Boston College

Department of Economics

ECON 3389

Big Data

Detecting Malicious URLs using Logistic Regression, Support Vector Machines, and a Random Forest Classifier.

James LeDoux

Chrsitian Cofoid

Zach Angell

Spring 2016

1 Introduction

If you do not have a Mac, you need to be careful of linking to malicious URLs. It is far from practical, however, to scan through list after list after list of URLs to determine which are malicious and which are harmless. In this paper we aim to classify whether a given URL is malicious or not. We apply the methods of logistic regression, random forest classifiers, and support vector machines to classify URLs as malicious or benign from a sample of 56,000 URLs over approximately 3.24 million independent variables.

2 Our Data

Our data were derived from the University of California, San Diego's malicious url data set, published by Justin Ma et. al in the 2009 *International Conference on Machine Learning*. It features 56,000 observations, each a url, and approximately 3.2 million features. The dependent variable Y is a binary variable, indicating whether the url was deemed malicious, defined as being involved in a criminal scam, and the independent variables are represented as a sparse array, since dense representation would be too big to be contained on a personal laptop. The data is indeed quite big. We conducted a simple train-test split, setting aside 25% of the data as a final test set to evaluate and compare the models' relative performances.

3 Method 1: Logistic Regression

3.1 Theoretical Underpinnings

We begin by defining the maximum likelihood estimator. Let $(\Omega, \mathcal{A}, \mathbb{P})$ be a probability space, and define the random variable $Y : \Omega \to \mathbb{R}$ with probability density f, which has a parameter θ Suppose we take a sample of size n from the Y distribution. Then, we define the likelihood function as

$$Lik(\theta) = \prod_{i=1}^{n} f(Y_i),$$

and the log-likelihood function as $\ell(\theta) = \log \prod_{i=1}^{n} f(Y_i)$. Then, if we want to estimate the parameter θ, we need to find

$$\hat{\theta} = \arg\max_{t \in \Theta} \ell(t), \tag{1}$$

where $\Theta = \{t : t \text{ is a parameter for } Y\}$. This estimator $\hat{\theta}$ is called the maximum likelihood estimator.

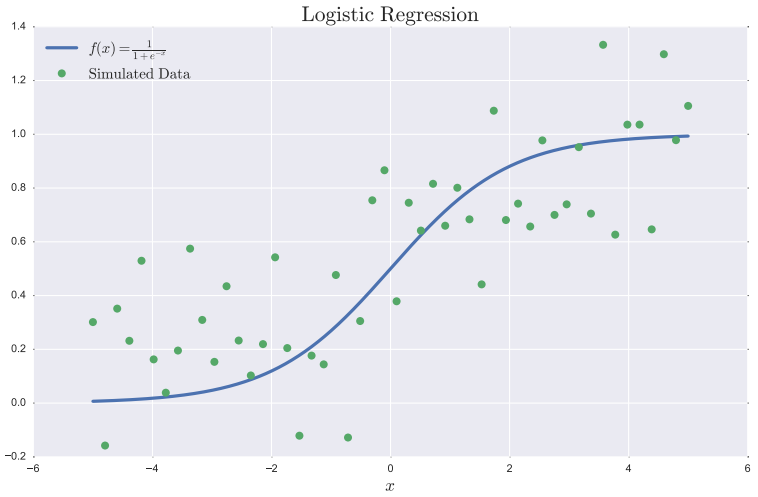
We assume that $Y_i \sim \text{Bernoulli}(p_i)$. Then,

$$\mathbb{P}(Y_i = y) = p_i^y (1 - p_i)^{1-y},$$

for $y \in \{0, 1\}$. Furthermore, suppose that $Y_i \perp Y_j$ for $i \neq j$. Now, let $X : \Omega \to \mathbb{R}^k$ be a random vector on the same probability space, and suppose that we draw a sample of size n from the X distribution. We want to estimate the parameter $\beta \in \mathbb{R}^{k+1}$ such that

$$\mathbb{P}(Y_i = 1 | X_i) = \frac{1}{1 + \exp(-\langle \beta, X_i \rangle)},$$

where $\langle \beta, X_i \rangle = \sum_{j=0}^{k} \beta_j x_{j,i}$, for $x_{0,i} = 1$ for all $i = 1, 2, \ldots, n$. We find the estimator $\hat{\beta}$ of β by using ML estimation stated in (1). For a regression of Y onto X for $X : \Omega \to \mathbb{R}$, we obtain a model that looks like the following graphic.



Our data has a large number of regressors, so we apply the ℓ_1 norm penalty to perform feature selection. We introduce a shrinkage penalty λ, which will smooth the model by penalizing for high coefficient values in the function to be optimized. The new estimator we are finding is

$$\hat{\theta} = \arg\max_{t \in \Theta} \ell(t) - \lambda \left|\left|\beta\right|\right|_1 .$$

This is similar to the estimator used in the LASSO, except, since this estimator is maximizing log likelihood rather than minimizing sum of squared residuals, the penalty term is subtracted rather than added.

3.2 Results on URL Data

We implemented scikit-learn's `LogisticRegressionCV` module, using five folds and an ℓ_1 penalty, fitting the model to the `xtrain, ytrain` data. The optimal inverse regularization parameter c for this model, found by the `LogisticRegressionCV` module by applying 5-fold cross validation on a set of 10 C values, was 2.7825. The relationship between C and MSE in shown in figure 1. With this optimal regularization parameter determined, we then predict upon the test data, obtaining 98.77% accuracy on 14,000 out-of-sample observations. To interpret this further, we constructed the confusion matrix below (figure 2).

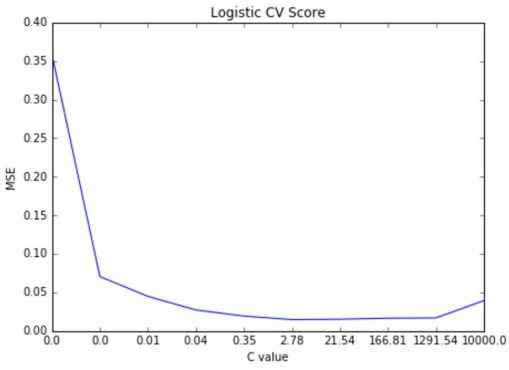


Figure 1: Cross-validated MSE of logistic regression model with ℓ_1 penalty

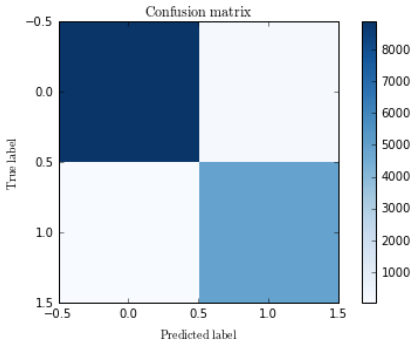


Figure 2: Upper left: true negative, upper right: false positive, lower left: false negative, lower right: true positive

Figure 2 shows that, of the 171 observations misclassified, 48 were false negatives and 123 were false positives. Seeing more false positives than false negatives is an encouraging sign for a malicious url classifier, since false positives in an application of such a model in a real-world scenario would would be far less concequential than a false negative that led to a user being scammed or infected with a virus. This visualization also reflects that, while false positives and negatives do exist in this model, the overall predictive accuracy is quite strong, with accurate predictions significantly outweighing inaccurate ones.

4 Method 2: Random Forest Classifier

4.1 Theoretical Underpinnings

Random Forests are an ensemble learning method that improves upon the decision tree model by constructing an entire "forest" of trees that serves outputs with significantly lower variance and comprable biases, serving as a marked improvement from the single-tree model. Before we can go into too much depth on the random forest, however, we must first understand how an individual decision tree works.

A decision tree (in our case a classification tree) serves the simple prediction of assigning the class to a test observation that belongs to the highest number of the training observations existing in that same region. To construct the tree, the model uses a greedy method called *recursive binary splitting*. This method makes the split at each step of the tree's construction that results in the largest reduction in the error measurement. While with regression this means the maximum drop in RSS, with classification a different measure is needed. One of the most common measures used for this is the Gini index, which is defined by

$$Gini = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}),$$

where $\hat{p}_{mk}$ is the proportion of nodes in region m belonging to class k. A high gini value reflects a "pure" node whose classes are highly uniform, and a low value reflects the opposite, where the node is not very useful for a classification task. The tree model, then, aims to develop rules that will greedily find these highest gini values, explaining the maximum amount of the observations' classifications in each added node. The decision tree model continues to find half-plane splits $R_1(j, s) = \{X|X_j < s\}$ and $R_2(j, s) = \{X|X_j \geq s\}$, where s is the cutoff point defining the split between the two regions, that maximize these gini values until the terminal node contains some pre-determined minimum quantity of observations.

Transitioning from individual trees to forests, the random forest model takes advantage of a technique called bagging to achieve significantly lower variance than an individual tree without causing a significantly higher bias. Bagging takes a single training set and breaks it into many smaller sets. By building separate models on the many training sets that are created from the original set, and then averaging the predictions that these models generate, the bagging technique allows low-bias predictions. In simple terms, a bagging model takes $\hat{f}^1(x), \hat{f}^2(x), ..., \hat{f}^B(x)$ for B sets and averages them for a low-bias prediction, producing the model

$$\hat{f}_{bag}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^b(x),$$

where the B different sets are bootstrapped resamplings from the original training data. This is the method by which we are able to go from a single tree to a random forest. The random forest takes B bootstrapped training sets from the original full training data, fits B decision tree models to these, and averages their outputs for a reduced-variance model.

Parameters to be tuned for a random forest include the maximum depth of the trees (deeper trees lead to lower bias, higher variance), the number of features to be considered in each tree, and the minimum number of observations to be allowed in a terminal node.

4.2 Results on URL Data

We test a random forest classifier using the Scikit-Learn `RandomForestClassifier` module. We test the number of estimators and depth allowed per tree, using a 70/30 train-test split and testing n_ estimators $\mathbf{i} = \begin{pmatrix} 100 & 500 & 1000 \end{pmatrix}$ and max depths $\mathbf{j} = \begin{pmatrix} 100 & 500 & 1000 & None \end{pmatrix}$. While a full cross-validation would have been ideal, providing more reliable results because of the lower variance of the model MSEs due to its averaging of the k models tested on each (i,j) pair, the random forest model was too computationally expensive to complete a full cross validation in a reasonable amount of time on our local machines, leading us to use the validation set approach for this model.

The results of our parameter testing are shown below. While depth 100 proved to be clearly too small, there was no major difference between the other depth limits tested except for with a smaller number of estimators, where assigning "no max depth" was inferior to the other values.

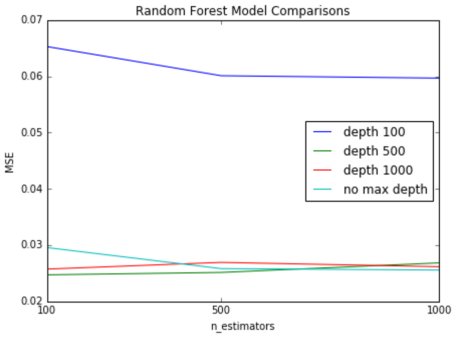


Figure 3: MSEs produced by varying values of max depth and n-estimators

The optimal (max depth, n_ estimators) pair was 1000 estimators and no max depth, producing an MSE of 0.021. This was not, however, very different from some of the other combinations tested, as can be seen in the figure above. The main takeaway from this appears to be that the model was not incredibly sensitive to these parameter values once they surpassed a certain point of roughly 500 estimators and max depth 500.

In the end, this model was 97.8% accurate, with 113 false positives and 195 false negatives, as is shown in the confusion matrix below.

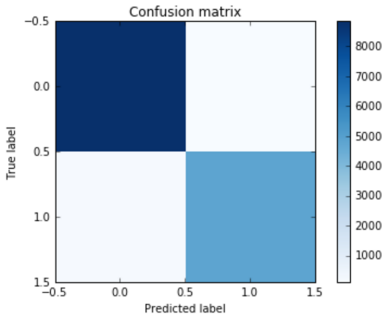


Figure 4: Confusion Matrix for Random Forest Model

5 Method 3: Support Vector Machines (SVM)

5.1 Theoretical Underpinnings

In an intuitive sense, a support vector machine (SVM) creates a set of hyperplanes, which separates each class from the others. For a normal vector $\mathbf{n}$, we call the hyperplane $P = \langle \cdot, \mathbf{n} \rangle = c$ the maximum-margin hyperplane. If data can be perfectly separated by a hyperplane, then an infinite set of these hyperplanes must exist. The best choice of these hyperplanes for classification purposes, however, is the maximum-margin hyperplane, since this allows the most space between the classes, reducing the risk of overfitting the data's decision boundaries. The maximal margin hyperplane is a solution to the optimization problem

$$\underset{\beta_0,\beta_1,...,\beta_p}{maximize}\ M$$

$$subject\ to \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip}) \geq M\ \forall i = 1, ..., n$$

.

In the case of the support vector classifier, however, it is worthwhile to sometimes allow the model to misclassify observations in the name of not overfitting the data. This model, sometimes called the soft margin classifier, does this by allowing observations to be on the "wrong" side of the margin, or even hyperplane. This modification of the previously discussed model is a solution to the optimization problem

$$\underset{\beta_0,\beta_1,...,\beta_p,\epsilon_1,...,\epsilon_n}{maximize}\ M$$

$$subject\ to \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip}) \geq M(1 - \epsilon_i),$$

$$\epsilon_i \geq 0, \sum_{i=1}^{n} \epsilon_i \leq C,$$

where C is a nonnegative tuning parameter, M is the width of the margin that we seek to maximize, and $\epsilon_1, ..., \epsilon_n$ are *slack variables* allowing observations to be on the wrong side of the hyperplane as discussed before. The classifications are then based on the sign of $f(x*) = \beta_0 + \beta_1 x_1^* + ... + \beta_p x_p^*$, with C serving as a budget of sorts for the extent to which the observations can violate the margin by imposing the restriction $\sum_{i=1}^{n} \epsilon_i \leq C$.

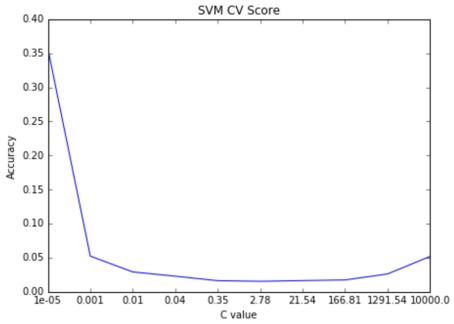
Last, the SVC has the property that its hyperplane is only affected by observations that lie on the margin or on the wrong side of it. These observations are the support vectors that affect the classifier. This property controls the *bias-variance tradeoff* through which the primary parameter tuning with this classifier takes place. As C increases and the constraint placed upon the ϵ values is loosened, increasingly more support vectors exist, due to more values being allowed to violate the hyperplane. As the number of support vectors increases, the number of observations determinging the hyperplane is also increasing, which represents a decreased variance in the model, at the expense of an increased bias due to the lost precision of having less of the observations correctly separated by the hyperplane. The opposite can be said of decreasing C, where the values of ϵ face greater constraints, leading to less values affecting the hyperplane, increased variance as a result of this, and decreased bias due to the more precise classification of the training observations that results.

6 Results on URL Data

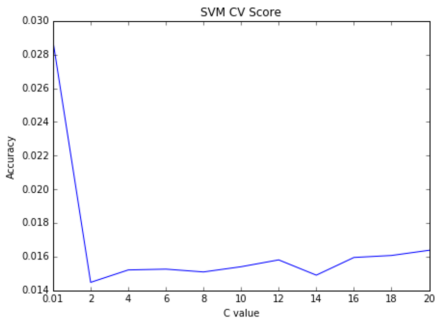
Through trial and error, we arrived at the regularization parameters $\mathbf{c} = \begin{pmatrix} 0.8 & 0.9 & 1 & 1.1 & 1.2 & 1.3 \end{pmatrix}$ as the final set to test during cross validation. We began with sets covering larger ranges, and then narrowed our search down to this smaller range in order to find the optimal c. Then, we cross-validated on five folds, partitioning the data in five pairwise disjoint subsets, using each as the test set once, with the rest as training data, over five iterations, testing each c_i, $i = 1, 2, \ldots, 6$ during each fold of the cross-validation. Then, we computed the cross-validated mean squared error, given by

$$MSE_{c_i} = \frac{1}{5} \sum_{j=1}^{5} MSE_{j,c_i},$$

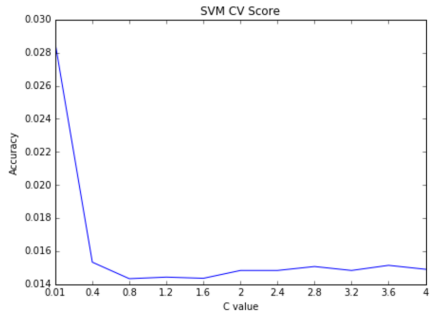
where j is the j-th fold, and c_i is the i-th element of $\mathbf{c}$, for $i = 1, 2, \ldots, 6$, $j = 1, 2, \ldots, 5$. We chose the value of c by picking the c_i corresponding to the lowest MSE_{c_i}, which was 0.9. The narrowing process of arriving at $C = 0.9$ is shown in the series of plots below, comparing MSE to C, where we began with a wide range of C values, and then closed in upon that which worked best.



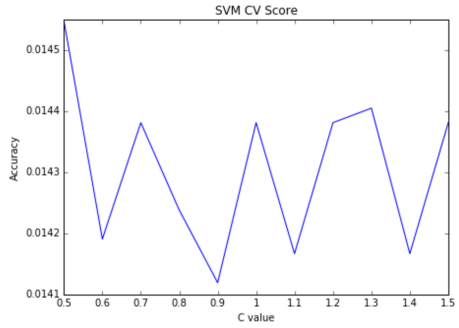
(a) C = (0.01, ..., 10000)



(b) C = (0.01, ..., 20)



(c) C = (0.01, ..., 4)



(d) C = (0.5, ..., 1.5)

Figure 5: SVM MSEs vs C-values

0.9 as the optimal parameter is an unstable optima that is sensitive to slight changes in the way the train and test data are partitioned, which can be inferred due to the roughness of the plot testing $\mathbf{c} = \begin{pmatrix} 0.5, ..., 1.5 \end{pmatrix}$. In our many attempts, however, 0.9 was the most common optima we arrived at.

With the value of C decided upon, we then fit the model to the full training data, predicted on the test data, and compared those predictions to the results observed in reality, obtaining an $MSE \approx 0.0125$ and approximately 98.74% accuracy. A confusion matrix, interpreted the same way as in figure 2, is displayed below.

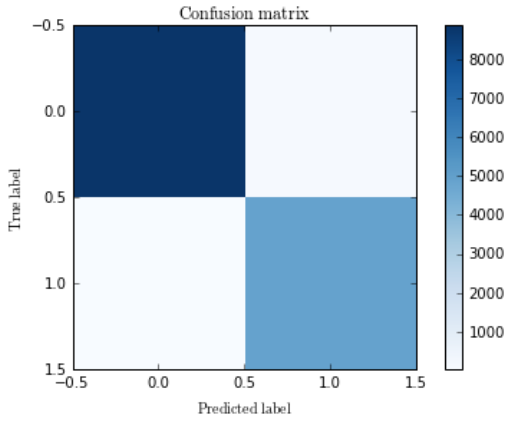


Figure 6: Upper left: true negative, upper right: false positive, lower left: false negative, lower right: true positive

This model's performance was strikingly similar to that of Logistic Regression, with near-identitcal predictive accuracy (98.74% vs 98.77%) and similar patterns of false negatives and false positives (SVM: 46 false positives, 130 false negatives. Logit: 48 false negatives, 123 false positives). We had initially expected this model to outperform Logistic Regression due to its ability to achieve low bias with the regularization parameter while detecting subtle patterns in the data, so this outcome was unexpected.

7 Conclusion

In conclusion, logistic regression and the SVM model performed nearly identically, and the random forest was the clear loser of the three models tested. The two best models showed high predictive accuracy, both achieving results near 99% accuracy. The random forest was not terrible, still achieving accuracy of 97%, but to improve upon this model and have any hope of improving it to the point of rivaling the SVM and logistic models, we we would require additional time and resources to apply a more refined feature search on the model, testing more values of max depth and number of features with k-fold cross validation. All in all, the results of this experiment are an encouraging sign for the future of computer security, indicating that, with sufficient data, many of the phishing and virus schemes of the web can be accurately detected and prevented.