Learning Models of Users’ Informational Needs

Shawn R. Wolfe
School of Engineering
University of California Santa Cruz
Santa Cruz, CA, USA
srwolfe@soe.ucsc.edu

ABSTRACT

Most current models of users’ informational needs capture only the content, e.g., search terms, and ignore other aspects, such as metadata. However, a recent study has shown that in some cases, performance suffers dramatically when these additional aspects are ignored. But how can more complex user models be built? Motivated by prior work, we explore using machine learning methods to learn the weights of the various aspects, which we call criteria. Of particular interest is whether the criteria combine in a nonlinear way. We evaluate the use of linear and logistic regression on a criteria-based news filtering data set. We also present a way to learn nonlinear models with linear regression, and propose a new loss function based inspired by a common information retrieval metric.

1. INTRODUCTION

A crucial component of information retrieval is an accurate model of the user’s need, that is, some representation of what information the user wants that can be matched against the available set of information. Interest in information retrieval has grown with the widespread adoption of the World Wide Web, but the most often seen representation of the user’s need remains quite simple: a few terms that are presumed to occur in the documents of interest. There are good reasons for such a simple model, in particular, it is difficult to acquire the model from the user for a variety of reasons.

Nonetheless, it may well be that further significant advances in information retrieval will necessitate more complex models: an argument can be made that there is not sufficient discrimination from simple user models, particularly when queries are made of very few terms. Two questions arise when considering more complex models of the user’s need. First, what would be a more effective model? Second, how can this model be acquired? When historical data is available, it may be possible to use machine learning methods to induce a model of the user’s need, which is the approach we take here. Though machine learning may potentially be used to learn quite a number of aspects of the user’s model, we concentrate on only one, as described below.

In earlier work, Wolfe and Zhang [16] explored the use of a multi-criteria framework to model the user’s need. Inspired by operations research, they applied multi-criteria decision making models to model the user’s need. Given user-supplied ratings for the information on several pre-defined criteria, Wolfe and Zhang showed that combining the criteria produced considerably better results than using only a single criterion (such as content). In an unpublished followup study, Wolfe showed that the criteria combined in a nonlinear way, suggesting that further gains over the initial study (which used a linear model) were possible. However, the non-linear methods used were rather ad-hoc and potentially suboptimal. It is from this study that we base our current investigation upon, using the same user model based on several criteria.

Generally, we seek to answer the following in our investigation:

1. Is the underlying user model a linear or nonlinear combination of criteria?
2. If it is nonlinear, how can the models be learned effectively?
3. How can we use regularization to manage the complexity of nonlinear models without overly limiting their potential power?

The last question is of particular importance (though it also depends on the success of the prior questions). By removing the restriction of linearity, we have a much wider choice of possible models (which may also include linear models) and therefore a greater potential for an accurate fit of the data. But this increased power comes at an equally grave cost, as we may need to choose between a far greater number of possible models with the same limited data, and thus the danger of over-fitting is high. Careful use of regularization is needed to reduce the risk without reducing too much possible reward.

With respect to the earlier study of Wolfe, we also ask the following:

1. Can we repeat the results of the earlier study?
2. Can we improve upon these results with more principled methods?

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 2009 UCSC ...$0.00.
2. RELATED WORK

Most of the research in information retrieval that uses multiple criteria has been in information filtering. Manouselis and Costopoulou categorize 37 recommender systems that use multi-criteria aspect in their operation [6]. These systems primarily use only the weighted sum (i.e., linear combination) model. Of the information filtering systems we are aware of, PENG [9] is the most similar to the one in our experimental study. PENG is a multi-criteria news bulletin filtering system that utilizes several criteria, including content, coverage, reliability, novelty and timeliness. A later evaluation of PENG, using only content and coverage, showed comparable or superior performance to other approaches [1]. Farah and Vanderpooten have explored the use of multiple criteria in the context of search using rank-based methods. In their work [4], the user provides query terms as the only input (and thus criterion) for the search process. From this, additional criteria are formed from elements of the web page, such as text, keywords, anchor text and incoming links. Later work [5] expanded on this notion by using the rankings produced several high performing algorithms, with each algorithm essentially acting as a criterion or critic.

Learning user models based on multiple criteria (as opposed to content alone) is not common in information retrieval. Naïve Bayesian classifiers were used to learn content-based user profiles for movie search [3]. A more complicated scheme was used to predict whether a user would watch television programs [10], first by building a model of what genres a user likes, and then classifying each show based on its genres by means of a support vector machine. DIVA [7] uses a somewhat similar approach to recommend movies, using the C5.0 algorithm to classify each movie based on its metadata.

Outside of information retrieval, general additive independence models have gained some adoption and are more akin to our current approach. One method for estimating generalized additive independent utility functions is to regard them as random variables and use Bayesian techniques to estimate them [2]. This same utility decomposition concept was later applied to multi-issue negotiation, by representing the utility of a buyer in a utility graph [11].

3. DATASET

We used a news filtering data user study data provided by the University of California, Santa Cruz and Carnegie Mellon University [17]. The data was previously collected in a user study performed on the Yow-now news filtering system. We re-used the data for our experiments, however we could not change what or how the data was collected. Yow-now was an information filtering systems that delivered news articles to users from various RSS feeds. Approximately twenty users used the Yow-now system for about a month, reading news for at least one hour each day. More than 9,000 records of user reading behavior are included in the data set.

The users rated each article according to the following four criteria:

Authoritative: how authoritative the article appeared (0 or 1)

Novel: the novelty of the article (integers 1 through 5)

Readability: the ease of reading the article (0 or 1)

Relevant: the degree to which the article was relevant to general subject category of the article (integers 1 through 5)

In addition, the users gave an overall rating User Like for the article (also integers 1 through 5). However, in our use of the data, we shifted and rescaled the User Like rating to range from 0 to 1 for ease of use and interpretation.

In practice, the ratings for a news item on each criteria will be unknown and must be estimated by the filtering system; however, for our present investigation we take these as given. Based on the estimation of these criteria, the filtering system can further predict whether a user would like the news or not, and make filtering decision accordingly. Our machine learning task is to predict the User Like rating given the ratings for each of the four criteria.

4. LOSS FUNCTIONS

With any machine learning exercise, a loss function must be selected that accurately captures the evaluation of error. In our case, the ideal loss function should be specified by the users. Unfortunately, this is not possible as this information was not captured in the original study. It may also be too much to ask of users who are not familiar with loss functions to specify their own. Therefore, in this study we must pick what appears to be reasonable loss functions.

We take something of a shotgun approach and use several loss functions. Certain algorithms assume particular loss functions, so a fair comparison with those algorithms should include such loss functions. In our case, we use the matching loss framework of Warmuth and Hristakeva [15] for squared loss and logistic loss, as well as absolute loss and classification accuracy. Our primary focus is on squared loss, both to compare to the original study of Wolfe (which used square loss) and also because it is a reasonable choice in the domain (assuming users are more upset by a single document that is very inappropriate than by a few that are slightly off). We also evaluate our results based on a new loss function, inspired from a commonly used information retrieval measure.

4.1 Loss Based on a Generalization of the Fβ Measure

Information retrieval has several measures for evaluating the quality of returned results. Presumably, these measures have some correspondence to how users would typically rate returned results, and so are a reasonable substitute for the users’ personal evaluation measures. One of the most common, if not the most common, information retrieval measures is the F_β measure of van Rijsbergen [13], often defined in terms of precision (the fraction of documents in the result set that are relevant) and recall (the fraction of relevant documents over the entire set that are also present in the result set). One formulation for F_β-measure, in terms of the number of true positives (tp), false negatives (fn) and false positives (fp) is:

\[
F_\beta = \frac{(1 + \beta^2)tp}{(1 + \beta^2)tp + \beta^2 fn + fp}
\]

However, this assumes a binary model of relevance (either completely relevant or completely not relevant), whereas our
The analytical solution is to compute an arbitrary default weight vector rather than a zero weight vector. We regularize, with one modification; we regularize to an arbitrary squared error. We use the formulation of Warmuth [14] to replace the default algorithm is simple linear regression, which minimizes least squared error. We use the formulation of Warmuth [14] to replace the default transfer function and view the loss based on the generalized framework. As a result, we used simply a linear function that we are familiar with, the $F_\beta$-measure are required. Third, unlike the other loss (or gain) functions we use in our approach. We used standard machine linear algorithms in our investigation, though sometimes in unusual ways. Our main datasets have several degrees of relevance. We are not aware of any generalization of the $F_\beta$-measure to accommodate fractional levels of relevance, so we developed a new formula. Defining $P$ as the vector of predicted relevance values, and $Y$ as the vector of actual relevance values, our generalized version of $F_\beta$-measure is

$$ F_\beta = \frac{(1 + \beta^2)||\min(P, Y)||_1}{\beta^2||P||_1 + ||Y||_1} $$

where $\min()$ returns the pairwise minimum of the vectors. Since $F_\beta$-measure is a measure of gain, our loss function is simply $1-F_\beta$-measure. In our case, we choose $\beta = 1$, thus weighing precision and recall equally.

Unfortunately, several challenges present themselves with respect to this new loss function. First, as can be seen in Figure 1, though continuous, the generalized $F_\beta$-measure is not smooth, making it more difficult to analyze. Second, we were not able to identify an analytical solution for the minimum, so slower search algorithms such as gradient descent are required. Third, unlike the other loss (or gain) functions we are familiar with, the $F_\beta$-measure is defined not on individual predictions but on the entire set of predictions. This made the application of the matching loss framework perplexing. As a result, we used simply a linear transfer function and view the loss based on the generalized $F_\beta$-measure as a work in progress.

$$ W = (\lambda I + X^T X)^{-1}(\lambda W_0 + X^T Y) $$

where an exponent of $T$ indicates matrix transposition, $\lambda$ controls the amount of regularization, $I$ is the identity matrix, $X$ is the instance matrix, $Y$ is the vector of target values, $W_0$ is the regularization vector and $W$ is the vector of coefficients we seek. Larger values of $\lambda$ cause the solution to be closer to $W_0$. When no regularization is used, we use a different formulation that employs the matrix pseudoinverse to avoid potential singularities:

$$ W = \text{pinv}(X)Y $$

where $\text{pinv}()$ indicates the pseudoinverse, with the other quantities defined as before.

Another algorithm we use is gradient descent, which searches for the minimum of a function by moving down the instantaneous gradient at each step until some stopping condition is reached. We use three stopping conditions (stopping whenever one or more are true): if the number of iterations is 500; if no step can be found that moves closer to the objective; or if the L1-norm of the gradient is less than $10^{-4}$. The weights are initially set to $W_0$. At each step, they are updated according to the following equation

$$ W_t = W_{t-1} - \eta t \frac{\eta t - 1}{n} \frac{||W_{t-1} - W_0||_1 + \text{Loss}(f(W_{t-1}, X), Y)^T X)}{n} $$

where $\text{Loss}()$ is the loss function (one of those previously described), $\eta$ is a dynamically set learning rate, and the other elements are defined as before.

For gradient descent, we initialize $\eta_0$ (the value of $\eta$ before any iterations) to be 1. At every step, we make a limited attempt to satisfy the strong Wolfe conditions [8]. If $\eta_{t-1}$ satisfies the strong Wolfe conditions, it is used. If not, we test $2^{t-1}\eta_{t-1}$ with successively larger values of $n$ for the strong Wolfe conditions, until $n = 5$. If no satisfactory value of $\eta_t$ is found at this point, we weaken our criteria to be values of $\eta_t$ that reduce the objective, abandoning the search if no such $\eta_t$ is found once $n = 50$. At that point, the search terminates.

Finally, we also can treat the problem as a classification into multiple classes instead of a regression problem, which we will motivate later. We use binary classification algorithms applied to each separate class value to create the multiple classifier. The first algorithm we use for this purpose is logistic regression with gradient descent, as described above. The classifier predicts the given class if the probability prediction is above 0.5, predicts the negative of the class if it is below 0.5, and predicts nothing at exactly 0.5. The other algorithm we use is AdaBoost [12], in an analogous fashion, using the binary features as (weak) learners. In this case, the classifier predicts positive if the prediction is above 0.5, negative if it is below 0.5, and predicts nothing if the prediction is exactly 0. The feature representation is likewise changed to be either 1 or -1 (presence or absence of the rating combination). For the purposes of our accuracy measure, no prediction is treated as an incorrect prediction.

5. ALGORITHMS

We used standard machine linear algorithms in our investigation, though sometimes in unusual ways. Our main algorithm is simple linear regression, which minimizes least squared error. We use the formulation of Warmuth [14] to regularize, with one modification; we regularize to an arbitrary default weight vector rather than a zero weight vector. The analytical solution is
6.1 Loss Lower Bound

To define a lower bound on loss (and thus an upper bound on performance), we define the loss of the machine learning demigod, who must make its decision based on the features alone but uses some fantastic generalization that may not ever be realized. In other words, the demigod makes the optimal decision based on the features and knowing the true underlying distribution of the entire dataset (including unseen data), but without knowing the specific target value for the current instance.

We define the loss of the machine learning demigod only for the squared loss case, though it would be possible to do so for other loss functions (such as absolute loss). Squared loss in minimized by the mean, so for a given set of features, the machine learning demigod predicts the mean of all data with the same feature values, including the instance to be predicted. Thus, there is no test set--The machine learning demigod reports loss on the entire data set as if it was all training data. No algorithm based on the observed features could do better in the limit, though randomized algorithms could do better in particular trials. We stress that this is not a feasible algorithm, but a lower loss bound that may not be achievable.

6.2 Standard Models

As our primary interest is determining whether or not the overall rating is a linear combination of the criteria, our primary comparator is linear regression. Wolfe gives a mean squared error of 0.02689 on the same data set with hold-out (using all data for training except for the single test instance), so we expect to see similar in our study, though there are some differences: we use regularization tuned on a validation set and larger test set, whereas the original study had no explicit regularization and a hold-one-out evaluation.

In addition, we use gradient descent as described previously to minimize logistic loss and maximize the generalized F-measure gain. For our regression models, we also add a bias term (a constant feature in the data). We set our regularization default weight vector such that the set of minimum ratings yields the minimum overall rating, the set of maximum ratings yields the maximum rating, and each criterion is weighted the same.

6.3 Partially Nonlinear Models

We theorize that a nonlinear combination would result when there is some interaction between the criteria; namely, that the sum of the ratings is greater than the whole (high ratings work together for an even better result) or less than the whole (high ratings cancel each other out to some degree). We investigate the possibility for a nonlinear combination by enhancing our feature space, rather than using new algorithms or some transfer function. This is feasible as our features are discrete with a low number of possible values.

We explore only interactions between only two criteria at a time for several reasons. First, an interaction between two criteria is sufficient to prove that the underlying model is nonlinear. Second, such a model is simpler, which makes it easier to understand and computationally more efficient. Third, our suspicion is that there is not enough data to support more complicated interactions.

We model interactions amongst pairs of criteria by creating new binary features that correspond to specific values of criteria. We cast this as a linear combination as follows:

$$ P_{ab} = \sum_{i=1}^{m} w_i v_i + c_{ab} \sum_{x \in A} \sum_{y \in B} I(v_a = x, v_b = y) $$

where $P_{ab}$ is the predicted overall rating, $a$ and $b$ are the features to be combined, $A$ is the set of possible values for feature $a$, $B$ is the set of possible values for feature $b$, $I$ is an indicator function that returns 1 when the arguments hold, 0 otherwise, $v_i$ is the rating for the $i^{th}$ feature and $w_i$ and $c_{ab}$ are the coefficients to be learned. Note that the first summation in the equation is the same as the original linear combination, and the second summations is simply a linear combination over a new set of (derived) features. In other words, we have created new binary features for each possible pair of values for features $a$ and $b$. For example, when combining authority and readability (two binary features), $2 + 2 = 4$ new binary features are created; when combining authority and novelty (a binary and a five-valued feature), $2 + 5 = 10$ new binary features are created. One can think of these induced binary features as correction factors, and as such, any nonlinear combination involving only these two features can be represented. In practice, not all of these induced features are needed (two of them add redundant information), so we eliminate the lowest pair and highest pair of binary features as “anchor points”.

With our nonlinear combination expressed as a linear combination over a larger feature set, we can once again use linear regression to find a least-squares solution. In his earlier “hold-one-out” approach, Wolfe cited a best squared loss over all such combinations as 0.02392.

6.4 Fully Nonlinear Models

Finally, we take our nonlinear representation to its full extreme, creating binary features for every possible set of feature values. Since there are two binary valued features and two five-valued features, this results in $2 \times 2 \times 5 = 100$ combinations, but in fact four possible combinations never occur, so our representation is of 96 binary values. Since these 96 binary values cover all possibilities, there is no information remaining in the original feature space, so we discard it. We make the same transformation over the overall rating (target values), resulting in five binary class values. Such a representation is well-suited to a multi-class classification problem.

However, our multi-class classifiers are actually series of binary classifiers, one for each class. This presents the disturbing possibility that multiple classes, or no class, could be predicted by the group of classifiers. One possibility would be to use the class with the highest predicted value, however, we used a slightly different approach. Rather than have each binary classification problem represent a single class value, we represented it as all ratings at that level and above. So, the third class (originally a 0.5 overall rating) would also be true for original ratings of 0.75 and 1.0 (but not 0.25 or 0.0). It was our hope that this would decrease the possibility of conflicting predictions, though we did not evaluate this; in practice we simply used the highest positive predicted class. Nonetheless, this representation has several other advantages. First, since the $\geq 0.0$ class was always true, we eliminated it from our representation. Second, for some classes, the positive/negative ratio was closer to even, which we presume makes the instances we easier to
differentiate. Third, it is intuitively easier to separate based on a boundary (e.g., distinguishing items rated above 3 from those rated below) than on a specific value (e.g., distinguishing items rated exactly 3 from those rated either 1, 2, 4 or 5).

It was our expectation that the fully nonlinear approach would be unsuccessful, as the space was too unconstrained. With multi-class logistic regression, we used the same regularization vector as described before, mapping whatever combined values would have resulted onto the coefficients for the new binary values; thus, in the absence of any training, both regularized models would produce exactly the same predictions. For multi-class AdaBoost, we do not know how to regularize except by limiting the number of weak learners selected.

7. EXPERIMENTAL RESULTS

In all our experimental runs, we ran each test case ten times and report the mean and standard deviation results on the test data of each case. We randomly permute the data each time, selecting one fourth of the data for testing. The remaining three-fourths of the data is used to build our model. We use 10-fold cross validation to automatically determine the amount of regularization to use (which we represent with $\lambda$). We initially set $\lambda=1$, and successively halve and double $\lambda$ until a “valley” is observed (i.e., the loss on the validation set is consistently worse than the best loss thus far). We did not set any requirement on how deep the valley needed to be.

7.1 Loss Bounds

Table 1 shows the losses for the machine learning demigod (our lower bound on squared loss). In this case, since the demigod uses both training and test data as training data, we evaluated over the entire data set for each permutation. Thus, the result of every trial was the same. Also, there was no tuning, as the machine learning demigod has no use for regularization.

We ran the machine learning demigod in two modes. In the first mode, the data from each user was interpreted as a separate data set, with separate models learned for each. In the second mode, the user information was ignored, so a single model was learned for all users. The demigod which treated each user as separate had superior performance over the one that ignored user information. This would seem to indicate that the underlying user models are different and that better results are possible when different models are built for each user. However, this difference is not enough to establish this, as the machine learning demigod, by its construction, will tend to have better results when larger sets are divided into smaller sets (more unique features are better, unlike normal machine learning problems). However, we assumed that separate models for each user made sense, and took this approach henceforth.

7.2 Standard Models

Table 2 shows the results for several linear models; losses other than squared loss are given for comparison, though squared loss is the only “fair” evaluator for linear regression. Basic is our standard linear regression model with regularization. NoReg is the linear regression model without any regularization (using the matrix pseudoinverse, as described previously). RegW is not a learned model, but the regression results when using the regularization default vector alone

<table>
<thead>
<tr>
<th>Table 1: Losses of the Machine Learning Demigod</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loss FN</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>Abs. Err. mean</td>
</tr>
<tr>
<td>Abs. Err. Std. Dev.</td>
</tr>
<tr>
<td>Accuracy mean</td>
</tr>
<tr>
<td>Accuracy Std. Dev.</td>
</tr>
<tr>
<td>1-Fβ mean</td>
</tr>
<tr>
<td>1-Fβ Std. Dev.</td>
</tr>
<tr>
<td>Logistic mean</td>
</tr>
<tr>
<td>Logistic Std. Dev.</td>
</tr>
<tr>
<td>Squared mean</td>
</tr>
<tr>
<td>Squared Std. Dev.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2: Basic Linear Regression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loss FN</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>Abs. Err. mean</td>
</tr>
<tr>
<td>Abs. Err. Std. Dev.</td>
</tr>
<tr>
<td>Accuracy mean</td>
</tr>
<tr>
<td>Accuracy Std. Dev.</td>
</tr>
<tr>
<td>1-Fβ mean</td>
</tr>
<tr>
<td>1-Fβ Std. Dev.</td>
</tr>
<tr>
<td>Logistic mean</td>
</tr>
<tr>
<td>Logistic Std. Dev.</td>
</tr>
<tr>
<td>Squared mean</td>
</tr>
<tr>
<td>Squared Std. Dev.</td>
</tr>
</tbody>
</table>

(equivalently, the learned model when $\lambda = \infty$). Figure 2 shows the “valley” of the loss function that we used to select $\lambda$; the actual value selected in our runs tended to be close to 2. Figure 3 shows that regularization decreases the distance to the default vector, as expected. Figure 4 shows the effect of regularization on time; there is no obvious pattern, but it does show that linear regression is relatively speedy.

The loss incurred for linear regression with and without regularization is comparable; it is somewhat surprising and disappointing that the loss with regularization is slightly higher for all the loss functions. The loss when using the default vector alone may hold a clue; it is rather high, however it is lower than it would be when using a default zero-vector (which would result in predicting 0 regardless of the ratings). Whether regularizing with the zero-vector (minimizing total weight rather than minimizing distance from a default vector) is superior is not known; of course, even other vectors could yield better results. This is one instance of the unfortunate “cooking” that sometimes manifests itself in machine learning, but too much cooking can lead to overcooking, specifically over-fitting.

As we use regularization in the rest of our experiments, the squared loss result (0.02736) for linear regression serves as our primary comparator. We note that this is well below the squared loss result (0.02736) for linear regression serves as our primary comparator. We note that this is well below the squared loss result (0.02736) for linear regression serves as our primary comparator. We note that this is well below the squared loss result (0.02736) for linear regression serves as our primary comparator. We note that this is well below the squared loss result (0.02736) for linear regression serves as our primary comparator.
with our linear regression methods. Figure 5 and figure 8 show the effect of regularization on the loss function for logistic loss and $1-F_\beta$, respectively; both use a value for $\lambda$ around 0.5, where the gradient descent with squared error had a value closer to 1 (which is lower than that used in linear regression). Figures 6 and 9 show the regularization decreases the distance to the default vector, as expected. Figures 7 and 10 show how regularization affects execution time, with more regularization apparently decreasing the execution time. Note, too, that the total time is much larger than linear regression, which is expected as we are searching for the minimum rather than calculating it directly.

The results of these runs are rather surprising. Gradient descent does not do a good job of optimizing for $1-F_\beta$ when this is loss function used, in fact, all other methods do a better job. Closer examination reveals that $1-F_\beta$ is apparently likely to get trapped in a local minimum; in no case did the search reach the iteration or gradient limit, and since it is not outperforming the other methods, the minimum found is apparently not the global minimum. Therefore, it appears it may be better to use a smoother loss function, even when the goal is to minimize $1-F_\beta$. On the other hand, gradient descent reaches the iteration limit nearly have the time, so it may have benefitted from more iterations. Even more curious, gradient descent with squared loss had not only better logistic and $1-F_\beta$ losses than their counterparts, but also had better squared loss than the linear regression solution. This is really quite surprising. This could be due to random effects, but is more likely due to differences from the formulation for regularization in both methods, as well as the lower regularization used for gradient descent.

### 7.3 Partially Nonlinear Models

Table 4 and 5 give the results for our partially nonlinear models. They are named according to the index of the features combined: Combo 12 (authority and novelty); Combo 13 (authority and readability); Combo 14 (authority and relevance); Combo 23 (novelty and readability); Combo 24 (novelty and relevance); and Combo 34 (readability and relevance). We omit the analysis of the effect of regularization, except to note that all methods tended to use a value of $\lambda$ of 4, except Combo 13 which used a value of 2.

The results are encouraging. With the exception of combo
23 (novelty and readability), each method performs as well or better than any of the previously examined methods on all loss functions. Indeed, Combo 24 has the best result on every loss function except for classification accuracy, which is close. Some of the differences are slight, however, it is encouraging that Combo 24 performs as well or better than any other method. This also fits the result of the prior experiments of Wolfe. Specifically, the square loss of 0.02595 beats the linear comparator of 0.02736. However, this is still not as value of 0.02392 given by Wolfe in the prior study.

### 7.4 Fully Nonlinear Models

Taking nonlinearity to its extreme, we give the performance for multiclass classification in table 6. Here, an instance is deemed correctly classified only if all the binary classifiers give the correct classification. We only use classification accuracy in this case, because the other loss functions are more difficult to apply.

Not surprisingly, none of the multi-class classifiers performed particularly well, with accuracies well below the other methods. MultiClass AdaBoost performed particularly poorly. This is likely because we only have implicit regularization through early stopping. With so many degrees of freedom,
regularization plays an important role, but AdaBoost here has little guidance in building a good model.

7.5 Training with More Data

Finally, we made one more attempt to reach the squared loss given by Wolfe. We ran the combo24 case, our best performer so far, with 99% of the data used for training and 1% reserved for testing. The method for searching for the right amount of regularization stayed the same. Since 1% of the data is not very much to test upon, we reran this process 100 times (whereas it was 10 times before). Of course, since the data is randomly permuted each time, some instances may occur in our test statistic (though not in a given test set) more than once.

Table 7 shows the result. Indeed, the value of squared loss decreased, as did that for absolute error and accuracy. Our other loss functions increased, however this is not necessarily meaningful as linear regression is only minimizing squared loss. It is not surprising that different loss functions might be at odds (i.e., not sharing an optimum). We do note that the lower squared loss value of 0.02434 still has not reached the value from the previous study of 0.02392. Of course, we have not exactly replicated that study; there hold-one-out was used, as well as a different formulation of linear regression. Nonetheless, it seems unlikely that a noticeably lower value of square loss can be achieved on this data set without exploring new techniques.

8. CONCLUSIONS

Our results confirm that of the followup study of Wolfe, namely, that there is some nonlinear interaction between criteria in the YowNow data set. Moreover, our approach of enhancing the representation to learn the partially non-

<table>
<thead>
<tr>
<th>Table 6: Multi-Class Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loss FN</td>
</tr>
<tr>
<td>Accuracy mean</td>
</tr>
<tr>
<td>Accuracy Std. Dev.</td>
</tr>
<tr>
<td>Lambda mean</td>
</tr>
<tr>
<td>Lambda Std. Dev.</td>
</tr>
</tbody>
</table>
linear model seemed to work well. We were able to take advantage of certain aspects of the data set which would limit its applicability to other data sets, namely that the features were discrete with relatively few values. Also, the greater degrees of freedom increases the risk of over-fitting; in our case, this risk was mitigated, probably because of regularization and the amount of training data available. This was not necessarily true for our extreme (totally) nonlinear representation; we would expect that much more data would be needed to make use of such an unrestricted model.

Our experience with regularization was mixed. Our hope was that regularization would help guide the learning algorithms to more effective models. On the other hand, regularization, by its very nature, trades off some (training) model accuracy in order to move the solution towards some expected value in order to (hopefully) improve performance on the test set. It’s something of a dilemma: if the expected solution is a good one, regularization will help, but on the other hand, if there is a strong expectation towards a particular solution, why employ machine learning at all? In our case, the preference towards a particular solution (our default weight vector) turned out not to be rather good and this may have hurt performance. Searching for a good regularization vector leads us back to the same Catch-22. Nonetheless, we showed that our refined regularization did indeed move solutions to the desired vector, and with proper tuning, even a not-so-optimal default vector still produced reasonable results.

The generalization of the $F_\beta$-Measure proved to be difficult to use as a gain (loss) function. Gradient descent was often trapped in local minima; an analytical solution would be preferable, but we did not immediately see how to derive one. Indeed, the $F_\beta$-Measure has different properties than other loss functions that we had encountered, making it more challenging to use. One possibility would be to enhance gradient descent with some number of random restarts, but this can be time consuming. A similar possibility, based on the observation that minimizing other functions also found better values for $F_\beta$, is to use a starting point based on the found minima of another function (particular squared loss, as linear regression is fast) for the gradient descent.

Finally, our search for squared lower loss than the original study proved to be in vain. We acknowledge that our experiments had some key differences (such as hold-one-out validation v.s. testing with larger folds) which may explain the differences, but in any case, it does not seem likely that a large decrease in loss could be achieved on this dataset with similar (or perhaps dissimilar methods).

\section{Future Work}

Though our results clearly show that an interaction exists between criteria in the YowNow data set, we do not know whether this is true in general. Therefore, we would like to try similar experiments on more data sets. Unfortunately, the multiple criteria framework has not been widely adopted in information retrieval, making other data sets difficult to find. One possibility is to use online reviews, such as those on epinions.com; in fact we had crawled a limited data set from epinions.com, but ultimately did not have time to use it in this report.

Our results validated our choice of building separate models for each user; apparently these is enough difference in how the final overall rating is dependent on the ratings for the individual criteria from user to user that learning separate models give better results, despite having less training.

\begin{table}[h]
\centering
\caption{More Training: Combo 24}
\begin{tabular}{|c|c|}
\hline
Loss FN & combo24-Squared \\
\hline
Abs. err. mean & 0.1066 \\
Abs. err. Std. Dev. & 0.01094 \\
Accuracy mean & 0.9139 \\
Accuracy Std. Dev. & 0.02564 \\
1-F$_\beta$ mean & 0.08835 \\
1-F$_\beta$ Std. Dev. & 0.009467 \\
Logistic mean & 0.079 \\
Logistic Std. Dev. & 0.0182 \\
Squared mean & 0.02434 \\
Squared Std. Dev. & 0.005322 \\
\hline
\end{tabular}
\end{table}
data available for each model. Unfortunately, we had made some early design decisions that proved difficult to revise during the available time frame. Specifically, we optimize the amount of regularization over all users, so all users have the same amount of regularization. Intuitively, this is not justified; users with more training data need less regularization than those with less training data, for example. This may be one reason why the use of regularization was less clearly useful. A similar problem is that all weights are regularized the same. However, in the partially nonlinear case, we essentially have two different sets of weights; those on the original features and those on the combined features. It might make sense to regularize these two sets differently, for instance, it may be that the combined features require more regularization. Unfortunately, this would lead to more complexity.

Another possibility that is to try to improve the default regularization vector. A unconstrained search for a better vector would appear to be misguided, however, there are at least two other possibilities. One would be to learn an average weight vector from all other users (perhaps without any regularization), and to use this as the default weight vector for this user. Thus, in the absence of data, we would predict this use to be like the mean of the other users. Such “borrowing” occurs frequently in collaborative filtering. A second possibility for the partially linear model is to build a linear model from the original features and use this as the default weight vector. In fact, this was the approach used by Wolfe in the original study, though his concept of regularization was different.

With respect to an interaction between criteria, we showed that there are interactions, but their full extent is not established. If there is interactions among two of the criteria, there may also be interactions among three or all. We can easily extend our two criteria interaction model to use three criteria. Indeed, our decision to discard the original features in the totally nonlinear case may have been a mistake; with a proper use of regularization, we would expect to see better results.

Finally, the overall area is of acquiring more complex models of users’ informational needs is full of potential uses of machine learning methods. In this study, we have assumed the ratings on each criterion were already given in order to decompose the problem, but in practice they will need to be estimated in some way. When training data is available, it would make sense to use machine learning to predict the ratings. Also, for many applications (such as information filtering), the data is likely to have strong time-dependent features as interests and needs change. Online algorithms may be applied to exploit the online-ness.

10. ACKNOWLEDGMENTS

The author would like to thank Professor Yi Zhang for advice during the original study, and Matt Brown for contributing to the generalization of $F_\beta$-Measure.

11. APOLOGIES

Sorry about the placement of the figures and tables (and the filenames appearing above the figures). My \LaTeX{} is still a work in progress.

12. REFERENCES

13. APPENDIX A: MOTIVATION FOR THE $F_\beta$ GENERALIZATION

Intuitively, the generalized version of $F_\beta$ should return the same result as the original measure when all relevance judgements and predictions are binary. Also, a perfect prediction should have a score of 1, and predicting all documents as irrelevant when at least some partially relevant documents exist should produce a score of 0.

The count of true positives $tp$ and false positives $fp$ should be the size of the (query) result set; adjusting for our fractional relevance, this is naturally defined as (where $\hat{Y}$ represents the prediction to avoid confusion with precision):

$$tp + fp = ||\hat{Y}||_1$$

Likewise, the count of true positives $tp$ and false negatives $fn$ should be the size of the relevant set; adjusting for our fractional relevance, we have represents the prediction to avoid confusion with precision):

$$tp + fn = ||Y||_1$$

Identifying the set of true positives alone is more challenging. After some consideration, the minimum of the prediction and actual value was chosen, motivated as the intersection between the result set and the relevant set.

This gives us a definition of precision $P$

$$P = \frac{tp}{tp + fp} = \frac{||\min(\hat{Y}, Y)||_1}{||\hat{Y}||_1}$$

as well as recall $R$

$$R = \frac{tp}{tp + fn} = \frac{||\min(\hat{Y}, Y)||_1}{||Y||_1}$$

Substituting into the formula for $F_\beta$ in terms of precision and recall gives us the generalized form:

$$F_\beta = \frac{(1 + \beta^2)P \cdot R}{\beta^2P + R} = \frac{(1 + \beta^2)\min(||\hat{Y}, Y||_1) \cdot ||\min(\hat{Y}, Y)||_1}{\beta^2||\min(\hat{Y}, Y)||_1 + ||\min(\hat{Y}, Y)||_1}$$

$$= \frac{(1 + \beta^2)||\min(\hat{Y}, Y)||_1}{\beta^2||\hat{Y}||_1 + ||Y||_1}$$

14. APPENDIX B: ANALYSIS OF COEFFICIENTS

We present a very brief analysis of the learned coefficients for the original feature space using linear regression without regularization. We use hold-one-out in these experiments, scaling all features to be defined over a 0 to 1 scale.

Table 8 shows the results with the original number of options for novelty and relevance. Assuming this is a somewhat accurate model of the underlying user model, this would seem to indicate that novelty is less important than the other criteria overall, and that there is more variation in importance for authority and relevance.

However, we wondered if the fact that the criteria had a different number of possible ratings was skewing these results (binary for authority and readability, five values each for novelty and relevance). Unfortunately, we can’t tell what the ratings would have been with a different number of options. To get an estimate, we map the five values of novelty and relevance onto a binary scale by mapping all values below 0.5 to 0, all values above 0.5 to 1, and randomly mapping 0.5 to either 0 or 1.

Table 9 shows the results with this binary representation. Interestingly, the results are different; novelty again is less significant, but only slight, and the variation is about equal for the criteria except for relevance which has less variation. The significance of this, however, is an open question.

<table>
<thead>
<tr>
<th>Mean</th>
<th>Authority</th>
<th>Novelty</th>
<th>Readability</th>
<th>Relevance</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1768</td>
<td>0.1677</td>
<td>0.1878</td>
<td>0.1735</td>
<td></td>
</tr>
<tr>
<td>Std. Dev</td>
<td>0.1187</td>
<td>0.1061</td>
<td>0.1031</td>
<td>0.0574</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mean</th>
<th>Authority</th>
<th>Novelty</th>
<th>Readability</th>
<th>Relevance</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0889</td>
<td>0.0746</td>
<td>0.0908</td>
<td>0.0946</td>
<td></td>
</tr>
<tr>
<td>Std. Dev</td>
<td>0.0927</td>
<td>0.0390</td>
<td>0.1014</td>
<td>0.0350</td>
</tr>
</tbody>
</table>