File-type Identification with Incomplete Information

Siddharth Gopal
Carnegie Mellon University
sgopal1@andrew.cmu.edu

Yiming Yang
Carnegie Mellon University
yiming@cs.cmu.edu

Konstantin Salomatin
Carnegie Mellon University
ksalomat@cs.cmu.edu

Jaime Carbonell
Carnegie Mellon University
jgc@cs.cmu.edu

ABSTRACT

File-type Identification (FTI) is an important problem in digital forensics, intrusion detection, and other related fields. Using state-of-the-art classification techniques to solve FTI problems has begun to receive research attention; however, general conclusions have not been reached due to the lack of thorough evaluations for method comparison. This paper presents a systematic investigation of the problem, algorithmic solutions and an evaluation methodology. Our focus is on performance comparison of statistical classifiers (e.g., SVM and kNN) and knowledge-based approaches, especially COTS (Commercial Off-The-Shelf) solutions which currently dominate FTI applications. We analyze the robustness of different methods in handling damaged files and file segments. We propose two alternative criteria in measuring performance: 1) treating file-name extensions as the true labels, and 2) treating the predictions by knowledge based approaches on intact files; these rely on signature bytes as the true labels (and removing these signature bytes before testing each method). In our experiments with simulated damages in files, SVM and kNN substantially outperform all the COTS solutions we tested, improving classification accuracy very substantially – some COTS methods cannot identify damaged files at all. Our experiments also show the scalability of SVM and kNN to large applications after adequate feature selection.

Categories and Subject Descriptors

E.5 [**Data**]: Files – *Recovery/Backup.* I.5.2 [**Pattern Recognition**]: Design Methodology – *Classifier Design and Evaluation.* H.1.0 [**Information Systems**]: General

General Terms

Algorithms, Experimentation, Performance.

Keywords

Digital Forensics, File-type Identification, Classification, Scalability, Comparative Evaluation

1. INTRODUCTION

File-type Identification (FTI) is the task of assigning a pre-defined label (the file type) to each instance (each file) based on observed data in the file. The conventional application of FTI is in operating systems where computers need to choose different programs to process the information based on the type of each file. Algorithmic solutions are needed for automated identification because systems cannot always rely on human-assigned extension in file names; users occasionally choose a wrong extension when creating a file name, or simply forget to specify it. A variety of Commercial Off-The-Shelf (COTS) software has been developed for automated FTI. For example, Libmagic [8] is open-source software in Linux for FTI (the 'file' command). Other popular COTS software includes TrID [22], Outside-In [21], DROID [20], and so on.

In the past decade, FTI has become an increasingly important area in digital forensics research where the focus is on extracting and analyzing useful information from digital devices such as mobile phones, computer hard disks and CD-ROMs. Forensic practitioners often encounter broken CD-ROMs, damaged hard-disks, or partially deleted files. They are frustrated with the limitations of COTS solutions whose predictions are essentially based on the detection of *signature bytes* in each file, and the detection relies on a manually created database of mappings (rules) from signature bytes to file-types. For example, a Microsoft Windows bitmap file is typically matched with the signature string 'BM'; a JPEG file is matched with the two-byte signature '0xFF, 0xD8'. If the signature bytes or the allocation information of the file segments are missing or garbled, COTS solutions will work poorly if at all (see Section 5 for empirical evidence).

Other application areas where automated FTI has become important include intrusion detection [9] , virus removal, firewall protection [30], etc. For example, in intrusion detection, individual packets are monitored; if any offending file-type of data is detected, those data will be filtered out. In another example, firewalls are often setup to detect executable files from unknown sources; if such files are detected, they will be blocked. In such scenarios, the location of signature bytes and the allocation information about file segments are often not available. COTS solutions or similar knowledge-engineering approaches to FTI would perform poorly.

Several statistical classification methods have been studied to address the limitations of COTS solutions or knowledge-engineering based approaches. Those methods treat each file type as a category (class), and use supervised learning techniques to predict the category label for each test instance (a file) based on its content and a training set of labeled instances. Such approaches are referred to as *content-based*, in distinction from

those relying on file-name extensions or file-header information alone. Each file is represented using a vector of feature weights where the features are typically n-gram bytes, and the weights are typically the within-file frequency of the features [18] or some kind of TF-IDF (term frequency multiplied to Inverted Document Frequency) weight (see Section 2). Just like in text categorization where word order is typically ignored by statistical classifiers, the order of n-gram bytes is also often ignored by the classifiers in FTI. Of course by tuning the value of n for n-gram features, local context can be partially captured. Once we have files represented as feature vectors, any statistical classification method can in principle be applied. Approaches examined so far include centroid-based methods [12],[15],[16], [17], [18], [19] where each category is represented using the centroid of its member instances in the training set, and the category centroids are compared to each test instance for inference. Other methods include 1-Nearest Neighbor (1-NN) [17], k-Nearest Neighbor (kNN) [1],[2], 3-layer neural networks (with PCA-induced features) [4] Support Vector Machines (SVM), etc. [1],[2].

Although good progress has been made in statistical approaches to FTI, general conclusions are difficult to obtain with respect to the strengths and weaknesses of different methods, and it is not clear which ones are representative for the current state of the art. The reasons are:

- **The lack of evaluation results on shared benchmark datasets**: All the published results so far were obtained on unshared datasets, making it impossible to directly compare methods across studies or to replicate published results. A realistic data collection, called *Realistic Data Corpus* (*RealisticDC*) [10], has been recently made publicly available; however, no evaluation result of any method has yet been reported on that collection.
- **The lack of well-established evaluation methodology**: To our knowledge, no evaluation result was published for performance comparison against and among COTS solutions. Although COTS predictions are all based on signature bytes which can be found in a manually created external data base, different software may produce different labels for the same file type, or they may divide file types into sub-types inconsistently[1]. Comparing COTS solutions has been difficult due to the lack of standardization of file-type (category) labels. On the other hand, the evaluations of statistical classifiers for FTI often use file-name extensions as the true labels, which is contradictory to the common belief that user-assigned extensions in file names are highly unreliable [12]. This contradiction makes it difficult to interpret the reported evaluation results for statistical classifiers in FTI.
- **No cross-method comparative evaluation has been reported on damaged files**. This is the most crucial issue for the security and forensic applications mentioned above. The claimed advantage of statistical classification approaches over COTS or knowledge-based solutions has not been empirically examined using any quantitative measure. As a result, software developers and forensic examiners cannot tell which tools would be best for their problems, and researchers in FTI-related fields also face difficulties in reaching conclusions regarding the state of the art.

This paper addresses the above key issues by conducting a thorough investigation with several representative statistical classifiers and COTS solutions, as follows:

a) We report the first comparative evaluation using controlled experiments with statistical classification methods (Support Vector Machines and k-Nearest Neighbor classifiers) and popular COTS solutions (Libmagic, TrID, Outside-In and DROID) on a shared and publicly available *ReasliticDC* dataset.

b) We propose two strategies for cross-method evaluation. The first is to use the labels assigned by a COTS solution (e.g., Libmagic) on the intact files as the true labels of test instances, and to measure the *accuracy* of statistical classifiers in predicting file types accordingly. The second is to use file name extensions as the true labels, and to measure the *consistency* in label assignment by each COTS solution accordingly. The former (accuracy) allows us to compare statistical classifiers conditioned on the choice of software for information extraction (as the next step after file type identification). The latter (consistence) allows us to compare different COTS solutions without subjective unification of software-specific labels.

c) We use the Realistic Data Corpus (RealisticDC) as the test bed, which is recently made publicly available by Garfinkel et al [10] for digital forensics research, and we provide the first set of empirical results on this corpus. By making our detailed documentation and data preparation toolkit together accessible, we ensure that future results on this dataset can be compared with ours.

d) Our experiments focus on performance analysis of different methods over incomplete files (using files with simulated damages and file segments) as well as complete files; the latter has been the setting in all previous evaluations. Incomplete files are particularly prevalent in forensics. We found SVM and kNN outperforming Libmagic (among the best of COTS solutions) by a factor of 10 in micro-averaged F_1, and by a factor of 7.3 to 8.0 in macro-averaged F_1 (Sections 3.2 and 5) .

e) Our experiments also show that with adequate choice of n in n-gram feature generation and statistical feature selection, SVM (and kNN) can scale very well to large applications without any (significant) sacrifice in accuracy.

The rest of the paper is organized as follows. Section 2 outlines our statistical learning framework for classification and the feature generation process. Section 3 discusses our evaluation methodology. Section 4 describes the experiments and data. Section 5 reports our results. Section 6 addresses scalability issue with statistical feature selection, and analyzes the effectiveness-

[1] For example, a 'C++ program' can be considered to belong to the following types – *C++ program text*, *Program Source code*, *Text*, in increasing order of generality. The desirable level of generality is a subjective choice.

efficiency trade-off. Section 7 concludes by summarizing our findings.

2. THE STATISTICAL APPROACH

In order to apply statistical classification methods to FTI, we need a set of features to represent files and to discriminate different types from each other. N-gram bytes have been found highly useful for FTI in previous work [1][15][19] hence we follow the same choice of features. Given a collection of files, the feature space is defined as the union of all the unique n-gram bytes in the files. Each file is represented as a vector of feature weights. Within-file frequency of n-gram bytes is a common choice of feature weighting scheme. It is analogous to the *term frequency* (TF) in document retrieval and text categorization; hence we call it TF weight for convenience. Other popular term-weighting schemes are also possible, such as TF-IDF weights where IDF stands for the Inverted Document Frequency of a term in a collection of documents. Applied to FTI, a "document" means a file, and a "term" means an n-gram byte.

Notice that the value of "n" need to be carefully chosen for both classification accuracy and for classifier training and run-time efficiency. Generally, the larger the value of 'n', the more byte order information is captured by the features. That is, the features could be more discriminative for classification. However, a higher value of 'n' also means a larger size of the feature space (growing exponentially in n), which will cause an increased time to train the model and a risk of overfitting the training data. Adequate choice of *n* can found empirically through cross-validation, i.e., using some held-out data (not a part of the test set) to tune the value of *n* and then fix the value in the testing phase.

Having the vector representation of files and discriminative features, any classification method could be in principle applied. We use two of the most popular methods in this study: Support Vector Machines (SVM) and k-Nearest Neighbors (kNN). Both methods have been highly successful in a broad range of classification applications [14][25][5][27]. SVM is formulated as a large-margin method for a geometric classification problem: the objective is to find the decision surface that best separates two classes of data points (vectors) with the maximal margin in between. SVM has been found robust in high-dimensional feature spaces and with skewed class distributions where many classes have a relative small number of labeled instances for training. kNN is radically different: it is typical among instance-based ("lazy") learning methods. It finds the nearest neighbors for each test instance in the training set on the fly, and makes inference based on the class labels in the local neighborhood. Specifically, our kNN uses the cosine similarity as the metric to select the top-k training instances for each test instance, and to weigh the class label of each nearest neighbor; the weights of labels are summed over for each class, and the class receives the highest score is assigned to the test instance. This kind of kNN is called *multi-class kNN* [5],[27], meaning that the unique class labels in each local neighborhood may be more than two. Multi-class kNN typically outperforms two-class kNN in multi-class or multi-label classification problems; the latter converts multi-class labels of training instances into binary labels for each class before training a two-way classifier for the class. SVM as an eager learner is computationally intensive in its training phase, whereas kNN is computationally intensive in on-line testing phase. This dichotomy allows us to investigate the scalability of both types of classifiers.

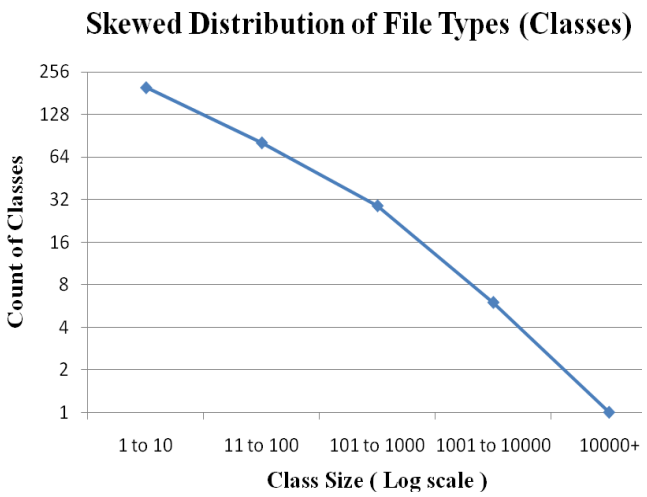

Figure 1: File-types in the RealisticDC dataset have a skewed distribution

3. EVALUATION METHODOLOGY

3.1 Alternative settings for the ground truth

It has been more difficult to obtain the true labels of files for FTI evaluations, compared to some other domains such as text categorization or image pattern recognition where human-assigned labels to documents or objects can be directly used as the true labels for evaluation. In FTI, extensions in file names are potentially incorrect or even missing -- that is why COTS solutions have been developed for automated FTI. This leads to two open questions regarding the evaluation methodology in FTI:

1) How can we get the true labels for evaluation, especially for comparing different statistical classifiers in FTI?
2) File-name extensions are imperfect, but are they still useful for cross-method comparison, especially among different COTS solutions and between COTS and statistical classifiers?

Our answer for the first question is to use the output of a COTS solution on intact files as the true labels, and to compare the performance of different statistical classifiers on damaged or fragmentary files based on those true labels. By using an application-specific choice of COTS solution to produce the true labels, we avoid the need for manual and subjective unification of inconsistent labels from different COTS solutions for the same file. For example, given an excel file (possibly incomplete or damaged) as the input, some COTS solution would label it as "Microsoft Excel 2000" and others would label it as "Microsoft Office Document" or "Microsoft Excel File". These labels follow different naming conventions, and/or provide different levels of detail about the file type. We cannot subjectively decide that one convention is better than the others, or a certain level of detail is most appropriate in general. What level of detail is appropriate depends on the next-step application, e.g., on the choice of program to be used for information extraction or execution after

FTI. Hence, if the output labels of a COTS solution are suitable for the next-step application, it is sensible to use those labels as the ground truth for evaluating statistical classifiers in file-type identification.

Our answer for the second question is yes. We believe that using file extensions as the true labels to evaluate COTS solutions is informative. It is reasonable to assume that file extensions are more often to be correct than incorrect. If the predicted labels by one method agree with file extensions in a large test set more often than another method does, then the chance for the former method to outperform the latter method is higher. Using noisy labels to evaluate the relative performance of FTI methods to each other is still informative, as long as the test set is sufficiently large for statistical significance.

3.2 Metrics

We choose to use micro-averaged F_1 and macro-averaged F_1 as the primary metrics. Both are standard and common in benchmark evaluations [26][27][28] for text categorization, information filtering, information extraction, and so on. Let $c \in C$ be a class, N_c be the number of test instances in the class, and TP_c, FP_c, TN_c and FN_c be the counts of the true positives, false positives, true negatives and false negatives among the system-made predictions with respect to class *c*, respectively. The performance metrics are defined as:

Local (per category) precision $P_c = \frac{TP_c}{TP_c + FP_c}$;

Local (per category) recall $R_c = \frac{TP_c}{TP_c + FN_c}$;

Local (per category) $F_{1,c} = \frac{2P_cR_c}{P_c + R_c}$;

Global precision $P = \frac{\sum_{c \in C} TP_c}{\sum_{c \in C} TP_c + \sum_{c \in C} FP_c}$;

Global recall $R = \frac{\sum_{c \in C} TP_c}{\sum_{c \in C} TP_c + \sum_{c \in C} FN_c}$;

Micro-averaged $F_1 = \frac{2PR}{P + R}$;

Macro-averaged $F_1 = \frac{\sum_{c \in C} F_{1,c}}{|C|}$.

Micro-averaged F_1 and macro-averaged F_1 provide complementary insights into performance analysis. If the classes have a skewed distribution, which is quite common in practical applications, the former is dominated by the system performance on large categories and the latter is dominated by the average performance on small categories.

4. EXPERIMENTS

4.1 Data

The *RealisticDC* dataset was introduced by Garfinkel et al [10] to alleviate the problem of lack of a standardized dataset for FTC research. The dataset was created under realistic situations that mimic the kind of data commonly encountered by forensics investigators. An experimenter was hired to play the role of a normal computer user, exchanging messages, browsing the web, performing office related work, reading news etc. The images of the experimenter's computer disk then were processed and made available as the dataset. By hiring individuals to mimic realistic users instead of directly collecting data from true users, privacy issues were avoided, making the data sharable to the research community.

After performing our own filtering, such as removing empty files and files without extensions, we obtained a total of 31,644 files and 316 unique file-type extensions, among which 213 are binary file-types and 103 are ASCII text file-types. The category distribution is shown in Figure 1. This filtered dataset has the size of 7.2 GB in total. Further details of the filtering process can be found at http://nyc.lti.cs.cmu.edu/clair/datasets.htm.

4.2 Methods for Comparison

For cross-method comparison we include both state-of-the-art classifiers and popular COTS solutions. We list these methods with a corresponding brief description.

SVM is a state-of-the-art classification method we described in Section 2. Specifically, we used the large-scale linear SVM implementation by Hsieh et al [13] in our experiments.

kNN is another state-of-the-art classification method we described in Section 2. We used our own implementation of kNN [27] in the experiments.

Libmagic [8] is one of the most popular COTS solutions for FTI, which has been implemented as a UNIX command line tool. It uses the information about the UNIX/Linux system to recognize certain file types (such as device files) as the first step; if the first attempt fails, then it analyzes the signature bytes of the input file to identify the file-type as the second step; if the second attempt also fails, then the ASCII content within the file is used to identify the file-type. If all the above attempts fail, the file-type will be labeled as not recognized.

TrID [22] is another popular COTS solution designed for identifying file-types from their signature bytes. TrID uses a database of signature patterns. Currently TrID supports the identification of 4093 different file-types.

Outside-In [21] is a part of the suite of algorithms distributed by Oracle for dealing with unstructured files. It uses a proprietary algorithm to identify the file-types without entirely relying on the file-extensions. It can identify more than 500 file-types.

DROID (Digital Record Object Identification) [20] is an open-source file-type identification tool developed by the *National Archives*. Rather than relying on signature bytes only, DROID uses regular expressions to allow flexible match in signature-based file-type identification.

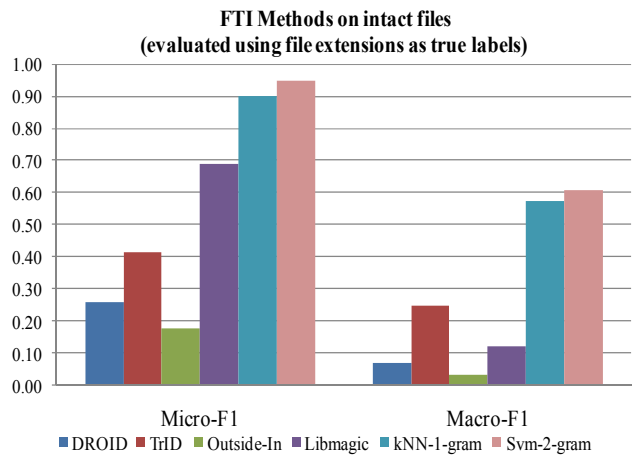

Figure 2: Performance of FTI methods on intact files: File extensions were used as the true labels in the evaluations

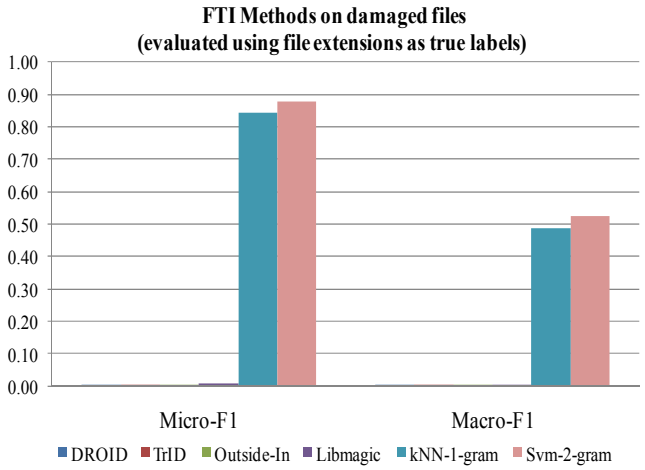

Figure 3: Performance of FTI methods on files with type-1 damages (missing signature bytes): File extensions were used as the true labels in the evaluation.

For comparing the methods on intact files, we used file-name extensions as the true labels of the test instances. For comparing the methods on damaged files or segments of files, we used the output of Libmagic on the undamaged and un-segmented version of the files as the true labels of the test instances. For these experiments, we used the subset of the dataset (30,254 files) on which libmagic was able to predict the file-types. We also investigated other signature-based COTS methods on intact files as the gold-standard, but we omit these variations for brevity, since they provide the same basic insight.

4.3 Simulated Damages and Segments

We simulate file damage in our experiments as follows:

Type-0 corresponds to the case where there is no damage. It reflects an ideal situation where the files are intact without any missing bytes. Also, complete information about file segment allocation is available so that we can treat each file as a contiguous string of bytes after preprocessing.

Type-1 corresponds to the case where the signature bytes in the file are missing. Generally a hard disk is arranged in the form of blocks (clusters) where each block is a contiguous sequence of 512 bytes, and each file is stored across different blocks. The signature bytes of a file are typically stored in the first block assigned to the file. Thus, if the first block is damaged, the signature bytes of the file are lost. In order the mimic such a situation, we removed the first block from each file, that is, the first 512 bytes of the file.

Type-2 corresponds to the case where additional bytes (after the removal of signature bytes) are missing at random locations, i.e., the missing bytes are randomly allocated. We conducted experiments with the random removal of bytes at 10%, 20%, …, 90% of each file in the test set.

Type3 corresponds to the case where files are stored as isolated segments instead of a contiguous segment. In order to mimic such a scenario, we divided the files into shorter segments of specific sizes and conducted experiments using the segments for training as well as testing. Sometimes, in practice it might not be easy to know the distribution of segments nor their labels, so it would be difficult to generate a labeled training dataset. In such cases, the alternative strategy would be use systems which are trained on complete (un-segmented) files. In our experiments for evaluating performance on file segment classification, we perform both the types of training, i.e., training on segments and training on complete files, respectively.

4.4 Detailed Experimental Setting

Our results for SVM and kNN are obtained through a five-fold cross validation process. We divided the full data into five subsets: four out of the five subsets were used for training and validation (parameter tuning), and the remaining subset was used for testing. We repeated this process five times, with a different non-overlapping subset for testing each time; the results were averaged over the five subsets. In SVM we tuned the regularization parameter and in kNN we tuned the number (k) of the nearest neighbors. We tried 5 different values for the SVM regularization parameter, from .01 to 100; and, we tried 10 different values for k in kNN, from 1 to 50. As for feature weighting in both SVM and kNN, we used a conventional TF-IDF weighting scheme named 'ltc' in information retrieval and text categorization [28]. We also varied the value of n in the generation of n-gram features, with n = 1, 2 and 3. COTS methods have neither a training phase, nor any parameter tuning, since they are not based on statistical learning.

5. RESULTS

Figure 2 shows the performance of all the methods on intact files, including both COTS solutions and the statistical classifiers on complete undamaged files. File extensions were used as the true labels. During validation, we found kNN with 1-gram features worked better than kNN with 2-gram features, and SVM with 2-gram features worked better than SVM with 1-gram in terms of classification performance, thus we included the better versions of kNN and SVM in the graph. In micro-averaged F_1, Libmagic is the best method among the COTS solutions; however, in macro-averaged F_1, TrID is the best among COTS solutions. In both

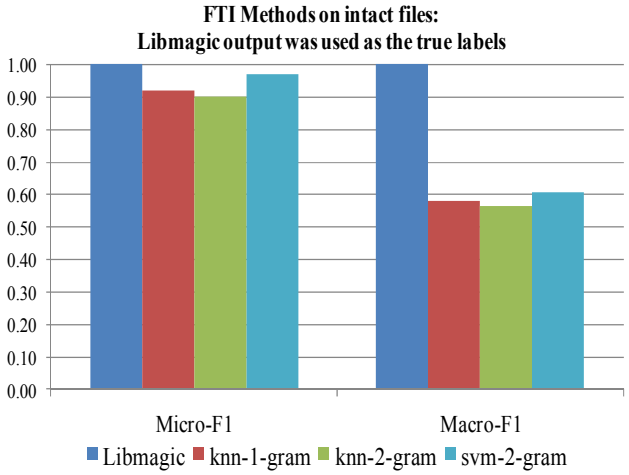

Figure 4: Performance of FTI methods on intact files: The output of Libmagic was used as the true labels.

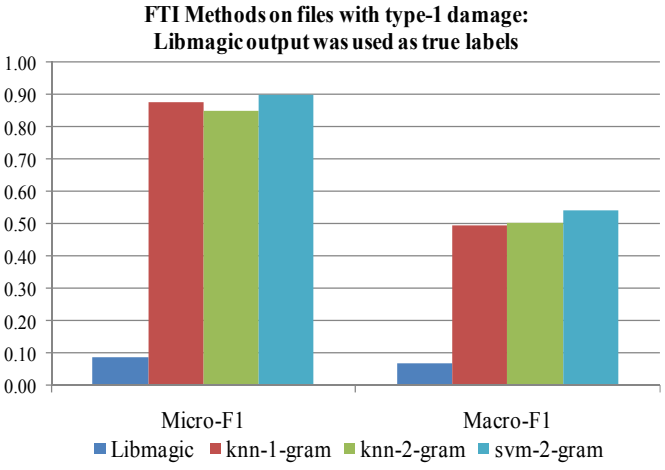

Figure 5: Performance of FTI methods on files with type-1 damage (signature bytes are missing): The output of Libmagic was used as the true labels

measures, SVM and kNN are substantially better than all the COTS solutions being tested. This means that statistical classifiers are more discriminative with respect to user-specified file types in file-name extensions. The larger performance improvements in macro-averaged F_1 by the statistical classifiers over COTS, compared to the smaller improvements in micro-averaged F_1, indicates that COTS predictions tend to agree more with file extensions for common file types, and agree less with file extensions for rare file types.

Figure 3 shows the performance of all the methods on files with type-1 damages, i.e., when the signature bytes of each test instance are missing. Again, file extensions were used as the true labels in this evaluation. Comparing the performance in this graph to that in Figure 2**,** we can see that most COTS solutions failed miserably (with the zero or near-zero value in both micro-averaged and macro-averaged F_1) when the signature bytes are missing, while the statistical classifiers suffer much less. The statistical classifiers are much more robust in FTI with respect to this kind of damage.

Figure 4 compares the results of our statistical classifiers on intact files; the output of Libmagic for each test file was used as the true label of that file. We include the performance of Libmagic for reference, which has the perfect score (F_1=1), of course. We include the results of SVM and kNN with 1-gram and 2-gram features, respectively.

Figure 5 compared the results of these methods on files with type-1 damages, i.e., when the first 512 bytes (including the signature bytes) of each test file is missing. Libmagic failed dramatically in this case, while SVM and kNN are highly robust. SVM using 2-gram features works better than SVM using 1-gram features, but the former is more computationally costly than the latter. We analyze the efficiency and effectiveness trade-off in Section 6. On the other hand, kNN using 1-gram features had better results than kNN using 2-gram features. SVM and kNN have a comparable performance. In general, the statistical learning methods perform better in micro-averaged F_1 (vs micro-averaged F_1) because the common classes have more training instances.

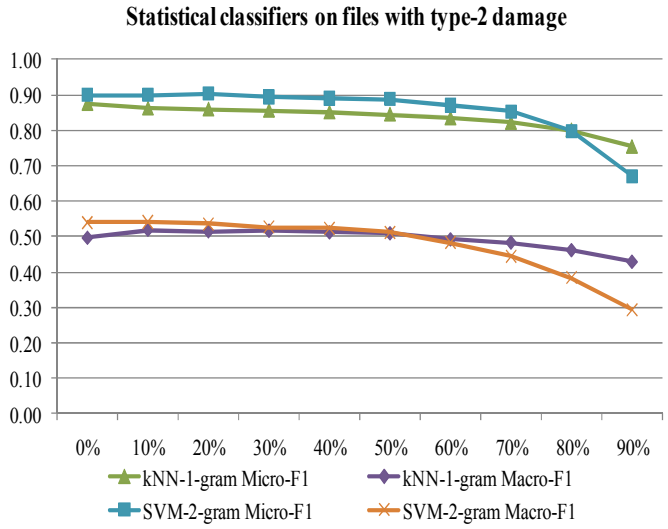

Figure 6: Performance curves of statistical classifiers on files with type-2 damages: The output of Libmagic was used as true labels

SVM (2-gram) outperforms Libmagic by a factor of 10.3 (0.900 vs. 0.088) in micro-averaged F_1 and a factor of 8.0 (0.540 vs. 0.068) in macro-averaged F_1 **.**KNN (1-gram) outperforms Libmagic by a factor of 10.0 (0.874 vs. 0.088) in micro-averaged F_1 and a factor of 7.3 (0.496 vs. 0.068) in macro-averaged F_1**.**

Figure 6 compares the performance curves for SVM (using 2-gram features) and kNN (using 1-gram features) on files with type-2 damages. A certain percentage of each file was removed at random, as well as the first 512 bytes from each file. Again, the two methods have similar curves: until the damaged proportion reaches 50% or higher, there is no significant degradation in classification performance for both methods, but kNN is somewhat more robust when most of the file is missing.

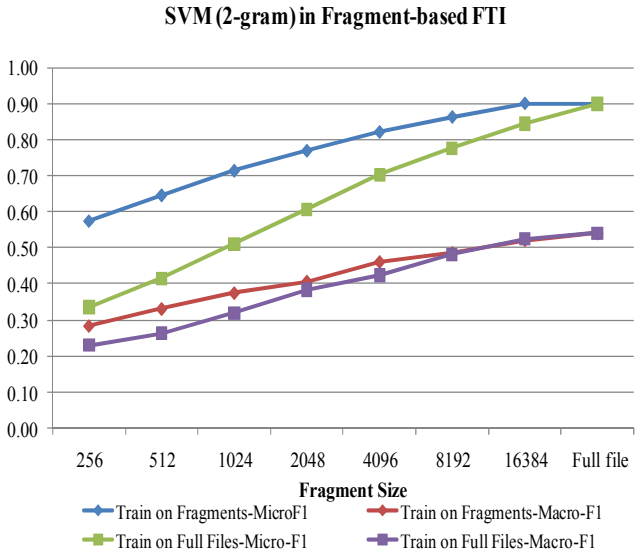

Figure 7: Performance curves of SVM (2-gram) in fragment-based FTI: the output of Libmagic was used as the labels.

Figure 7 compares the performance curves for SVM (using 2-gram features) on segments of files (type-3 damage). We evaluated the methods with two settings: training SVM on segments (of the same size as the test segments), and training SVM on the full files. The former setting yielded a better performance but it had an unrealistic assumption, i.e., the size of the segments in the test set must be known or estimated in advance. The latter setting is more realistic. All the curves show that the smaller the segments, and harder the prediction task.

6. SCALABILITY ANALYSIS

In order for our approach to scale, we need to carefully balance the trade-off between effectiveness (better models) and efficiency (time required to train the models). As the value of '*n*' in the n-gram feature space increases, the individual features capture more information about byte order which may lead to more accurate predictions about file-types; but on the other hand, the time required for computation (in training SVM or in searching kNN given a test instance) also increases. It is therefore crucial to select the value of '*n*' that offers a desirable effectiveness/efficiency trade-off.

We conduct a systematic analysis on how the performance of classifiers changes and how the time in training/testing increases as the value of n increases. Note that the dimensionality of the feature space increases exponentially with *n*; for *n*=1, the potential size of the feature space is 256, for *n*=2 the size is 65,536, for *n*=3 the size is 1,6777,216. For *n*=3, it is impractical to train SVM models with all the features because it will take over several weeks on a single machine. We therefore use statistical feature selection to control the size of the feature space, i.e. we select the most informative features.

In our experiments we used the Information Gain (IG) as the feature selection criterion. IG measures the average information associated with the absence or presence of a feature for file-type identification. Mathematically, the IG for a particular

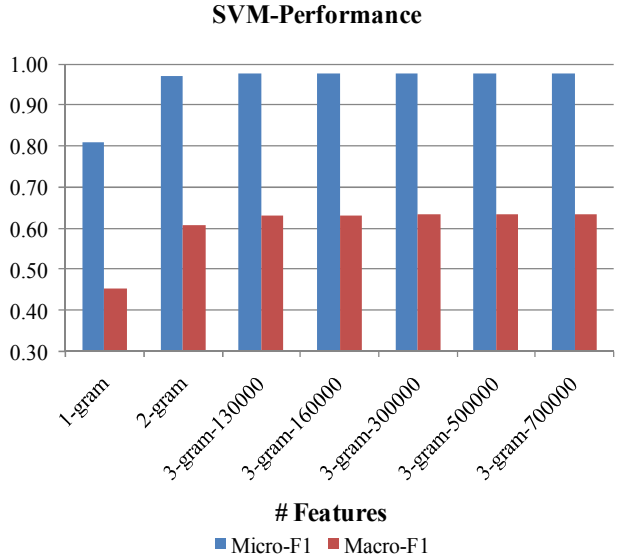

Figure 8: Performance of SVM (on intact files) for different values of *n* and feature selection: The output of Libmagic was used as the true labels.

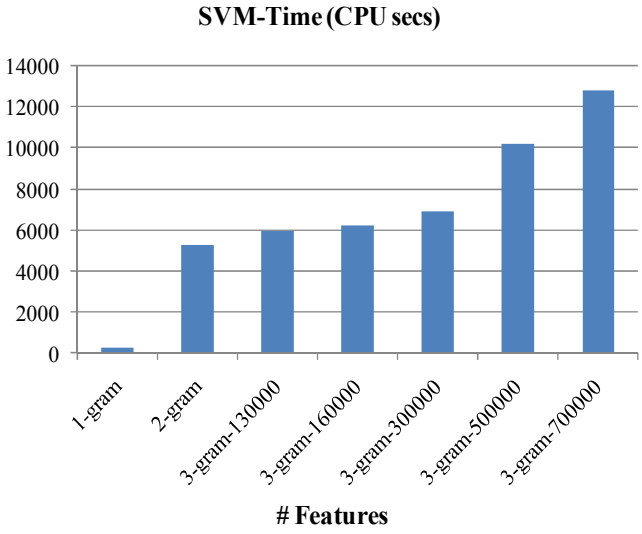

Figure 9: Average training time per fold for SVM (on intact files) for different values of *n* and feature selection: The output of Libmagic was used as the true labels.

feature measures the change in entropy of the file-types given the presence or absence of that particular feature. IG has been commonly used for feature selection in text categorization [29], in decision tree induction [24], dimensionality reduction [11], etc. Given multiple file types (316 in our case), each n-gram feature has multiple IG scores, one per file type. We used the maximum of these multiple IG scores of each feature as the final score of the feature, and we obtained a ranked list of all the features based on their final scores. We measured the F_1 scores as well as the training times for SVM with 1-gram features, 2-gram features, and for SVM 3-gram using the top-*m* features in the IG-ranked list of features. Since kNN does not need any offline training, we just

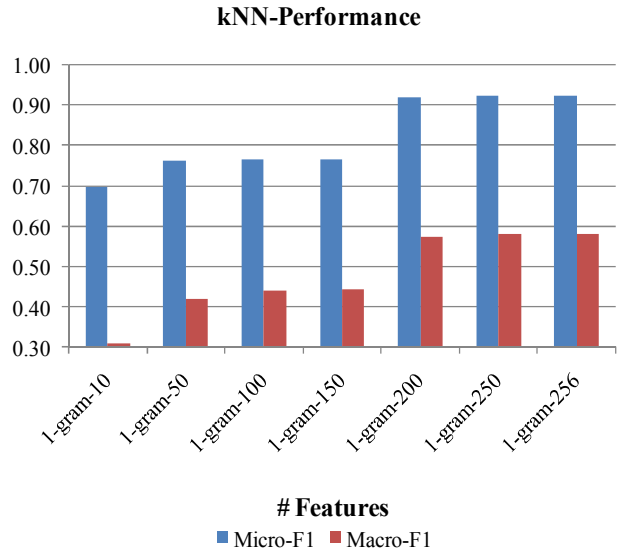

Figure 10: Performance of kNN 1-gram (on intact files) with feature selection: The output of Libmagic was used as the true labels.

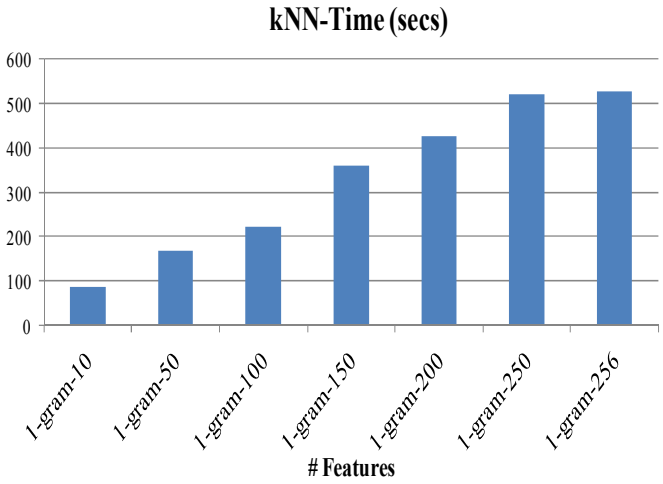

Figure 11: Average Testing time per fold for kNN 1-gram (on intact files) with feature selection: The output of Libmagic was used as the true labels.

measured the F_1 scores and the testing times according to increasing values of m. We only examined feature selection for kNN 1-gram because we found it works better than kNN 2-gram when using all the features without selection. kNN 2-gram required the usage of distributed computing techniques (Hadoop based Map-reduce) to calculate the nearest neighbors, as using a single machine took more than a week. The heavy computational cost is partly due to the relatively large number of non-zero features per file in FTI. For example, when using 2-gram bytes as the features, the average number of non-zero unique features per file is nearly 10000, which is much higher than the typical number (300 or less) of unique words in news-story categorization.

Figure 8 and Figure 9 show the performance and the training times (CPU seconds) for SVM with different feature sets. Figures 10 and Figure 11 show the performance and the test time of kNN using 1-gram features.

Based on the observations on the SVM figures, we see that using 2-gram features (without feature selection) exhibits a desirable balance between effectiveness and efficiency (3-grams with 160K features is another reasonable tradeoff). On the contrary, SVM 1-gram is significantly worse in F_1 measure while SVM 3-gram with additional features only had negligible improvements in F_1 but significantly increased computation time. In 5 fold cross validation across 30524 examples in the RDC dataset, the average training time per fold for SVM 2-gram (using 24203 training examples per fold) without feature selection is 7.3 hours on a single core of Intel Xeon 3.16 Ghz processor. The testing time per fold (6051 test examples per fold) for all the SVM-based methods was about 1second.

Based on the observations on the kNN figures, we see that using the full set of 1-gram features yielded the best F_1 score for kNN. The average computational time per fold is 8.8 CPU minutes on 6051 test examples (and 24203 training example), or 0.09 CPU second per test example. We used single core of Intel Xeon 3.16 Ghz processor for all the nearest neighbor computations. The similarity score calculation between two instances was cached so as to avoid redundant computations.

7. CONCLUSION

We conducted the first thorough comparative analysis of FTI methods on damaged or fragmentary files, contrasting COTS methods and statistical learning ones (SVM and kNN). The study found statistical learning methods to be far more robust than COTS in all the measures. SVM and kNN outperform COTS when the gold standard is set of available file extension for intact files. More importantly, SVM and kNN far outperform COTS on different types of simulated file damages: files with missing signature bytes, files with randomly deleted sections, and isolated file segments. These tests were conducted on a new realistic publicly available data set, encouraging future research and rigorous comparative evaluations. We also thoroughly analyzed the scalability of our statistical classification approaches to FTI. .

REFERENCES

[1] Ahmed, I., Lhee, K., Shin,H. and Hong, M.P 2009. On Improving the Accuracy and Performance of Content-based File Type Identification. In *Proceedings of the 14th Australasian Conference on Information Security and Privacy* (ACISP), 44-59.

[2] Ahmed, I. and Lhee, K. and Shin, H. and Hong, M.P. 2010. Fast file-type identification.In *Proceedings of the 2010 ACM Symposium on Applied Computing,* 1601-1602.

[3] Ahmed, I., Lhee, K., Shin,H. and Hong, M.P. 2010. Fast Content-based File-type Identification. In the *proceedings of Seventh Annual IFIP WG 11.9 International Conference on Digital Forensics*.

[4] Amirani, M.C , Toorani, M., and Beheshti Shirazi, A.A.B. 2008. A New Approach to Content-based File Type

identification. In *Proceedings of the 13th IEEE Symposium on Computers and Communications* (ISCC), 1103-1108.

[5] Belur, V. D. 1991. Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques. McGraw-Hill Computer Science Series. *IEEE Computer Society Press*.

[6] Calhoun, W.C. and Coles, D. 2008. Predicting types of file fragments. *Digital Investigation,* S14-S20.

[7] Cheng, W. and Hüllermeier, E. 2009. Combining instance-based learning and logistic regression for multilabel classification, *Machine Learnig,*211-225.

[8] Darwin, I.F. 2008. Libmagic. ftp://ftp.astron.com/pub/file/

[9] Dreger, H. , Feldmann, A., Mai, M. ,Paxson, V. and Sommer, R. 2006. Dynamic application-layer protocol analysis for network intrusion detection. *USENIX Security Symposium*.

[10] Garfinkel, S., Farrell, P., Roussev, V. and Dinolt, G. 2009. Bringing science to digital forensics with standardized forensic corpora. *DFRWS* , S2-S11.

[11] Guyon, I. and Elisseeff, A. 2003. An introduction to variable and feature selection. *The Journal of Machine Learning Research,* 1157-1182.

[12] Hall A.G and Davis W.P, Sliding Window Measurement for File Type Identification.

[13] Hsieh, C.J., Chang, K.W., Lin, C.J., Keerthi, S.S. and Sundararajan, S. 2008. A dual coordinate descent method for large-scale linear SVM. In the *Proceedings of the 25th International Conference on Machine learning,* 408-415.

[14] Joachims, T. 1998. Text categorization with support vector machines: Learning with many relevant features. *European Conference in Machine Learning,* 137-142.

[15] Karresand, M. and Shahmehri, N. 2006. Oscar—file type identification of binary data in disk clusters and RAM pages. *Security and Privacy in Dynamic Environments*, 413-424.

[16] Karresand, M. and Shahmehri, N. 2006. File type identification of data fragments by their binary structure. In *Proceedings of the IEEE workshop on information assurance*; 140–147.

[17] Li, W.J.,Wang, K. and Stolfo, S.J. and Herzog, B. 2005 Fileprints: identifying filetypes by n-gram analysis. In: *Proceeding of the 2005 IEEE workshop on information assurance*,64-71.

[18] McDaniel, M. 2001. Automatic File Type Detection Algorithm, Masters Thesis, James Madison University.

[19] McDaniel, M. and Heydari, M.H. 2003. Content based file type detection algorithms. In *Proceedings of the 36th Hawaii international conference on system sciences*, Track 9. Washington, D.C.: IEEE Computer Society, 332a.

[20] National Archives of United Kingdom. 2003. DROID - http://droid.sourceforge.net/

[21] Oracle Outside In Technology. http://www.oracle.com/us/products/middleware/content-management/outside-in-tech/index.html

[22] Pontello, M. 2008. TrID – File Identifier. http://mark0.net/soft-trid-e.html

[23] Roussev, V. and Garfinkel, S.L. 2009. File Fragment Classification-The Case for Specialized Approaches. Fourth *International IEEE Workshop on Systematic Approaches to Digital Forensic Engineering (SADFE)*, 3-14.

[24] Mitchell, T. 1997. *Machine Learning*, McGraw Hill.

[25] V. Vapnik, 2005. The nature of statistical learning theory, *Springer verlag*, New York,

[26] Van Rijsbergen, C. 1979. *Information Retrieval.*.Butterworths, London.

[27] Yang, Y. 1994. Expert Network: Effective and Efficient Learning from Human Decisions in Text Categorization and Retrieval. *ACM SIGIR*, pages 13-22.

[28] Yang, Y. 1999. An Evaluation of Statistical Approaches to Text categorization. *Information Retrieval*, 1386-4564.

[29] Yang, Y. and Pedersen, J.O. 1997. A comparative study on feature selection in text categorization, *International Conference in Machine Learning*, 412-420.

[30] Yoo, I.S. and Ultes-Nitsche, U. 2003. Adaptive detection of worms/viruses in firewalls. *In the proceeding of the International conference on Communication, Network, and Information Security.*