Feature Engineering for a Gene Regulation Prediction Task

George Forman
HP Labs, Palo Alto, CA

ABSTRACT

This paper describes an approach that won honorable mention for the gene regulation prediction task of the 2002 KDD Cup competition [1]. Our methodology used extensive cross-validation to direct the search for an appropriate problem representation and the selection of an 'off-the-shelf' induction algorithm. A prominent trait of the dataset is the presence of three hierarchical attributes, for each of which we generated a novel predictive feature: the percentage of positives hierarchically aggregated at the node specified by the instance.

Keywords

Machine learning, hierarchical attributes, bioinformatics.

1. INTRODUCTION

In contrast with most of the machine learning benchmark datasets, much of the challenge of this year's KDD Cup prediction contest was in determining how best to represent the available data—as is often the case with company-internal prediction tasks we face in the Data Mining group at HP Labs.

A companion paper in this issue [1] describes the competition, the available data, and the competitors' results. Due to space limitations, we must assume the reader is familiar with the task. The purpose of this paper is to explain one of the approaches that achieved honorable mention. The take-away messages beyond the competition itself include the methodology employed and an approach for representing nominal attributes having a hierarchical relationship among the values.

Section 2 describes the methodology and philosophy guiding the work. Section 3 presents our feature engineering. Section 4 lists the final choices we made. Section 5 summarizes.

2. METHODOLOGY & PHILOSOPHY

Since the contest score is the sum of the performance on the narrow and broad prediction tasks, we optimized for each independently.

In order to facilitate an efficient search for the best prediction model, we leveraged a Perl software framework we had previously developed. It enables us to focus our effort on quickly prototyping a variety of feature engineering options. It provides an automated process to perform feature selection and induction performance testing using stratified cross-validation. Rather than measuring accuracy, we extended it to evaluate the same performance measure used to judge the contest: area under the ROC curve. Using this framework and four 600-800 MHz CPUs, we were able to quickly measure the performance for many configurations. Induction algorithms that we trialed included the WEKA open-source implementations of Naïve Bayes, linear kernel Support Vector Machines (SVM), and AdaBoosted decision stumps [5]. The feature selection methods trialed were Information Gain, Bi-Normal Separation [2] and variants. Because the framework provides for feature selection (within each cross-validation fold), we need not trouble ourselves with manually eliminating useless features, but only with designing features that may be predictive, as discussed in the next section.

To consider the cross-product of the entire design space is not feasible, however, automated testing of portions of the space gives much more visibility of the search terrain to the person guiding the exploration. Cross-validation helps mitigate, but cannot eliminate, the possibility of overfitting the data. Likewise, although SVMs are popularly touted for their theoretical guarantees against overfitting [4], their wide margin only exists in a feature space that is arbitrarily malleable when reformulating the problem representation [6]. Nonetheless, we attempted to optimize average performance on stratified cross-validation samples of the available training data.

The guidance drawn from cross-validation testing is constrained somewhat by large variance. For example, in the narrow task with just 38 positive examples, a stratified 10-fold split yields just 3.8 positives on average in the testing fold, leading to wide variance in the performance estimation. To compensate, we perform a large number of randomized trials (e.g. 20-100 as in bootstrapping), rather than 10 as in traditional cross-validation.

We simplified management of the voluminous and distributed performance data by appending all results to a single database table, columns capturing all parameters of the test conditions. We could easily determine leading configurations with our Perl tools or with interactive pivot charts in Excel.

3. FEATURE ENGINEERING

Next we discuss our feature engineering from the three hierarchical attributes, the interaction graph, the textual abstracts, and the gene names themselves.

Hierarchical Attributes: Given a nominal attribute whose values can be hierarchically aggregated with a known tree, a natural representation is to generate a binary feature for each node in the tree—set to '1' only on the path to the current value. (We treat the many missing values as a separate top-level node.) We generated 494 such features, and many were predictive, but we were dissatisfied with how dispersed the information is in the large hierarchies. Only a few nodes contained positives at all.

To remedy this, we engineered additional *hierarchy prevalenc*e features for each hierarchical attribute. Optional step 1: We pruned away nodes for which there are no instances in the contest's testing set, which we refer to as *transductive pruning*. It is in the spirit of *transductive learning* [4], which focuses the modeling task on the specific examples to be labeled and does not use information from the answer key. This eliminated 200+ nodes. Step 2: We generated two percentage-valued features: given the attribute value, we return the percentage prevalence of positives (vs positives & negatives) found in the lowest and highest node in the path of the pruned tree. For example, a gene with function attribute='mRNA synthesis' would have the value 1/41 for the low attribute, and 2/80 for the high attribute, since there are just two positives out of 80 genes at or under the top-level node 'Transcription'. A value of 0% indicates that no positives ever had the current attribute value. Without this feature, it would be much harder (more data) for an induction algorithm to learn to '*or*' together the 230 mutually exclusive 'function' nodes that contain no positive examples.

Interaction Graph: The undirected interaction graph lists associated genes. We generated a simple integer feature that indicates for a given instance, the number of genes it interacts with. This was a strong predictor for the broad task.

A natural feature engineering approach for this sort of interaction information is to duplicate the feature set, copying the features for the gene(s) it interacts with. Since the motivation boils down to a hunch that the prediction of the association helps predict the instance (as in relational learning), and since in this contest the set of ultimate test instances is small and fixed, it seemed more straightforward to generate a single additional feature that modeled this assumption directly. So, we generated just two additional integer features indicating the number of narrow and broad positives the gene interacts with (determined only from those known positive in the current training split). This features proved less valuable, yet somewhat predictive.

Textual Abstracts: For each gene, we concatenated all pertinent abstracts as determined by gene-abstracts.txt, and generated a binary feature for each unique (lowercased, alphanumeric) word. Having to load/process the 18 MB of abstracts increased the run time from 7 seconds to 50 seconds, and, unfortunately, tended to degrade prediction accuracy overall.

Gene Names: We supposed that the naming of the genes, e.g. YMR228W, was generated by a non-random process that may have some bearing on the prediction tasks at hand. We generated a numeric feature for the number, and binary features for various sub-sequences of characters. As it turns out, for the narrow task, the second and third characters together are (negatively) predictive: 'nl' is among the strongest predictors (appearing in 0 positives and 158 negatives), followed by 'gr' and 'pl'. These were not very strong predictors overall, so we do not believe they leaked information from the answer key illegitimately.

4. FINAL CHOICES

In the end, our best average predictor for the narrow task was Naïve Bayes on just twelve features, including the percent positives at the top and bottom of the three unpruned hierarchies, the three interaction features (made binary), and gene name substrings 'nl' and 'n'. Its estimated performance when training on 90% of the available training data was 0.67 ROC area, and when trained on 100% of the data, 0.6731 on the contest test set.

Our best average predictor for the broad task was Naïve Bayes on 48 features, including the top and bottom hierarchical prevalence features for the three *transductively pruned* hierarchies, the three interaction features (but not made binary) plus binary indicators for several of the hierarchy nodes, e.g. localization=transport vesicles / golgi ER transport vesicles, function=classification not yet clear cut, function=cell rescue defense and virulence, and protein class=protein phosphatases / catalytic subunits / PP2C family. The latter is a positive predictor with two positives and two negatives, but most features were negative predictors. Its estimated performance on 90% of the training data was 0.59, and 0.6295 in the contest.

While we expect some variance, it is difficult to estimate how much, given that we can generate only highly correlated samples. By plotting the learning curve as we vary the percentage of training data, and extrapolating the performance for % of the training data, we estimated an additional ~+.007 ROC area for both tasks in the final contest. After seeing the low ROC scores, we wished to validate the competition following [3]. A randomized distribution analysis of all the contestants' scores validated that they are significantly better than a random collection of simple classifiers.

5. CONCLUSIONS & TAKE AWAYS

No amount of clever induction or feature selection can make up for a lack of predictive features in the input. Hence, feature engineering is a key step for difficult prediction tasks. We estimate via a cross-validation lesion study that creative feature engineering was responsible for adding +11% to the performance (+0.08 and +0.06 ROC area for narrow and broad tasks).

Likewise, extensive use of automated cross-validation to guide the search for an effective model added immeasurable benefit over selecting a single model 'blind,' which is the only method that can safely be said to avoid overfitting. Even structural risk minimization techniques such as SVM [4] cannot safeguard against 'overfitting' the features to the dataset [6]. Nonetheless, experience shows that feature engineering is generally a worthwhile risk, and cross-validation helps mitigate the risk.

Perl again proved an excellent language for quick prototyping—minimizing human programming effort rather than CPU time. Although C is often chosen for being faster, the overall running time was very acceptable at under a minute per data point on an 800MHz HP Kayak XU running Linux—despite the inefficiencies on each run of re-parsing 19+ MB of input data, & launching a separate Java process running the WEKA machine learning algorithms, communicating via generating/parsing files.

6. ACKNOWLEDGMENTS

We wish to thank Bin Zhang, Jaap Suermondt, and WEKA.

7. REFERENCES

[1] Craven, Mark. The Genomics of a Signaling Pathway: A KDD Cup Challenge Task. KDD Explorations 4(2), 2002.

[2] Forman, G. An Extensive Empirical Study of Feature Selection Metrics for Text Classification. J. of Machine Learning Research, forthcoming in 2002.

[3] Forman, G. A Method for Discovering the Insignificance of One's Best Classifier and the Unlearnability of a Classification Task. DMLL Workshop, ICML, 2002.

[4] Vapnik, V. The Nature of Statistical Learning Theory, 1995.

[5] Weka machine learning project, www.cs.waikato.ac.nz/ml

[6] Zhang, B. Is the Maximal Margin Hyperplane Special in a Feature Space? Hewlett-Packard Labs Tech Report HPL-2001-89, 2001.

About the author:

George Forman is a research scientist at HP Labs in the Data Mining Group. He received his CS Ph.D. from the University of Washington. http://www.hpl.hp.com/personal/George_Forman