Detecting Web Pages with Events

Matan Zinger
mzinger@stanford.edu

Junmin Hao
junmh@stanford.edu

ABSTRACT

The semantic web is a concept being promoted by the World Wide Web Consortium, with the vision of allowing machines to "understand" web-pages, without maintaining scrapers of human-visible HTML. This structured markup in web-pages have recently been powering features in Google Search, such as Rich Snippets[0]. The project purpose is detecting web pages that contain information about events, based on the main text in the page. Thus making it possible to reach the websites' owners, suggesting they add structured data for the events in their site.

1. INTRODUCTION

Traditionally, web pages were designed to be read by human beings. Web visualization technologies have been evolving significantly fast over the last two decades, allowing websites to provide rich visibility to their users. In parallel, a growing trend of "scrapers" has been taking place, powering systems that utilize information aggregated from other sources in the web. While some of these "scrapers" may be considered spammers, others actually create a mutable benefit for both the content provider and consumer (e.g. a video store website presenting IMDB reviews and ratings before rental, and encourage customers to add a review to IMDB afterend).

The biggest pain is that making machines "understand" information modelled in a fashion designed for human consumption (such as HTML) is a tedious task that requires frequent maintenance. An alternative might have been having websites provide public API to access the data presented in their web pages, but a proprietary API would also make data integration an effort-consuming task.

During the last decade, the semantic web initiative created open standards for defining structured data on top of web-pages, as well as standards for repositories and query languages on top of them, making it possible to build semantic applications.

Google also contributes to the semantic-web vision, by utilizing structured markup detected in web-pages in different Google Search features. The most famous of which being Rich-Snippets. These improve both the experience of a user going through web results, as well as allowing website to surface further information in its search result snippet.

The purpose of this project is to utilize signals from webpage textual content in order to detect pages that contain information about events, making it possible to make a mass outreach to event website owners, with a suggestion (and instructions) regarding adding events structured data to their websites.

The approach we take is building a quick basic classifier, keeping in mind possible optimization steps that could be taken in the future, in order to evaluate the initial performance and choose the enhancements that would yield the best impact on our classifier.

Our preference is as high precision as possible, even by hurting recall, since the effect of sending "spam" suggestions to webmasters is very undesirable.

2. DATA GATHERING & PRE-PROCESSING

In order to gather a labeled set of web URLs, we have used the Open Directory Project (ODP)[1], which contains an hierarchical set of over 780,000 categories. In each category, a few URLs that were placed after a human process of content evaluation.

In order to gather positive examples, we have selected all hierarchical categories where the leaf category is "Events" (e.g. 'Top/Sports/Running/Events', 'Top/Arts/Entertainment/Events').

In order to gather negative examples, we have selected all category paths that did not contain an "Event" topic in any of the path element (e.g. 'Top/Arts/Crafts/Paper', 'Top/Business/Arts_and_Entertainment/Photography /Photographers').

Once gathered the lists of positive and negative example URLs, we have taken the following processing steps per each URL:

- Fetching & Text Extraction
  The 'boilerpipe'[2] open source (under Apache License 2.0) library in order to fetch the content of these URLs, parse the HTML, discard boilerplate[3] and extract the main text content of the URL.

- Stemming
  In order to avoid contribution of specific inflections, words were reduced to their root form (stem) using a python implementation of the Porter stemmer[4].
- Transforming into term vector
  The last data-processing step we've done is transforming each document into a term vector (by using 80,000 different tokens as features).

Having a set of 3,000 examples, represented as term vectors, we splitted is it a random fashion into a training set (with 70% of the documents) and a test set (the rest of the documents). While doing so, we made sure each ODP category in use has representatives in both training and test sets.

The training set was splitted in a similar fashion into subsets, making it easier to measure the learning rate, as well as performing cross validations.

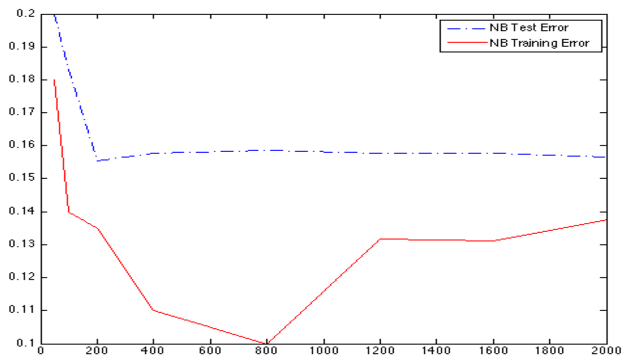
3. CLASSIFICATION

The algorithms we have evaluated were:

- Naive Bayes with Laplace smoothing.
- Linear L2-regularized L2-loss soft-margin Support Vector Machine.

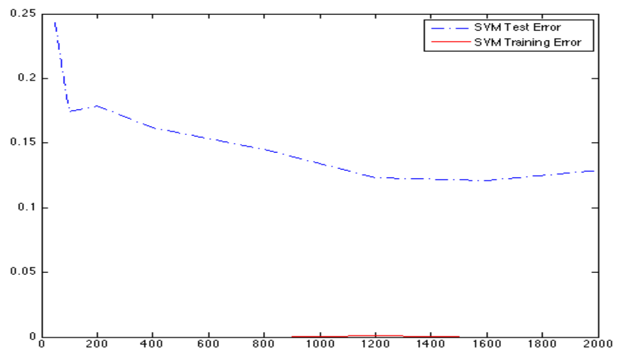
Per each, we trained a model with the different subsets of the training set (each produced a classifier). We measured the error rate of both the training set and test set with each classifier.

+ Naive-Bayes



The most indicative terms for events: *m42, ct, marathon, fundraiserjoin, m40.*

+ Support Vector Machine



Since both algorithms have shown similar levels of accuracy during the first execution, we have also measured the Precision and Recall (The numbers in the table below are the numbers of positives or negatives divided by the number of test examples):

	Actual: Positive	Actual: Negative
NB Positive	0.7658	0.1287
NB Negative	0.0277	0.0777
SVM Positive	0.7469	0.0821
SVM Negative	0.0466	0.1243

Naive-Bayes Precision: 0.8561; **Recall:** 0.9650.
SVM Precision: 0.9009; **Recall:** 0.9413.

4. PERFORMANCE ANALYSIS

As these are the components of our classification pipeline, the following parameters essentially determine the classifier quality:

- Webpage cleansing
- Text pre-processing
- Choice of terms list
- Features (except term frequency)
- Choice of algorithm
- SVM-Choice of algorithm parameters: regularization term, soft margin C.
- Choice of test set

While building the initial execution, we had ideas of how to improve all of the above (in more than one manner). However, we intend to try and evaluate which of these could yield the best improvement in classification quality.

A strongly noticeable problem is the 3 meaningless terms out of the 5 most indicative terms in the Naive-Bayes classifier. This can be either due to the intrinsic flaw in NB, or due to our over-permissive policy to consider every possible term. However extremely noticeable, we preferred further analyzing the results, before choosing our next step.

Considering our strong preference for precision, the SVM seems like a better candidate to be examined (given the lower false-positive rate it yielded). Judging from the learning curve - both the extremely low training error rate, and the fact that the marginal improvement in accuracy is negligible, indicate a problem of overfitting. In order to reduce variance, useful steps would be using less features and then using more training examples.

In addition, in order to reduce the variance of the SVM-based classifier, C should be reduced.

5. PERFORMANCE OPTIMIZATION

While the previous steps were mostly about gathering data and establishing a classification baseline, the next purpose was improving quality (mainly - precision.) We therefore engaged in the following:

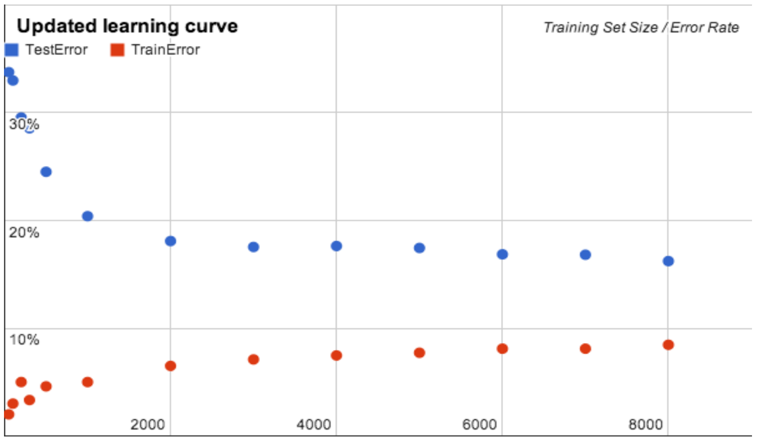
- Increasing Training Examples Set

The first action we took towards solving the variance problem was to gather more training examples.

In order to gather more negative examples, our boiler-pipe based pipeline was fetching the data of 3700 additional non-event URLs, all of which were retrieved from the ODP dataset. we found it important to increase the proportion of the negative examples, after noticing the positive examples are strongly correlated with stop-words. We eventually increased the negative portion from 20% to 42%.

In order to retrieve additional positive examples (after using all positive examples found in the ODP dataset), we have used the Sindice© (semantic web index)[5], and executed a query for URLs that have already been marked up with Structured Data of type 'http://schema.org/Event'. We suspected that using an additional source for URLs (other than ODP) would increase the diversity of texts, and might be useful in facing the overfitting sample.

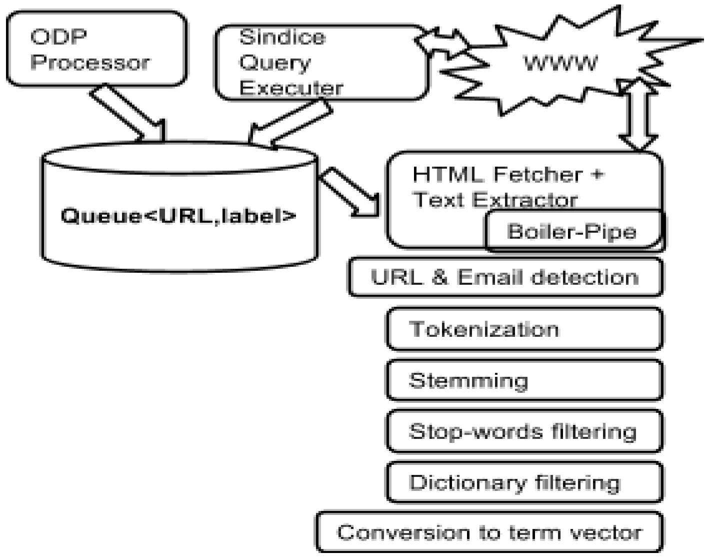
This yielded additional 3500 positive examples, Thus concluding an increase of the data-set from 3000 to 10,200 documents.



Using a 2000 documents test set, the plot still seem like a typical high-variance case, but the increased set still improved accuracy by 1.86 percentages.

- Additional Pre-Processing Steps

At the previous step, we've noticed the indicative terms seems radnsom (such as 'm42'). We've added the following steps in order to eliminate these terms:
(1) Filtering non-dictionary terms: We have used the English words list included in the Unix file system (at `/usr/dict/words`) to filter non-English words.
(2) Identification of URLs and email addresses: We have defined regular expressions for this purpose, and replaced these instances with a pre-defined tokens ('EMAIL' / 'URL').
(3) Filtering of single-letter terms.
(4) Filtering stop-words: useing the Ranks.NL[6] list.



These additional steps have reduced the number of tokens from 198,742 to 25,048. As overfitting is our main problem, and reducing the feature set is one of the main methods in solving it, we chose eliminating meaningless features as the first step in this regard.

This activity have also made the indicative terms list appear to be much more relevant for the events case: 'event', 'festiv', 'toggl', 'ticket', 'particip', 'saturday', 'share', 'mathemat', 'password', 'address', 'venu', 'sponsor', 'descript', 'sunday', 'workshop', 'attend', 'twitter', 'date', 'circl', 'notifi', 'friday', 'miss', 'qualiti', 'busi', 'music'.
(Tokens appear in their post-stemming format.)

- Tuning (Reduced) Value of C

Another step aimed at reducing variance - we have used the k-fold cross validations technique in order to examine different values of C, with setting K=6, and calculating the error rate for each value:

$$\varepsilon_{(C=c)} = \frac{1}{K}\sum_{i=1}^{K}\sum_{j=1}^{|S_i|} 1\{h_{(C=c, S\setminus S_i)}(x_i^{(j)}) \neq y_i^{(j)}\}$$

Where:

* S_i : ith slice of the examples set, out of K equally-sized slices.

* $h_{(c,\widehat{S})}$: the hypothesis function generated when training the SVM algorithm with C=c over a subset $\widehat{S}$ of the examples set S.

* $x_i^{(j)}$: the jth data point in the examples subset $S_i \subseteq S$.

* $y_i^{(j)}$: the label corresponding to $x_i^{(j)}$.

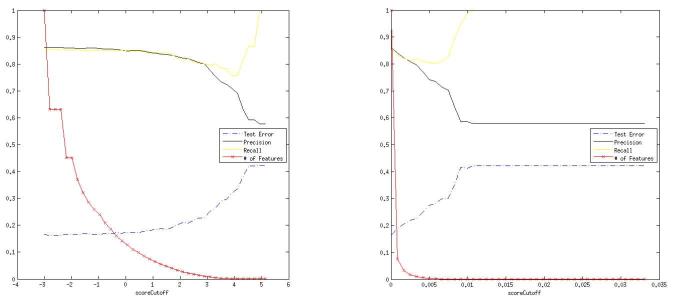
Using the 6-fold method, we've also counted the number of true-positives, false-positive, true-negatives and false-negatives, in order to calculate the precision and recall rates for each possible value of C. The results:

C	Test Error	Precision	Recall
1.0000	19.07%	83.64%	83.32%
0.5000	18.74%	83.93%	83.59%
0.2500	18.37%	84.18%	84.02%
0.1250	18.07%	84.45%	84.27%
0.0625	17.58%	84.87%	84.72%
0.0312	17.13%	85.17%	85.23%
0.0156	16.82%	85.45%	85.46%
0.0078	16.73%	85.66%	85.36%
0.0039	16.55%	85.86%	85.44%
0.0020	16.4%	85.93%	85.65%
0.0010	16.51%	85.93%	85.43%

The optimal value found is C = 0.002; the accuracy has been improved by 3.07 percentages, and precision by 2.39 percentages.

- Feature Reduction

We were looking to further eliminate features; First, we implemented "Filter Feature Selection", with two feature scorers: mutual information, and indicativeness score (described in HW2 Q3b). Accuracy did not improve, as seen below:



However, the positive outcome was raising the necessity in a larger portion of negative examples, since in the initial 80%-positive data set, even a dummy model that always predict positive yields 80% accuracy. Therefore we've increased the negative proportion to 43% (as previously described.)

The next method we tried was Backwards Feature Search (over the 25,040 features in the 10K data set). First, we removed infrequent tokens (i.e. with less than 8 appearances), to reduce feature set size to 10K. In order to speed the search, we've used 2-fold cross validation - the fastest possible setting.

At each backwards-search step, the program iterates over all features, and looks for the feature f_i that minimizes: $\varepsilon_i = \frac{1}{K}\sum_{j=1}^{K}\sum_{t=1}^{|S_j|} 1\{h_{(F\setminus\{f_i\}, S\setminus S_j)}(x_i^{(j)}) \neq y_i^{(j)}\}$,
Where S_i , $x_i^{(j)}$, $y_i^{(j)}$ are as previously described, and $h_{(\widehat{F},\widehat{S})}$ is the prediction function generated when training over a subset $\widehat{S} \subseteq S$, and a subset $\widehat{F} \subseteq F$.

This feature shall be discarded at the end of the step. After 63 iterations (only 0.6% of all features), the accuracy has increased from 83.6% to 83.85%.

However, each iteration took 40 minutes to run, and completing the algorithm could take up to a month. Therefore we looked to speed up this process, by executing each iteration in a parallel fashion: the errors per each feature removal (ε_i) shall be computed in parallel, and when all tasks complete, the minimal results indicates which feature to drop. However, due to MATLAB licensing issues, we could not have ran this version before the deadline.

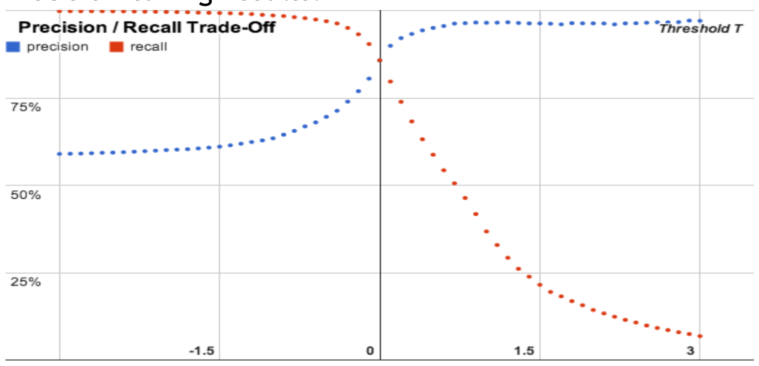
- Tuning for Better Precision

Training a SVM model yields values for w and b. Given a data point x, the amount $\gamma = w \cdot x + b$ is the distance from x to the decision boundary. By default, when $\gamma > 0$, the model predict it as a positive, and negative when $\gamma < 0$. The larger γ is, the more confident the model is about x being positive.

Therefore we can use a threshold T (different than zero) to modify how we make the prediction, that is, when $\gamma > T$ yields a positive prediction, and negative otherwise. By tuning T, we could control the confidence for positive predictions (i.e. precision).

It's important to note that the threshold actually controls a precision/recall tradeoff, and by increasing precision, the recall rate decreases.

Precision tuning results:



The precision seem to have an upper bound around 96.25%, where threshold = 0.7, recall rate = 50.56%. As the threshold further increases, the recall rate eventually drops to 6.83% at T=3, at the precision rate only increased to 97.04%.

6. Summary & Conclusion

Our work was mainly composed of 3 major activities:

First, we followed the practice of quickly building an initial version of the classification system (to be gradually enhanced based on experiments results.) We've implemented a basic platform to fetch and process web pages into term vectors (based on common practices), implemented the two text classification algorithms learned in class.

Second, we've analyzed the results, which indicated an overfitting problem.

Third, we've applied the practices given in the learning theory lectures, and implemented all the methods efficient at reducing variance; The most effective method proven to be reducing C, which improved accuracy by 3.07 percentages. Second, adding training examples increased accuracy by 1.86 percentages. Third, backwards search improved accuracy by 0.25 percentages, and further iteration could yield more improvement.

Last, we've tuned the level of confidence for considering a web-page as being about an event, and found a value for which precision is over 96%, and recall is 50.5%. While in general this is not necessarily a good result, it could very well fit the specific case we would like to solve;
Our classifier could potentially make it possible to find half of the event web-pages on the web, and approach their owners with high (>96%) confidence, to suggest adding semantic event markup.
That being said, testing over larger tagged web-pages sets is required in order to verify this statement.

7. Future Enhancements

In order to further improve the classifier, the first step we would have take is to complete the feature reduction process, by being able to make a parallel execution of the backwards search algorithm. This can initially be done on the cores of a single machine, as we described in the last paragraph under "Feature Reduction".

Another possible enhancement is in the preprocessing pipeline: currently phone numbers, dates and geo-locations are being filtered out on the account of not being in the English words list. However, since dates and locations are the major properties of an event, we speculate that these could be indicative signals. In order to examine that, a method to detect such terms should be plugged in, and map them into a constant term (similar to what's being done with URLs and email addresses).

REFERENCES

[0] *"Rich Snippets - Events"* - http://support.google.com/webmasters/answer/164506
[1] http://www.dmoz.org/docs/en/about.html
[2] http://code.google.com/p/boilerpipe/
[3] Christian Kohlschütter, Peter Fankhauser, Wolfgang Nejdl. ***"Boilerplate Detection using Shallow Text Features"***
[4] M.F.Porter. ***"An algorithm for suffix stripping"***.
[5] *Sindice© - A semantic web index* - http://sindice.com/
[6] *Ranks.NL stop-words list* - http://www.ranks.nl/resources/stopwords.html