COURSE MEDIAN PREDICTION VIA SYLLABI ANALYSIS [FINAL REPORT]

CORALIE PHANORD, GRAESON McMAHON, KELSEY JUSTIS

March 15, 2015

1 Problem Statement

College students often find median grades helpful in the course selection process. Knowledge of a course's median grade provides insight on the difficulty of that course, enabling students to set up a well-balanced schedule. In this project, we applied supervised machine-learning algorithms to course syllabi, hoping to find some correlation between the content and rhetoric of those syllabi and their associated median grades.

2 Data

2.1 Data Collection

Data collection was a lengthy process and involved two primary sources: Dartmouth class/department websites and department heads themselves. By searching the former and reaching out to the latter, we managed to accumulate upwards of 500 syllabi, largely in .pdf format. 30-40% of these were unusable, either because their respective courses' medians were not listed online or they did not contain parsable text (several were scans of physical syllabi).

Median grades were taken from the registrar's website[1] and placed into a .csv file using Excel.

2.2 Data Formatting

After collection, we used Xpdf's[2] *pdftotext* program to convert every .pdf file to .txt to facilitate later parsing. Each of these .txt files was renamed using the following scheme: DEPARTMENT-COURSE #-TERM-SUBCOURSE #, closely following the registrar's course-naming system. We then wrote code in MATLAB that, given the title of a syllabus .txt file and the columns of the median .csv file (term, class name, and median), output a syllabus's corresponding median. This was error-prone, as small inconsistencies in the registrar's course-naming conventions made our code generate a large number of false negatives when attempting to match syllabi titles to entries in the .csv file. Manual examination of each unmatched syllabus was therefore necessary. As a result, we used about 235 syllabi in our milestone tests. However, we were able to add 24 more syllabi to our dataset and represent 19 academic departments at the time of our final run.

2.3 Syllabi Text Parser

Upon successfully collecting and formatting the syllabi we were then able to parse through the .txt files and begin examining the more than 450,000 words of content. The final parser design was decided upon reviewing best practices found in others' code online. One script that proved especially impactful was developed by

[1] http://www.dartmouth.edu/ reg/transcript/medians/
[2] http://www.foolabs.com/xpdf/home.html

Suri Like.[3] Using heavily modified portions of this work we filled in the skeleton of our program that filters through the text content for alphanumeric words. This parser is then capable of producing the dictionary of vocab used in each individual syllabus. The parser was then further extended to handle a desired batch of syllabi .txt files and enable feature extraction.

2.4 Features

Our final implementation included a number of high-level and low-level features:

High-Level: Course syllabus department, number, syllabus length (word count), enrollment.

Low-Level:

- Negation word (no, not, never, etc.) count
- Testing word (quizzes,tests,exams, etc.) count
- Homework word (problem set,essays, homework,etc.) count
- Number of percent signs present (as an indication of the grade-breakdown)
- Presence of lab words
- Presence of project words
- Frequency of you words
- Frequency of I words

For our final tests, these features and an additional request were included. The additional request (made in the final week by Professor Torresani) for having a binary variable for each word used in all syllabi was implemented by finding the unique words of each syllabus, combining them together and finding the unique words of that collection, thus creating a vocabulary for Dartmouth College syllabi. We then found all words which appeared in more than one syllabus and checked against them all in each syllabus.

3 Method and Results

We used decision-tree regression for our approach, basing our algorithm on the capabilities of Quinlan's C4.5 algorithm [4]. Specifically, we use C4.5's approach to stopping criterion, creating a leaf node when the examples at that node reach some threshold of homogeneity. We also apply C4.5's method of mixing categorical and numerical inputs; we handle the latter by sorting training sets by the continuous feature and analyzing the split centered between each pair of examples.

Although our target data could be interpreted as categorical, given there are a finite number of letter grade medians, we decided that there is utility in predicting intermediate values. For instance, a flat classification of A (4.0) might be less informative than 3.93, which is only slightly closer to an A than an A/A- (3.83); continuous predictions better convey uncertainty. Accordingly, we substituted variance for the information gain/entropy metric used in classification-based decision trees. Essentially, we consider the optimal split of our data to be that which most greatly reduces the variance in the resulting subsets. Prediction at leaf nodes is accomplished by averaging the training examples' associated medians at that node.

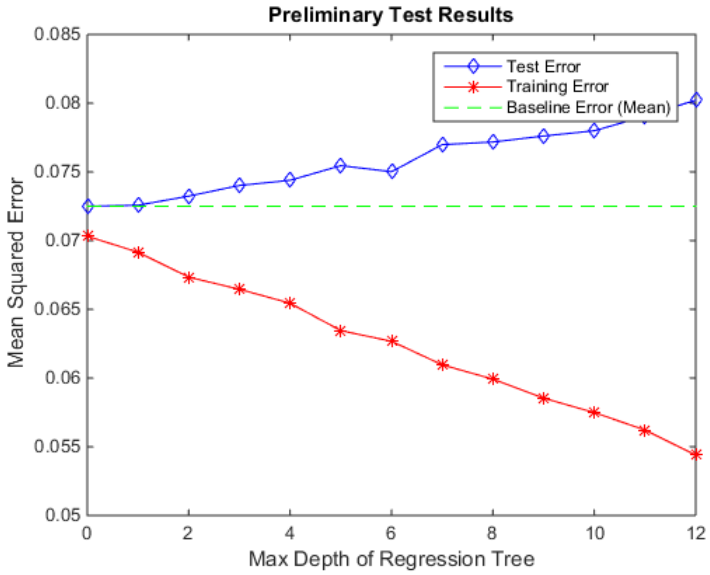
[3] Suri Like's Wordcount.m Script

[4] http://en.wikipedia.org/wiki/C4.5_algorithm

3.1 Version I

Our initial implementation suffered from several issues. We had a limited feature set (size four) at the time of testing, and the algorithm lacked the ability to handle categorical data. Additionally, our code had a few bugs, including an inability to perform prediction if a test example was inconsistent with the features of the training subset at every leaf node. More importantly, when calculating the summed variance after a split, we did not take into account the size of each subset. As a result, the algorithm favored splitting off a single example (with 'zero' variance) each time.

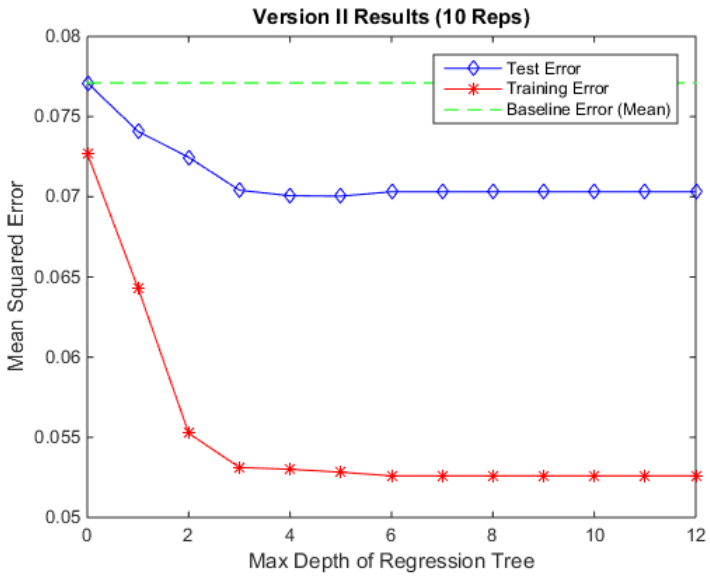
Consequently, our results were rather poor, with clear overfitting even at shallow maximum depths. The following graphs displays our initial performance (averaged over ten random partitions of the dataset into training and test data).



3.2 Version II

Our next iteration included an expanded feature set (size twelve) and improvements on our algorithm. We addressed the issues from our first version; to fix the variance problem, we instructed the algorithm to only partition on the middle sixty percent of the data. We attempted several different approaches to splitting on categorical features. Originally, we tried the brute force approach of examining every possible split. For categorical features with a large number of possible values, however, this method is very slow. We subsequently tried random partitions, hoping to arrive at something close to the optimum after a large number of attempts. Finally, we opted to sort the data by category, sorting the categories according to their average median grades. This imposed an order on the categories and allowed us to partition between them in the same manner that we did for numerical features.

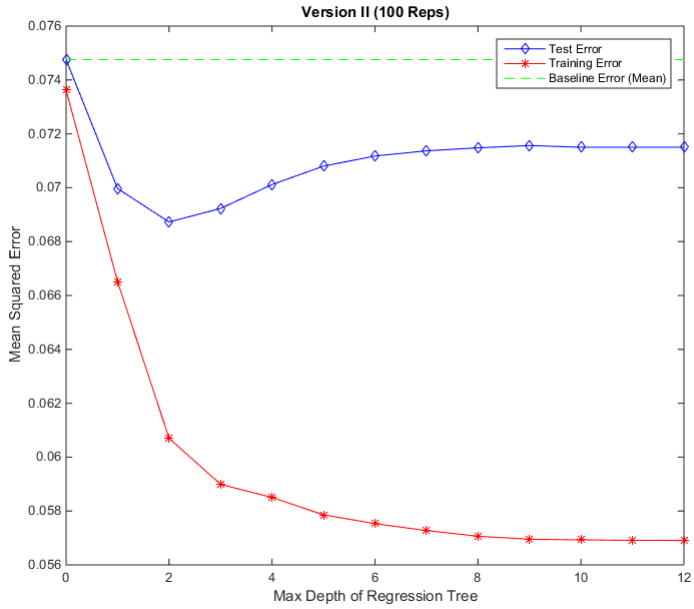
The performance of our algorithm increased significantly:

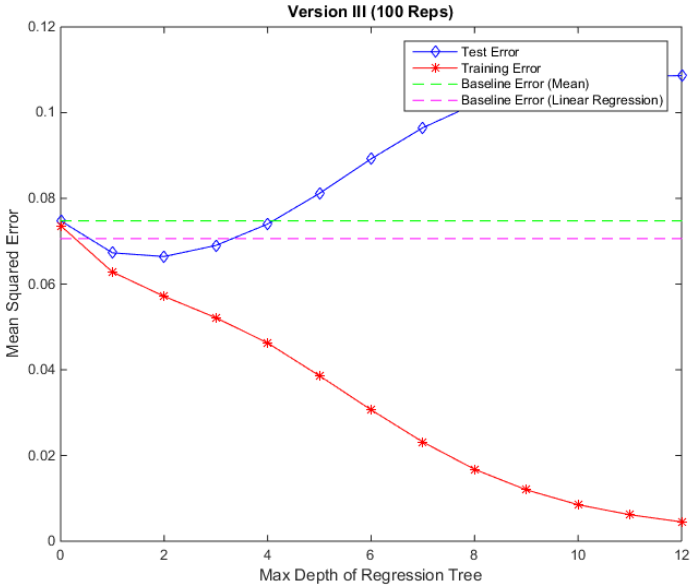


In Versions II and later, the error decreases, so we are able to identify features with predictive power. Commonly chosen features include course enrollment (admittedly not a syllabus feature, but one included with median data), percent sign frequency, number of testing words (quiz, exam, etc.), presence of labs, and department.

3.3 Version III

In our final version, we made an additional change to our splitting criteria, removing the 'middle sixty percent' constraint in favor of weighting each subset's variance according to its size. This further improved our performance (for our final test, we ran our algorithm over one hundred random partitions to get a better sense of its general performance). The top graph is 100 repetitions of Version II, for comparison.





Though Version III overfits more heavily at higher depths, it reaches a better minimum than Version II (0.0664 versus 0.0687). For the Version III graph, we also included linear regression with regularization as a baseline (tested with the same 100 random partitions); we note that our algorithm beats its best minimum of 0.0706.

As a final experiment, we modified our parser to treat every word found in each syllabus as a new feature, such that the feature for a given word was '1' if present in a syllabus and '0' otherwise. This produced over ten thousand features. Our algorithm performed very slowly on such a large feature set, so we only managed to run it on one random partition up to depth four. The results were underwhelming, failing to improve on our hand-picked feature set.

With some modifications (such as saving the indices of our examples sorted according to each feature, rather than recomputing at each split), we could speed up the algorithm considerably and test this feature set for more repetitions and larger depths. This would be a good area for future work. In addition, we note that Versions II and III of our algorithm both have significantly varying performance on different random partitions, sometimes reaching minimum errors as low as 0.055. Analyzing the factors that contribute to these discrepancies might provide insight into possible improvements for our dataset and algorithm (for

instance, if the algorithm performed better when each department had examples in the training and test sets).

4 Conclusions

We identified recursive feature partitioning as a means of studying the correlation between the text in a course's syllabus and the course's median grade. With only 259 syllabi and less than half of Dartmouth's departments represented, we were still able to find notable results. We see that, with an appropriate feature set, syllabi text indeed correlates to course grade distributions. As we know that there is correlation between a course's median grade and certain features available in a course's syllabus, exploring additional features in a larger and more representative dataset could potentially lead to stronger results. This sheds light on how interesting a large scale study could be.

5 Appendix A: Implementation Details and Attributions

We would like to properly recognize Suri Like for her Wordcount.m script[5]; this code was quite helpful early on in identifying the initial direction of the parser code. We made use of Xpdf's *pdftotext* program[6] to convert our .pdf files to .txt files. We compiled a list of stopwords (found in OutsideVocabDatabases/stopwords.txt) from a number of sources.[7]

Otherwise, all code submitted is our own.

[5]Suri Like's Wordcount.m Script
[6]http://www.foolabs.com/xpdf/home.html
[7]ranks.nl stopword listxpo6.comGoogle Stopword ProjectPMTK