Statistical Natural Language Processing
Brandeis University
Fall 2015

Analysis of YouTube Spam

Jing Zou
Huilin Gang
October 26, 2015

1 Purpose

YouTube becomes more lucrative targets for spammers hoping to attract unsuspecting users to malicious websites, where a variety of threats such as scams (phishing, e-commerce) and malware can be found. We demonstrate how such of these can be tracked over time using prime SVM and Neural Network Model. Based on both models, we will estimate how these model work with our dataset.

2 Dataset Collection

Following the lead of earlier related YouTube research, a data set was collected in order to permit the investigation of contemporary spam comment activity.These data are retrieved from YouTube API which provides access to video and user information. Data retrieval ran from October 31st, 2011 to January 17th, 2012.

Table 2.1: Data Properties

Videos	6407
Total comments	6431471
Total users	2860264
Spam comment users	177542

Table 2.2: Sample data from training dataset

published	videoId	userId	commentText	spam
1321991078000	video1473	user812400	i lost count after last night at the gay bar	false

3 Preprocessing

Our methodology requires the generation of a network to represent the comment posting activity of users to a set of videos. Initially, the focus of the evaluation is English-language spam comments. Therefore we need to specify our dataset in only English. Also, we might try tokenization, removing stop words, finding the inclusion of URLs etc. in preparation.

4 Methodologies

1. SVM
   Our focus is how SVM can have good practical performance scaled to work with large training sets. We are going to implement algorithms use variant of primal stochastic gradient descent(SGD).

   (a) Generalization
   A learning problem mush have a measure that says how good and algorithm

performs on the task. A universal goal is *generalization*. We try to minimize some function over the training set like:

$$\hat{g}(\theta) = \frac{1}{n}\sum_{i=1}^{n} L(x_i, y_i; \theta)$$

where L is a loss function.

(b) Regularization
Regularization refers to augmenting the objective function g(θ) with an extra term that penalizes complex θ vectors. Therefore, the general form becomes

$$\hat{g}(\theta) = \frac{1}{n}\sum_{i=1}^{n} L(x_i, y_i; \theta) + r(\theta)$$

where r(θ) is a regularization term.

(c) Stochastic Gradient Descent
Gradient method computes $\theta = \text{argmin}\ \hat{g}(\theta)$, where $\hat{g}(\theta)$ is the sum of empirical loss and regularization terms. Stochastic Gradient Descent(SGD) is a simple yet very efficient approach to discriminative learning of linear classifiers under convex loss functions such as Support Vector Machines. It has been successfully applied to large-scale and sparse machine learning problems often encountered in text classification. We do not compute the gradient using the entire training set, however with respect to only a single randomly chosen example. The update rule is then:

$$\theta_{t+1} = \theta_t - \eta_t \nabla L(x_{i(t)}, y_{i(t)}; \theta_t) - \eta_t \nabla_r(\theta_t)$$

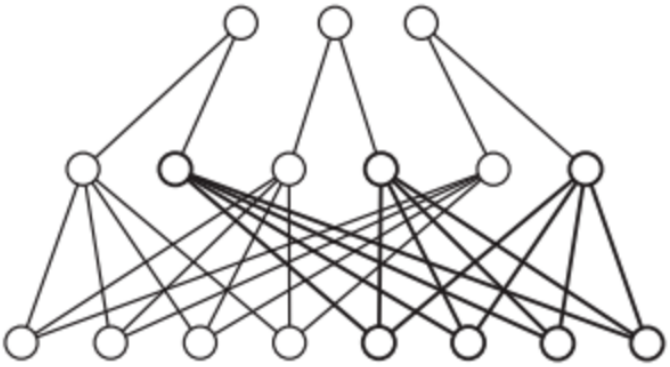
where $i(t)$ is an index drawn randomly from {1, 2, ..., n}. In expectation, this update is the same as gradient descent, because $E_{i(t)}[l(x_{i(t)}, y_{i(t)}; \theta_t)] = \frac{1}{n}\sum_i l(x_i, y_i; \theta)$

(d) The primal SVM formulation
There are two ways of treating the SVM problem. The classical method is the hard margin SVM[Vapnik and Lerner, 1963], which assumes that the dataset is linearly separable: hence every point must be correctly classified by the maximum margin hyperplane. The soft margin SVM [Bennett and Mangasarian, 1992, Cortes and Vapnik, 1995] allows for some points to be misclassified, but penalizes these points appropriately. We implemented the latter represented with the following optimization problem.
In contrast to (batch) gradient descent, SGD approximates the true gradient of $E(w, b)$ by considering a single training example at a time.
By implementing a first-order SGD learning routine, the algorithm iterates over the training examples and for each example updates the model parameters according to the update rule given by

$$\theta \leftarrow \theta - \eta(\alpha\frac{\partial r(\theta)}{\partial \theta} + \frac{\partial L(\theta^T x_i + b, y_i)}{\partial \theta})$$

where η is the learning rate which controls the step-size in the parameter space. The intercept b is updated similarly but without regularization.
The learning rate η can be either constant or gradually decaying. We give the default learning rate to be

$$\eta^{(t)} = \frac{1}{\alpha(t_0 + t)}$$



2. Neural Network
   For example, a neural network structure for 3 classes, 8 features and 2 feature groups per class:

Let n be the number of output units, g the number of feature groups per class and gn the number of hidden units given by the product. The output of the network for an input x is:

$$f(x) = W_o tanh(W_h x + bh) + b_o$$

where W_o is a $n \times gn$ matrix, W_h is a $gn \times |V|$ matrix, b_o is a vector of dimension n and bn is a vector of dimension gc.
Let $w_1, ..., w_n$ be a set of weight vectors derived from a linear model. Let $\pi_1, ..., \pi_g$ be a random partition of the set of feature indices $1, 2, ..., |V|$ into g equally-sized groups. For each class c, associate g hidden units $h_1^{(c)}, ..., h_g^{(c)}$. For all $i \in [1, g]$ and $j \in [1, |\pi(i)|]$, the weight of the connection between $h_1^{(c)}$ and the input unit $x_{\pi_j}^{(i)}$ is given by $(W_c)_{\pi_j^i}$