifier training and our held-out test data. The actual performance of our system on held-out data is guaranteed to be lower than this, but we observe that this gives us an adequate way of avoiding one potential source of overfitting.

3.4 Defining fitness

The primary challenge that we address in our experiments centers on how to define the fitness function for these individuals. This is not a straightforward problem, and many different factors could conceivably be introduced into the equation. Details of our implementation of some of these factors are given in Section 4; we now give a high-level understanding of our goals in choosing the factors that we did.

Our definition of fitness is based on the concepts of precision and recall, borrowed from information retrieval. This balances the tradeoff between the predictive value of a feature (how likely it is to appear in one class of documents, compared to another) and the rarity of the feature (features

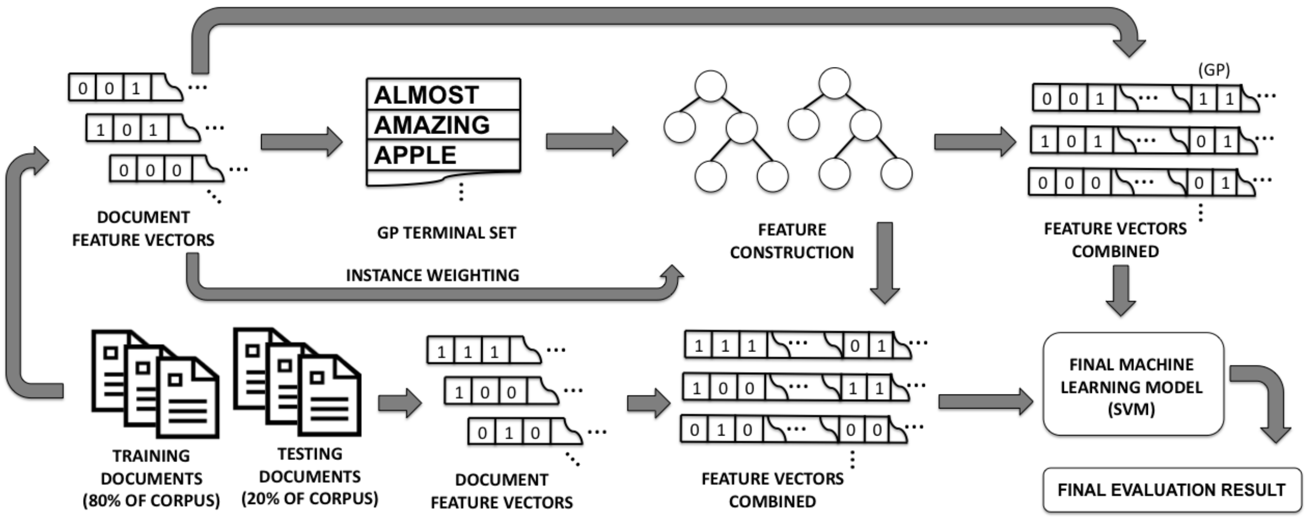

Figure 2: Workflow diagram of our feature generation process. For evaluation, this process is followed five times, once for each fold of cross-validation.

which are highly predictive but almost never occur are of little value). These two factors must be balanced in a way that makes sense for the problem at hand. The unigrams that are competing for weight against our constructed features are usually sparsely distributed throughout training data, so low recall is not a significant drawback. We therefore want to set precision to be significantly more heavily weighted than recall.

In addition to these measures, there are a number of penalties that we can add. This includes penalizing overly complex trees, which are more likely to overfit to training data; adding a constraint to fitness based on correlation with unigrams, which would encourage exploration of new information above and beyond that captured by unigrams alone; finding the most difficult training examples and weighting them more heavily in our fitness function; and last, avoiding overfitting by building our features based on only a portion of the examples in our training data. With this high-level understanding of our goals for the fitness function, we can now describe the ways in which these various penalties were implemented.

We define our set of documents as being comprised of a set of positive documents $P_1, P_2, ...P_u$ and a set of negative documents $N_1, N_2, ...N_v$. For a given individual I and document D, we define $hit(I, D)$ to equal 1 if the statement I is true of that document and 0 if it is not. Precision and recall of an individual feature for predicting positive documents[3] is then defined as follows:

$$Prec(I) = \frac{\sum_{i=0}^{u} hit(I, P_i)}{\sum_{i=0}^{u} hit(I, P_i) + \sum_{i=0}^{v} hit(I, N_i)} \tag{1}$$

[3]Negative precision and recall are defined identically, with obvious adjustments to test for negative documents instead of positive.

$$Rec(I) = \frac{\sum_{i=0}^{u} hit(I, P_i)}{u} \tag{2}$$

We then weight these values to give significantly more importance to precision, using the F_β measure, which gives the harmonic mean between precision and recall:

$$F_\beta(I) = \frac{(1 + \beta^2) \times (Prec(I) \times Rec(I))}{(\beta^2 \times Prec(I)) + Rec(I)} \tag{3}$$

To constrain the size of features being generated, we follow the lead of [15] in penalizing trees based on the number of nodes they contain, rather than setting a maximum depth. This penalty is labeled P.

We define the correlation between two prediction vectors using Pearson's product moment correlation. For two vectors X, Y with elements $\mu_1, \mu_2, ...$ and with standard deviations ρ_x and ρ_y, the correlation is defined as:

$$\rho_{x,y} = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y} \tag{4}$$

This equation results in a value between 1 (representing perfect alignment) and -1 (representing completely inverse alignment). Our penalty for correlation with a unigram is equal to the extent over a preset threshold that the generated feature correlates with any unigram it contains. This penalty is labeled C in our results.

We identify problem areas in our training data by cross-validation on that training data. By this process, each training document is classified exactly once, and in our experiments, around 15% of the documents are classified incorrectly. These documents can therefore not be our only source of fitness evaluation - the set is too small to attempt to match while expecting wide coverage. We solve this simply by weighting instances labelled incorrectly in cross-validation as being twice as important as other training documents -

it is assumed that this doubling formally happens at some stage prior to fitness calculation. This is not a penalty in the same sense as the other two, but for consistency in labeling, our results table shows this addition as W. We can label an F_β-measure that takes into account this instance weighting as F_w.

Finally, we know that the features we construct are naturally going to fit very well to training data, and may not generalize to other data sets. What we would want to do ideally is test each of our features against some validation set, separate from the set of data that is being trained on. However, our data set is not infinitely large. What we can do instead is to construct the features based only on their performance on a subset of the training data. In this configuration, precision and recall values, along with correlation penalties, are altered to only measure a feature over half of our training set.

Intuitively this would suggest that the features we gain are more likely to overfit, not less, because they are being trained on less data. However, because we are still training our SVM model on the entire 1,600 document training set, only those features which are more loosely fit to the training data there were induced from will generalize to the other half of the set. This results in those more generalizable features being given higher weight in the classifier. This can be used as a verification that the features being given high weight are not overly idiosyncratic to the variance of the training set. In our fitness function, we label F-measure which uses this training set division as F_h, and in our results tables, this is listed as H.

Combined, these aspects of our fitness function give a final equation of:

$$\text{Fitness} = F_{wh} + P + C \tag{5}$$

4. EXPERIMENT DESIGN AND RESULTS

We now need to test the ability of our fitness function to find powerful new features. To determine how promising the features that we describe are, we present an application of this approach to sentiment analysis, and show two results of that application. The first demonstrates that with a well-defined fitness function, our features can improve the quality of a unigram model with very little change in feature space size. We also show that our set of evolved features are more powerful than the component parts they are constructed from, by restricting that space to the same size through χ^2 feature selection.

For all experiments we performed 5-fold cross validation. In each fold, we take as input a corpus of documents and partition it into a training set (comprising 80% of the documents, evenly divided between all possible classes) and a testing set for evaluation (consisting of the remaining 20% of the documents). We report on results for each incremental part of our fitness function being included, to show what effect different options have on performance.

4.1 Experimental parameters

One question which we do not address here with regards to the model we have described is one of tuning. There are several parameters that are held constant in our experiments. The penalty of parsimony pressure is set to 0.005 per node. This penalty is approximately equal to the accuracy gain of three correctly classified documents. For example, if an individual changes by extending the depth of one branch by one level (adding two new leaves), it would have to correctly classify roughly six additional documents to make up for the penalty. Our correlation threshold is set to 0.5. We use this threshold such that a constructed feature with a 0.8 correlation would have its fitness penalized by 0.3. The β ratio for F-measure is set to $\frac{1}{8}$ for configurations with fitness based on all training data, and $\frac{1}{6}$ for configurations based on only half of the training data (labeled $+H$ in our results).

Features	**Accuracy**	**Δ**	**Oracle**	**Δ**
unigrams	85.5	-	-	-
GP	86.0	+0.5	87.2	+1.7
GP+C	86.1	+0.6	86.65	+1.15
GP+CW	86.0	+0.5	86.65	+1.15
GP+CH	86.25	+0.75	87.25	+1.75
GP+CWH	86.3	+0.8	87.15	+1.65

Table 1: Results for different configurations of GP features added to a unigram feature space, averaged over 5-fold cross validation.

4.2 Results

Results of our experiments are detailed in Table 1. To quantify our performance we give a simple percent accuracy - the percent of classifications made by our system on held-out reviews that were the same as the actual classification of those reviews. For evaluation, we also make use of Tukey's significance test to determine what differences we've made that are statistically significant. We report two numbers for each GP configuration. The first is the performance where the number of generations to stop at is decided by performance on training data. This decision is made for each fold of cross-validation separately, and the average of the five folds is given. This is the most stringent evaluation possible on our performance, and our results do not show a statistically significant difference from unigrams. The second number that we report is the best observed result for each fold on test data - we refer to this as our oracle result. This performance change is statistically significant but acts for us solely as an upper bound on the improvement that can be gained from our features, and not as a fully realized result.

In addition to measuring the performance of our features when added to a unigram model, we would also like to know the strength of the features individually (with a feature space size of only 30 features). It is known that in general, an individual unigram is a very weak predictor of a document's classification. We would like to know to what extent our constructed features are more predictive than unigrams. To test this, we compare performance of our evolved features with performance of the 30 most predictive unigrams, selected using χ^2 feature selection. In addition to this measure, we also compare our performance against the top 30 most predictive features when bigrams are also available for selection. The results of this comparison are shown in Table 2. The difference here is very clearly significant.

5. DISCUSSION

Figure 3 provides some insight into the performance of our features across generations. We see high variability be-

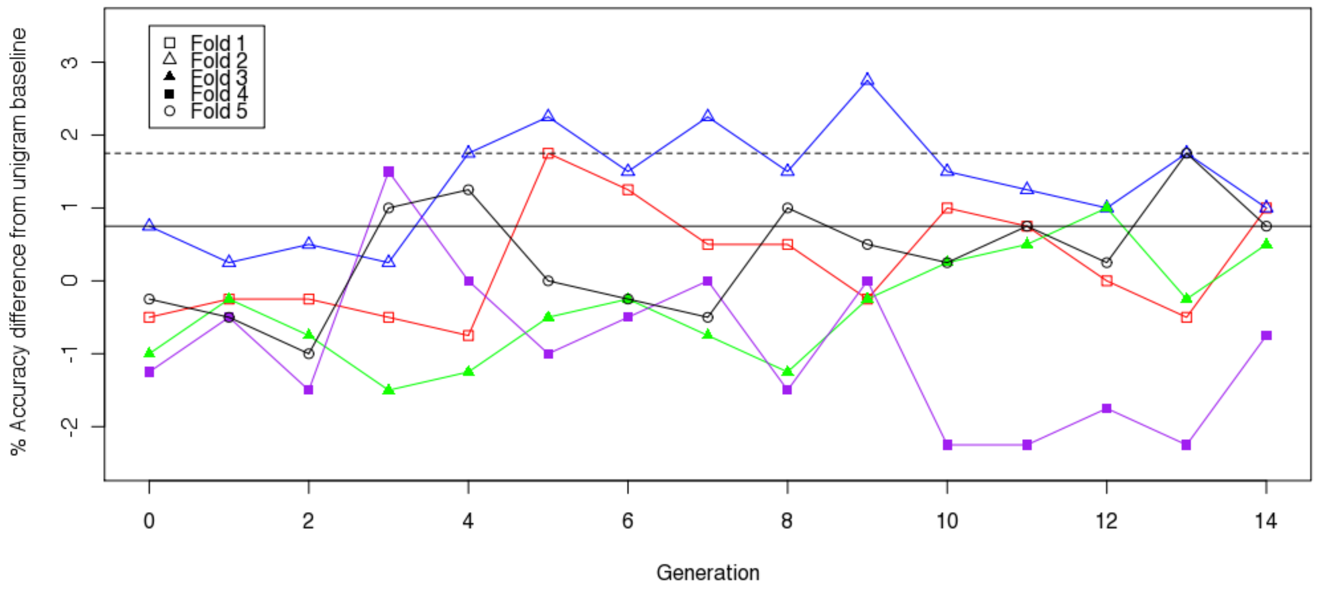

Figure 3: Results for GP features in the GP+CH configuration. Each line graph represents one fold's performance in each generation. The solid horizontal line represents the tuned performance as reported, and the dashed line is the theoretical "oracle" performance. To allow folds to be compared directly, each fold's performance is given as the difference from the unigram baseline for that fold.

Features	**Accuracy**	**Δ**
unigrams	56.25	-
unigrams + bigrams	61.6	+5.35
GP	69.0	+13.75
GP+C	68.95	+13.7
GP+CW	67.9	+11.65
GP+CH	71.8	+15.55
GP+CWH	69.0	+13.75

Table 2: Accuracy of GP features alone, compared to baseline models held (through χ^2 feature selection) to exactly 30 features.

tween generations, and we also see that in many cases, performance deteriorates compared to unigrams. While each fold provides performance improvements at some point, the time at which this improvement is reached differs - improvement comes early and then is reduced in folds 1 and 4, while folds 3 and 5 show a more steady and gradual improvement. However, we are able to consistently and fully automatically choose stopping points that avoid these negative generations. What is likely to be more interesting is what is different between the sets of features that are consistently better or worse than unigrams. For instance, performance of the GP features with no fitness constraints (labeled GP in Table 1) was highly variable between folds, with one fold performing significantly better than others. However, when the GP features generated are compared individually, without being added to a unigram space, this fold was not notably different from the other folds for that configuration. Part of this can be attributed to the large search space - with many different shapes and sizes of trees and the large number of potential terminal unigrams, it is common for genetic programming to become trapped in a local maximum. However, this is not the whole story behind the performance we observe.

What this means is that there is some aspect of the text that is being captured by the features in that fold that is not represented by unigrams. It is exactly this that we would like to capture with our features, and our approach presented here does that in some cases. One very important aspect of future work, however, centers on finding what is different between the features which contribute new information to the unigram space, and those which, while highly predictive on their own, add no new predictive power to the model as a whole. One implication of this is that our performance is likely to improve dramatically if our fitness function used some way of taking into account the contribution that each unigram feature makes to the baseline feature space.

6. CONCLUSIONS AND FUTURE WORK

We have shown that a unigram feature space can be improved by introducing new constructed features, which rely on no other source of information. We provided a process for doing this through use of genetic programming, building features which resemble boolean queries of documents. The resulting features are far more predictive and wider in coverage than unigrams alone are, and are with proper constraints, are not subject to overfitting. This complexity has been added with a mere 30 features added to the baseline feature space for machine learning, compared to the hundreds of thousands of features that are added with a blanket addition of high complexity features. These results are very promising and suggest that with further effort, genetic programming applications in text classification may be competitive with or superior to human annotated results.

We are also limiting our performance greatly by the terminal set and function set that we are using. While this reduces search space size, meaning it is useful as a proof of concept, the information that we are rearranging is already very degenerated from the documents that we are classifying. That we see improvement means that genetic programming can be used to gain more information from these simple features; however, a deeper representation of the meaning in the text would give the potential for more significant gains. Work on more complex features such as graphical models of sentences [10] is likely to provide a more powerful base for describing the content of a text. We also intend to explore the use of features representing rhetorical structure and discourse-level actions that an author can take, utilizing existing linguistic theory.

Acknowledgements

This research was supported by NSF Grant HCC-0803482.

7. REFERENCES

[1] L. Araujo and J. Pérez-Aguéra. Improving query expansion with stemming terms: A new genetic algorithm approach. In *Evolutionary Computation in Combinatorial Optimization*, 2008.

[2] S. Argamon, C. Whitelaw, P. Chase, S. Hota, N. Garg, and S. Levitan. Stylistic text classification using functional lexical features. *Journal of the American Society for Information Science and Technology*, 2007.

[3] S. Arora and E. Nyberg. Interactive annotation learning with indirect feature voting. In *Proceedings of NAACL-HLT*, 2009.

[4] J. Bala, K. D. Jong, J. Huang, H. Vafaie, and H. Wechsler. Using learning to facilitate the evolution of features for recognizing visual concepts. In *Evolutionary Computation*, 1997.

[5] M. Collins and N. Duffy. Convolution kernels for natural language. In *Advances in Neural Information Processing Systems 14*, 2001.

[6] O. Cordon, E. Herrera-Viedma, C. Lopez-Pujalte, M. J. Luque, and C. Zarco. A review on the application of evolutionary computation to information retrieval. 2003.

[7] J. Furnkranz, T. Mitchell, M. Mitchell, and E. Riloff. A case study in using linguistic phrases for text categorization on the www. In *AAAI/ICML Workshop on Learning for Text Categorization*, 1998.

[8] L. Hirsch, R. Hirsch, and M. Saeedi. Evolving lucene search queries for text classification. In *Proceedings of GECCO*, 2007.

[9] T. Joachims. Making large-scale support vector machine learning practical, 1998.

[10] M. Joshi and C. Penstein-Rosé. Generalizing dependency features for opinion mining. In *Proceedings of the ACL-IJCNLP*, 2009.

[11] K. Krawiec. Genetic programming-based construction of features for machine learning and knowledge discovery tasks. In *Genetic Programming and Evolvable Machines*, 2002.

[12] S. Luke. Ecj: A java-based evolutionary computation research system. `http://cs.gmu.edu/ eclab/projects/ecj/`.

[13] J. Martin and P. White. *The Language of Evaluation: The Appraisal Framework*. 2005.

[14] A. Moschitti. Efficient convolution kernels for dependency and constituent syntactic trees. In *Proceedings of the 17th European Conference on Machine Learning*, 2006.

[15] F. Otero, M. Silva, A. Freitas, and J. Nievola. Genetic programming for attribute construction in data mining. In *Proceedings of GECCO*, 2002.

[16] B. Pang and L. Lee. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the ACL*, pages 271–278, 2004.

[17] B. Pang and L. Lee. *Opinion Mining and Sentiment Analysis*. 2008.

[18] J. Rennie, L. Shih, J. Teevan, and D. Karger. Tackling the poor assumptions of naive bayes text classifiers. In *Machine Learning*, 2003.

[19] O. Ritthoff, R. Klinkenberg, S. Fischer, and I. Mierswa. A hybrid approach to feature selection and generation using an evolutionary algorithm. 2002.

[20] C. Rose. A genetic programming approach for robust language interpretation. In *Advances in Genetic Programming, Volume 3*, 1999.

[21] M. Smith and L. Bull. Genetic programming with a genetic algorithm for feature construction and selection. In *Genetic Programming and Evolvable Machines*, 2005.

[22] M. P. Smith and M. Smith. The use of genetic programming to build boolean queries for text retrieval through relevance feedback. 2003.

[23] K. Tsutsumi, K. Shimada, and T. Endo. Movie review classification based on a multiple classifier. 2007.

[24] I. Witten and E. Frank. *Data mining: practical machine learning tools and techniques with Java implementations*. 2002.

[25] Y. Yang and J. Pedersen. A comparative study on feature selection in text. 1997.

[26] O. F. Zaidan and J. Eisner. Using "annotator rationales" to improve machine learning for text categorization. In *Proceedings of NAACL-HLT*, 2007.