COSC4010/5010: Final Project Report

Trystan Bennett
Department of Computer Science
University of Wyoming
Laramie, Wyoming 82070
Email: tbenne10@uwyo.edu

***Abstract*—In this paper, we apply machine learning approaches to automatically detect Phishing comments on video sharing platforms. We first evaluate and compare the detection accuracy of two classic supervised learning methods. We then evaluate these methods with two different methods of feature extraction, and tune hyper-parameters to find optimal accuracy and f1 scoring. The results present various methods of optimizing the hyper-parameters, and we then give an evaluation for how the results perform optimally specifically to phishing comments in video sharing platforms.**

I. Introduction

Online video sharing platforms, such as Youtube, can often be clouded with comments that contain spam. Much of this spam is an attempt to lure standard users to fall for phishing scams. A phishing scam is an attempt by a malicious user to obtain sensitive information, such as passwords and financial info, by posting false links or convincing a victim of a separate motive. This poses a cybersecurity risk to users on social media platforms, and may hurt a company's competitive advantage if frequent spam drives users to seek out alternative platforms that do not contain as much spam. Since many phishing scams today are automated, sorting through each individual case would be very tedious. Thus, a corporation may seek to implement machine learning methods that are able to detect spam with a high accuracy. In this report, an analysis of two feature extraction techniques -count vectorization and term frequency-inverse document frequency- will be performed on a dataset of Youtube comments from popular videos provided by the University of California in Irvine. The report seeks to evaluate the tuning of various hyper-parameters in order to provide accurate results that are specific to comments on video sharing platforms.

II. Methodology

A. Notation and Problem Setting

We consider the supervised learning problem, with a training sample of size (.7). we use "f" to denote a model, x to denote an example and y its label. Samples are denoted as "S = (x_i, y_i)", where x_i is example i and y_i is its label.

B. Multinomial Naive-Bayes

MultinomialNB implements the naive Bayes algorithm for multinomially distributed data, and is one of the two classic naive Bayes variants used in text classification (where the data are typically represented as word vector counts, although tf-idf vectors are also known to work well in practice). The distribution is parametrized by vectors $\theta_y = (\theta_{y1}, \ldots, \theta_{yn})$ for each class y, where n is the number of features (in text classification, the size of the vocabulary) and θ_{yi} is the probability $P(x_i \mid y)$ of feature i appearing in a sample belonging to class y.

The parameters θ_y is estimated by a smoothed version of maximum likelihood, i.e. relative frequency counting:

$\hat{\theta}_{yi} = \frac{N_{yi}+\alpha}{N_y+\alpha n}$

where $N_{yi} = \sum_{x \in T} x_i$ is the number of times feature i appears in a sample of class y in the training set T, and

$N_y = \sum_{i=1}^{|T|} N_{yi}$ is the total count of all features for class y. The smoothing priors $\alpha \geq 0$ accounts for features not present in the learning samples and prevents zero probabilities in further computations. Setting $\alpha = 1$ is called Laplace smoothing, while $\alpha < 1$ is called Lidstone smoothing.[1]

C. Linear SVM

Linear Support Vector Classification.

Similar to SVC with parameter kernel=linear, but implemented in terms of liblinear rather than libsvm, so it has more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples.

This class supports both dense and sparse input and the multiclass support is handled according to a one-vs-the-rest scheme.[2]

III. Feature Extraction Methods

A. Count Vectorization

Implements both tokenization and occurrence counting in a single class. [3] This method converts each line of text into a matrix of vectors containing binary values.

B. TF-IDF Vectorization

Term frequencyinverse document frequency vectorization- In a large text corpus, some words will be very present (e.g. the, a, is in English) hence carrying very little meaningful information about the actual contents of the document. If we were to feed the direct count data directly to a classifier those very frequent terms would shadow the frequencies of rarer yet more interesting terms. Tf means term-frequency while tfidf means term-frequency times inverse document-frequency: tf-idf(t,d) = tf(t,d) * idf(t)[3]

Since this method applies to English comments on social media that often contain common words, this method is therefore more appropriate than Count Vectorization, and will be adjusted with tuned parameters.

C. TF-IDF parameter tuning

A total of five parameters will be tuned in the use of TF-IDF vectorization[4]:

smooth idf: Smooth idf weights by adding one to document frequencies, as if an extra document was seen containing every term in the collection exactly once. Prevents zero divisions.

max idf:When building the vocabulary ignore terms that have a document frequency strictly higher than the given threshold (corpus-specific stop words). If float, the parameter represents a proportion of documents, integer absolute counts. This parameter is ignored if vocabulary is not None.

sublinear tf: Apply sublinear tf scaling, i.e. replace tf with 1 + log(tf).

lowercase: Convert all characters to lowercase before tokenizing.

stop words = 'english': If a string, it is passed to checkStopList and the appropriate stop list is returned. english is currently the only supported string value. If a list, that list is assumed to contain stop words, all of which will be removed from the resulting tokens.

IV. Precision Testing

A. Confusion Matrix

Evaluates the accuracy of a classification

By definition a confusion matrix C is such that $C_{i,j}$ is equal to the number of observations known to be in group i but predicted to be in group j.

Thus in binary classification, the count of true negatives is $C_{0,0}$, false negatives is $C_{1,0}$, true positives is $C_{1,1}$ and false positives is $C_{0,1}$.[5] In pattern recognition, information retrieval and binary classification, precision (also called positive predictive value) is the fraction of relevant instances among the retrieved instances, while recall (also known as sensitivity) is the fraction of relevant instances that have been retrieved over the total amount of relevant instances. Both precision and recall are therefore based on an understanding and measure of relevance.[6] We thus obtain the following equations:

Prediction = $C[1][1]/(C[0][1] + C[1][1])$

Recall = $C[1][1]/(C[1][0] + C[1][1])$

False Alarm Rate = $C[0][1]/(C[0][1] + C[1][1])$

B. F1 score

V. Experiment

A. Dataset Description

The dataset used is the Youtube Spam Collection provided by the University of California Irvine. It is a balanced dataset created using comments from five separate popular videos: PSY-Gangnam Style, Katy Perry-Roar, LMFAO-Party Rock, Shakira-Waka Waka, and Eminem-Love the way You Lie. There are five features and, in total, 1956 instances. Each video's dataset is provided in individual files, but are concatenated for the purpose of this report. The features are as follows: Comment ID,Author,Date,Content,and Tag. However, machine learning is performed as a binary-class instance on the Content and Tag features. The content is a string of text that must be converted to numerical values using feature extraction. The tag is a binary value representing whether or not the data is classified as spam or non-spam. The dataset can be accessed with the following link: http://dcomp.sor.ufscar.br/talmeida/youtubespamcollection/.

TABLE I
Performance of Naive-Bayes

Feature Extraction	Prediction Accuracy
Count Vectorization	0.811583577713
Unattuned TFIDF	0.82991202346
TFIDF- Max df = .1	0.83137829912
TFIDF- Sublinear df = true	0.848973607038
TFIDF- Stop words enabled	0.846041055718
TFIDF- Lowercase = false	0.856304985337
TFIDF- Smooth df = false	0.83357771261
TFIDF- Optimized	0.887096774194

TABLE II
Performance of Linear SVC

Feature Extraction	Prediction Accuracy
Count Vectorization	0.941348973607
Unattuned TFIDF	0.922287390029
TFIDF- Max df = .3	0.925953079179
TFIDF- Sublinear df = true	0.923020527859
TFIDF- Stop words enabled	0.91715542522
TFIDF- Lowercase = false	0.947214076246
TFIDF- Smooth df = false	0.923753665689
TFIDF- Optimized	0.953079178886

TABLE III
F1 scores for Naive-Bayes

Feature Extraction	F1 Score
Count Vectorization	0.842818428184
Unattuned TFIDF	0.842818428184
TFIDF Optimized	0.887983706721

B. A Coarse Performance Evaluation

In this section, we evaluate performance of two feature extraction methods using random (or default, or optimal) hyper-parameter settings. We randomly select 70% examples in the data set for training, and use the rest 30% for testing.

First, we apply two feature extraction techniques to NaiVe-Bayes to learn prediction models from the training sample, and evaluate prediction accuracies of these models on the testing sample. Results are presented in Table V-B.

Second, we apply two feature extraction techniques to Linear SVC to learn prediction models from the training sample, and evaluate prediction accuracies of these models on the testing sample. Results are presented in Table V-B.

In this section, we evaluate the f1 score on the Count Vectorization, Unattuned TFIDF, and Optimized TFIDF for each of the prediction models. Table V-B.

Table V-B.

TABLE IV
F1 SCORES FOR LINEAR SVC

Feature Extraction	F1 Score
Count Vectorization	0.95530726257
Unattuned TFIDF	0.925561797753
TFIDF Optimized	0.925561797753

TABLE V
CONFUSION SCORES OF NAIVE-BAYES

Fea. Extraction	Prediction	Recall	False alarm
Count Vectorization	0.809895833333	0.878531073446	0.190104166667
Unattuned TFIDF	0.809895833333	0.878531073446	0.190104166667
TFIDF Optimized	0.854901960784	0.923728813559	0.145098039216

TABLE VI
CONFUSION SCORES OF LINEAR SVC

Fea. Extraction	Prediction	Recall	False alarm
Count Vectorization	0.920391061453	0.930790960452	0.0796089385475
Unattuned TFIDF	0.920391061453	0.930790960452	0.0796089385475
TFIDF Optimized	0.944751381215	0.966101694915	0.0552486187845

Finally, a confusion matrix evaluation is performed on each prediction model to evaluate the precision, recall, and false alarm rate of three methods of feature extraction.

VI. CONCLUSION

Accuracy obtained from Tfidf was more accurate than Count Vectorization as predicted. However, in Linear SVC classification, Count Vectorization outperformed the Unattuned Tfidf. This is not an error in prediction as they share the same results in prediction scoring methods. This is likely due to a difference in the default parameters, such as the stop words being enabled. In the Linear SVC classification, we find that enabling stop words shifts the prediction accuracy down, which is opposite in the Naive-Bayes classification. Function words, such as pronouns, are common and thus down-weighted in Tfidf, but are given equal weight to less frequent words in Count Vectorization. Therefore, one possibility is that the high accuracy of Tfidf results in a potential counterproductive shift in tokenization results. In addition, it was found that setting Lowercase to false yielded much better results in both Naive-Bayes and Linear SVC. In viewing the dataset, many comments classified as phishing would use strings of characters in all-caps as an attempt to grab the attention of viewers. Thus, it is important to tokenize these strings without setting them to lowercase, as the results show that words in all-caps are more likely to be classified as a phishing scam.

.

VII. REFERENCES

1.$http://scikit-learn.org/stable/modules/naive_bayes.html$
2.$http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html$
3.$http://scikit-learn.org/stable/modules/feature_extraction.html$
4. $http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.htmlsklearn.feature_e$
5. $http://scikit-learn.org/stable/modules/generated/sklearn.met$
6. $https://en.wikipedia.org/wiki/Precision_and_recall$
Dataset: $http://dcomp.sor.ufscar.br/talmeida/youtubespamcolle$