SocialSense: Quantifying Social Media Linkage Across the Web

Joel Shackelford, Alexander Spizler, Yang Xu
{ jshack02 , aspizler , yxqc4 } @ stanford.edu

Abstract— **We present SocialSense, a quantification of social media linkage across the web. We utilize content sourced from monthly crawls of the entire web to predict the category of links likely to dominate a website. The web data is conditioned into a set of interesting features based on intuition of the data items and ease and availability of common attributes. These principals may also be applied to explore the darker side of intent by identifying nefarious and potentially malicious links. This is done through the aid of various learning algorithms and implementations. In addition, we address the computational expense of the algorithms tested, their performance and the implications of the results.**

I. INTRODUCTION

SocialSense is about characterizing web linkage, specifically websites that are likely to contain links to Facebook, while demonstrating extensibility to more insidious and malicious linkage in a computationally efficient manner. The Common Crawl Corpus[8], a nonprofit dedicated to education and prosperity through an open web, crawls the entire web monthly and maintains this massive data set for open use. We parse our input features from HTTP interactions and metadata associated with crawling a given page. We also take advantage of Amazon Web Services (AWS) to interact with this data set through cloud based queries, aggregation tools such as MapReduce and cloud computing instances like EC2. We show that the application of several common machine learning algorithms, such as logistic regression and Support Vector Machines (SVM), produces a marked differentiation between websites with social media linkage and those without.

II. RELATED WORK

While other work has focused on identifying web intent based on user expectations or user intent [1] SocialSense's methodology uses keys and features associated with the underlying implementation of the site, HTTP headers and retrieval statistics. Other work [6] focuses more on brand safety, ad safe concerns, maximizing audience relevance and contextual targeting while also listing malicious detection of URLs as an advertised feature. Here we focus strictly on the intent of a given site and leave application and monetization to future work. There is a vast amount of research and development in the area of natural language processing and several tools [3] [4] [5] that focus mainly on text processing and sentiment analysis to judge intent. While this remains a future area of study for SocialSense it was not intimately explored. [2] categorizes web intent, but their intellectual property was not available for us to explore. To define truth in pursuit of quantifying malicious intent we leveraged blacklists of known suspect URLs [7]. While there is work in this area, current methods rely solely on known malicious links. SocialSence's goal is to use features of the website other than the links themselves to understand the "fingerprint" of a malicious website.

III. DATASET AND FEATURES

Data storage and processing takes place through AWS. Therein we use Java[13], Python, and Hadoop to pre-process the data, i.e. extract a feature vector from each website. It is therefore important to first define an entry in our data set. The Common Crawl repository, from where we glean all our data, is constructed by capturing HTTP requests, their responses and the associated metadata. This data is stored in WARC[10] format and is identified by a Universally Unique Identifier(UUI)[11]. The HTTP responses and metadata represent data supplied to a querying browser and are the main concern for us. We therefore key our training and test data on this same UUI.

In order to simplify the massive amount of data contained in a crawl of the entire web we condition the data to parse only the features in which we are interested. Initial attempts focused on items that were of minimal effort to parse and collect. These features included: retrieval time, expiration and page size as given by the HTTP response as well as total number of links on a page summed from the WARC format. As the proficiency of working with the WARC files in the AWS ecosystem increased we were able to infer additional information, such as the presence of Amazon

ads and the extension of Google APIs (e.g. Google maps, Google sign in). Two additional features, server type and domain extension were trivial to ascertain, but took some manipulation to provide as inputs into the learning algorithms discussed below. The difficulty lie in the fact that we required individual binary features, but there was an indeterminate set of values. To solve this problem we took a general history over a broad range of samples and found the most commonly used values. Apache and Nginx were common server types, but there were on the order of thousands of different types, unbenounced a priori. As a reasonable solution we chose the 80 most common server types which identified more than 90% of our data.

Below is a table of all features extracted from the Common Crawl web data. Note that not all features in Table I are considered in all plots in subsequent sections.

Features	Description
Unique Site ID	String
Number of page links	Integer
Page size	Integer
Contains Amazon adds	Boolean
Extends a Google API	Boolean
Contains expiration	Boolean
Domain extension	Boolean[261]
Server type[1]	Boolean[80]
Retrieval time	Integer
Text vectorization[1]	Boolean[300]

TABLE I

FEATURES EXTRACTED FROM EACH WARC FILE

IV. METHODS

For this binary classification problem, we apply our own implementation of logistic regression as well as SVM with different kernels available in lib-svm [12].

Given $i = 1, \ldots, m$ websites, with $j = 1, \ldots, n$ features extracted from the website, and a feature vector, $x \in \Re^{n+1}$, where $x_0 = 1$, we want create a classifier that predicts whether or not a new website links to Facebook, $y \in 0, 1$.

In order to do this, we can use the sigmoid function as our hypothesis function,

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}} \tag{1}$$

where if $h_\theta(x) > 0.5$ then y is most likely 1, and most likely 0 otherwise. To do this, we define the likely value of y for any one training example as

$$p(y|x;\theta) = (h_\theta(x))^y (1 - h_\theta(x))^{1-y}$$

and maximize the likelihood of the m independent samples. Since the parameters that maximize a function are the same for the log of a function, we can solve for the log-likelihood, written as

$$\ell(\theta) = \sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})).$$

Taking the gradient of $\ell(\theta)$ with respect to θ and setting it to zero, for each θ_j we get

$$\frac{\partial}{\partial \theta_j} \ell(\theta) = (y - h_\theta(x^{(i)})) x_j^{(i)}$$

Finally, the method we use to maximize $\ell(\theta)$ is stochastic gradient ascent. We iterate until convergence, updating each θ_j in θ given the old value of θ_j, the above equation, and a learning rate, α which controls the step size for each iteration. This is given by

$$\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)}. \tag{2}$$

Once we solve for θ, we make predictions for y given new values of x using equation (1).

For SVMs, we want to find a separating hyperplane by optimizing

$$\begin{aligned} min_{\gamma,w,b} \quad & \frac{1}{2}||w||^2 \\ s.t. \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1, \quad i = 1, \ldots, m \end{aligned} \tag{3}$$

where w and b are parameters of the hyperplane, $||w||$ is the orthogonal distance between the support vectors and separating hyperplane, and γ is the geometric margin.

By applying the kernel trick, we can then map high dimensional feature vectors to a much lower dimensional space, allowing us to use a higher dimensional hyperplane for binary classification.

V. EXPERIMENTS

In order to find the best binary classification algorithm for this problem, we run logistic regression and SVM on 53,779 samples taken from one slice of the Common Crawl[9]. We use simple cross-validation, training each classifier on the first 70% of the samples. Of the samples, 39.13% of training data have links to Facebook while 60.87% do not. The testing data has similar proportions at 39.66% and 60.34%.

From the testing results, we count the number of true positives (TP), false positives (FP), true negatives

[1]Features not vectorized in time for inclusion in this paper

(TN), and false negatives (FN). We then calculate the accuracy, recall, and precision, defined as

$$Accuracy \equiv \frac{TP + TN}{TP + FP + TN + FN} \quad (4)$$

$$Recall \equiv \frac{TP}{TP + FN} \quad (5)$$

$$Precision \equiv \frac{TP}{TP + FP} \quad (6)$$

Here, defining sites that link to Facebook as relevant, accuracy is the fraction of samples that are labeled correctly, recall is the fraction of all the relevant sites labeled as relevant, and precision is the correctness of a result labeled as relevant.

For logistic regression, we need to choose two values, the learning rate, α, and the threshold for convergence. We select an initial value of α to be 0.01, then, after every 3 iterations, we reduce α such that $\alpha := \alpha/3$. This gives a better estimation than using a constant large α and a quicker estimation than a constant small α. We define convergence to be when $||\theta_{old} - \theta_{new}||_2^2 < 0.01$. Through trial and error, we found that this set up converges to the same results more quickly than running additional iterations at each learning rate or having a lower threshold for the L^2 norm of the change in θ.

We also produce plots for the area under the curve (AUC) and the area under the precision recall curve (AUPRC) by varying our classification threshold between 0 and 1, and measuring the TP, FP, TN, and FN rates. For the AUC we can plot the TP rate versus the FP rate, and for the AUPRC we can plot the precision versus recall as the threshold varies. This allows us to see the effects of different values of the threshold, allowing users of this algorithm to select a threshold based on their priorities, e.g. choosing to find nearly all websites that link to Facebook despite having many false positives.

With SVM, to obtain better prediction accuracy, we first use the data-scaling tool provided by lib-linear to scale our feature data. Since some of the features have significantly higher values than others, data-scaling allows the maximum value for all features to be the same, improving our prediction accuracy compared to using unscaled features. Additionally, we assure that we have equal numbers of positive and negative samples for both testing and training data.

We then run SVM using the various kernels available in lib-svm, listed in Table II below. We also use the lib-linear classifiers to test the performance of other standard classifiers. We use the default parameters for both lib-svm and lib-linear.

Classifier Name	ID
LIB-SVM:	
Linear	t0
Polynomial	t1
Radial basis function	t2
Sigmoid	t3
LIB-LINEAR:	
L2-regularized logistic regression (primal)	s0
L2-regularized L2-loss support vector classification (dual)	s1
L2-regularized L2-loss support vector classification (primal)	s2
L2-regularized L1-loss support vector classification (dual)	s3
Support vector classification by Crammer and Singer	s4
L1-regularized L2-loss support vector classification	s5
L1-regularized logistic regression	s6
L2-regularized logistic regression (dual)	s7

TABLE II

LIB-LINEAR AND LIB-SVM CLASSIFIERS

VI. RESULTS

Training our logistic regression classifier on the first 70% of samples yield the following confusion matrix, Table III, which compares the classification of testing samples against their targets. Here positive (p) values represent predictions and outcomes where a site links to Facebook, while negative (n) values do not. The accuracy of logistic regression is 69.35%, while the recall and precision are 73.65% and 62.04% respectively.

		Predicted Outcome		
		p	n	total
True Value	p'	4712	1686	6398
	n'	2883	6853	9736
	total	7595	8539	

TABLE III

CONFUSION MATRIX FOR LOGISTIC REGRESSION

We next calculate the AUC and AUPRC in order to understand how different thresholds affect TP and TN rates, precision, and recall. Both plots in Figure 1 show a knee in the curve when the threshold is close to 0.5, the default threshold for logistic regression.

Next, we measure the testing and training errors as the sample sizes increase in Figure 2. For each data point, we take three random collections of samples from the full data set, and calculate the mean and standard deviation for error (1 - accuracy), recall, and precision.

We are also interested in the values at which each parameter converges. Table IV shows the weights of

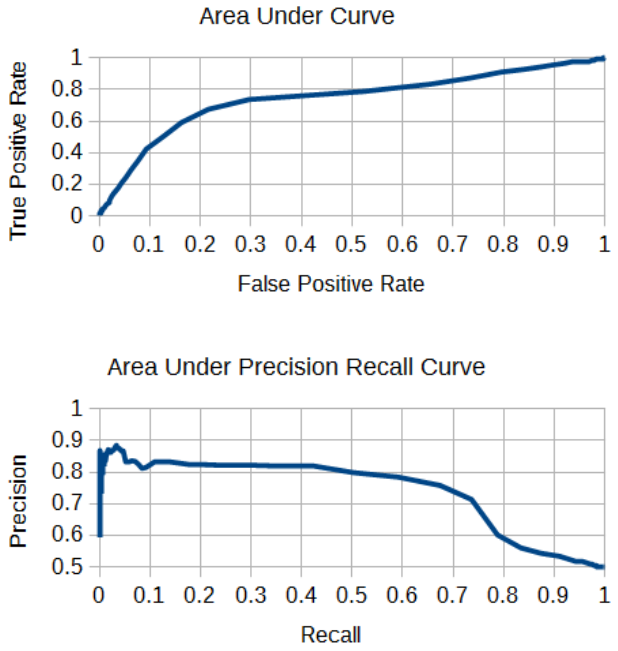

Fig. 1. Plot of AUC (top) and AUPRC (bottom). Non-monotonic decrease at the beginning of the AUPRC is possibly due to precision of small numbers.

several of the more influential parameters for logistic regression.

Finally, we look at the accuracy, recall, and precision of the various SVM kernels provided in lib-svm, t0-t1. The results for the methods in lib-linear, s0-s7, are also shown in Figure 3.

VII. DISCUSSION

Despite lower than desired accuracy, we are able to distinguish to some degree which sites are likely to link to Facebook. By evaluating the values of our parameters in Table IV, it is clear that the number of links on a page has the most influence by far, indicating that the more links a page has, the more likely it is to link to Facebook. Other weaker indicators, such as retrieval time and having a .com domain, show negative correlation with linking to Facebook.

The performance of our logistic regression classifier is poorer than desired, with 30.64% error, while a linear SVM performs similarly. The best performance, with L^1-regularized logistic regression, still has 28.37% error. In order to understand how we could improve performance in the future, we look at the testing and training accuracy versus sample size. If training error were much lower than testing error, we might suspect that the classifier was overfitting to the data. However,

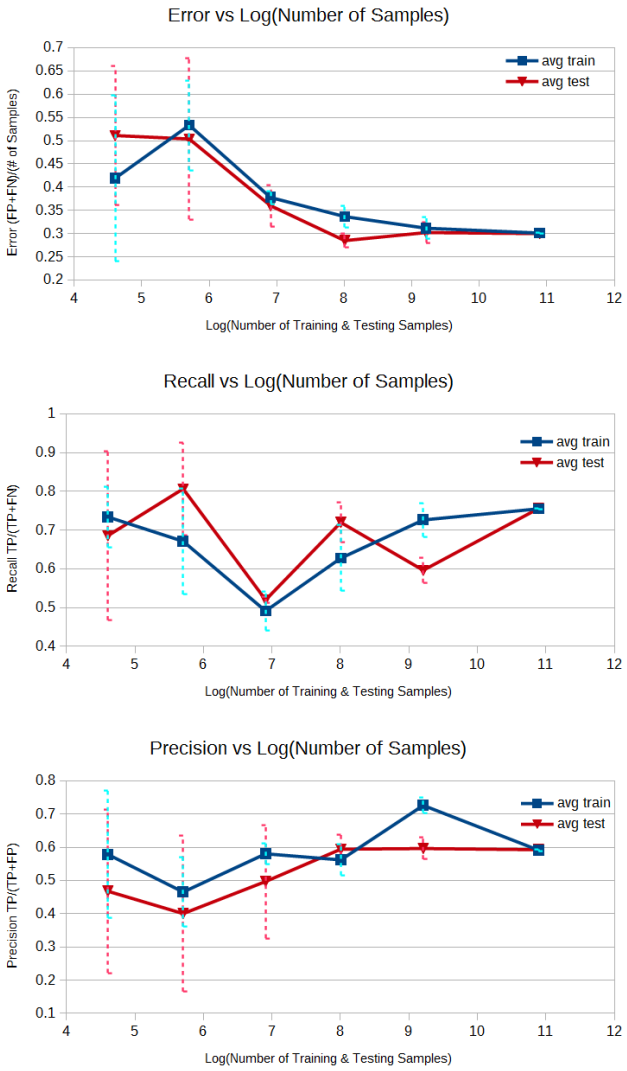

Fig. 2. Plot of error (top), recall (mid), and precision (bottom) for training (blue) and testing data (red) as the sample size increases. Notice that all three converge at the largest sample size

because testing and training error converge, we believe the classifier has a bias problem.

VIII. CONCLUSION

With the features available through the Common Crawl Corpus, we are able to differentiate between sites with links to Facebook and those without. The strongest indicator by far is the number of links on a site, where sites with more links are more likely to link to Facebook. One explanation for the strength of this feature, is that sites with many links, such as news sites and blogs, often also have Facebook pages to which they link.

Logistic regression has similar performance to

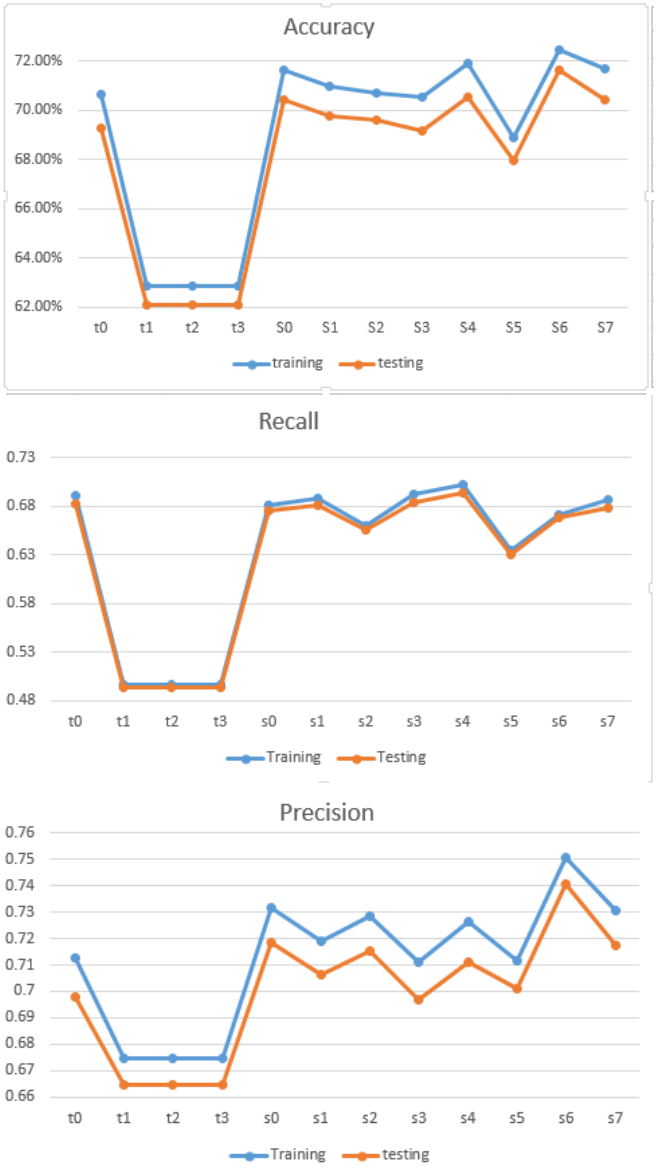

Fig. 3. Plots of accuracy (top), recall (middle), and precision (bottom) of SVMs with various kernels and linear classifiers on training and testing data. Method definitions can be found in Table II

SVMs, at near 70% accuracy. The results of our classifiers are biased, and improvements could be made by either adding features or selecting more informative features.

IX. FUTURE WORK

As previously mentioned, our final results left SocialSence with a small bias problem and less than desired accuracy. One correction for high bias is the addition of features. One fertile ground for intent mining is analysis of a website's raw text. Google provides pretrained vectors as part of its Word2Vec package

Parameter	Value
x0 (offset)	-140.23
number of links	27847.90
Page size	-26.77
Contains Amazon ads	10.28
Extends Google API	42.71
Contains expiration	-47.77
Domain	
.com	-61.35
.org	-32.91
.net	-6.34
.gov	-9.13
.edu	-12.09
.mil	-0.06
.int	-0.50
.co	-1.52
.ac	-1.76
Retrieval time	-398.46

TABLE IV
PARAMETERS AND THEIR VALUES FOR LOGISTIC REGRESSION

that represents a 3,000,000 word vocabulary with a 300 dimensional array per word. This array is based on copious English text from multiple news sources over time. In an attempt to gain additional content-rich features we could sum these values for every word on a given page, normalize this vector by accounting for page size, and include this as a feature in logistic regression and SVM.

An extension to the current approach to defining truth in the malicious realm is to match a link to a list of known suspect URLs. Future endeavors could also utilize unsupervised learning and clustering to avoid the postmortem analysis and even prewarn users and content distributors when suspicious activity is logged over time.

Additional gains may be realized by optimizing the data conditioning and machine learning algorithm's execution in the AWS cloud. Given the cost associated with every could compute cycle and the size of the data set we deal with, this optimization would be requisite if heavy future use is predicted.

ACKNOWLEDGMENTS

We thank Professor Andrew Ng and the teaching assistants for their work in CS229 and for the opportunity to build this system. We also recognize the Common Crawl Foundation and their dedication to a truly open web as well as Amazon for providing educational and research grants.

REFERENCES

[1] Improving Semantic Consistency of Web Sites by Quantifying User Intent

[2] https://developer.similarweb.com/website_categorization_API

[3] http://saplo.com/technologies

[4] http://www.datumbox.com/

[5] Word2Vec: https://code.google.com/p/word2vec/

[6] https://zvelo.com/

[7] http://www.malwaredomainlist.com/

[8] Common Crawl Corpus: http://commoncrawl.org/

[9] CC-MAIN-20150728002301-00000-ip-10-236-191-2.ec2.internal.warc

[10] ISO/IEC 28500:2009, Information and documentation – WARC file format

[11] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, http://www.rfc-editor.org/info/rfc4122

[12] http://cs229.stanford.edu/materials.html

[13] Hicklin, J., Boisvert, R., et al. JAMA: A Java Matrix Package. http://math.nist.gov/javanumerics/jama/

[14] Internet Storm Center: Suspicious Domains. https://isc.sans.edu/suspicious_domains.html