Using Machine Learning for Text File Format Identification

Santhilata Kuppili Venkata, Paul Young and Alex Green

December 3, 2020

Using Machine Learning for Text File Format Identification

Santhilata Kuppili Venkata$^{[0000-0003-2406-073X]}$, Paul Young$^{[0000-0002-1102-9664]}$, and Alex Green

The National Archives, Richmond, London, United Kingdom
{santhilata.venkata, paul.young, alex.green}
@nationalarchives.gov.uk

Abstract. File format identification is a necessary step for effective digital preservation of records. It allows appropriate actions for curation and access of file types. While binary files contain header information (metadata) about the file type which can aid identification, text files have none. Methods applied for binary file format identification are ineffective for text files. Most text formats can be opened as plain text files, however file type information is often needed to understand the files full use and context. When huge volumes of files need to be checked, automated methods are necessary for text file format identification. A project was initiated at The National Archives to identify file types from the contents of text files using computational intelligence methods. A machine learning based methodology was tested and implemented using test data. The prototype developed as a proof-of-concept has achieved reasonably good accuracy in successfully detecting five file formats.

Keywords: Text file formats · supervised learning · digital preservation

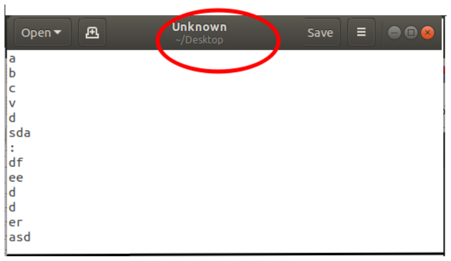
1 Motivation

As the official archive and publisher for the UK Government and England and Wales, The National Archives (TNA) is responsible for collecting and securing the future of the government record. TNA is already receiving born-digital material from government departments and will need to process larger numbers of digital files every year. One of the key steps of processing a new collection of digital records is 'knowing what you have got'[1]. An important factor of this is understanding the file format of each digital record. This allows appropriate preservation actions (e.g. migration, emulation) to be taken in order to ensure that the record is accessible for future researchers. Identifying specific formats of text files present a problem for current processes at TNA. This project was undertaken in order to research sophisticated methods which would allow the identification of formats for text files in an automated fashion.

[1] https://nationalarchives.gov.uk/document/information-management/parsimonious-preservation.pdf

1.1 Why do text files present a problem?

The National Archives develops and maintains the file format registry PRONOM[2]. It contains information for over 1800 different formats. PRONOM information is used to identify formats of digital files in every major digital preservation system via the use of tools such as DROID[3] which utilise the PRONOM information. PRONOM's primary form of identification is by signature patterns based on the structure of the format, determined by observing sample files, magic byte[4] information stored at the header of the format or by observing file format technical specifications. The aim is to provide 'unambiguous' identification of formats through unique pattern sequences. For binary formats this has proved to be very effective, for text formats however there is often no consistent pattern to observe so an 'unambiguous' identification is not possible. Identification via DROID and PRONOM for text files is often achieved by the extension of the format only e.g 'txt', 'csv' or 'py'. The extension of the format is generally an unreliable form of identification, prone to corruption or loss. The same extension can often be used for multiple formats e.g. 'dat'. Contents of text formats are often human-readable. i.e they can be opened as plain text file using a simple text editor. However, if the file extension is missing, incorrect or unambiguous and the format is not known, it is hard to know how the file should be used and accessed. TNA receives such files in the form of supporting files (usually generated by integrated development environments).



```
a
b
c
v
d
sda
:
df
ee
d
d
er
asd
```

Fig. 1. A sample text file with no file extension

A sample file with a missing extension is shown in Fig. 1. The file shown here is human readable and contains some characters written in a column. At a first glance, one would think that this could be a simple exercise in typewriting characters in order. Someone familiar with Unix environments would not rule out the possibility that the file was part of a set of bash script/commands. Opening each file and assessing potential uses is not feasible when we have a large number of such files. More effective automated processes are needed for text file format identification. The file format would provide additional context to use the document appropriately.

1.2 The importance of text formats to the public record

Text formats are becoming increasingly important to the government record. Plain text files '.txt' and CSV files '.csv' can contain important information and datasets. TNA's

[2] https://www.nationalarchives.gov.uk/PRONOM/Default.aspx

[3] https://www.nationalarchives.gov.uk/information-management/manage-information/policy-process/digital-continuity/file-profiling-tool-droid/

[4] Signature bytes at the beginning of binary file types, used by applications to detect how to appropriately parse the file

digital strategy states that records can be held in all sorts of formats including 'structured datasets and computer code'[5]. Programming code is held in text formats. DROID reports of material already held within TNA's digital archive show, based on extensions of the files, programming code and plain text files make up the majority of text based formats that TNA receives regularly.

As an archive of UK Government, TNA is aware that software is created and used in a number of contexts across government, including policy creation, implementation and analysis. It follows that collecting its underlying source code, as well as preserving, and providing future access to that source code is important. Being able to reliably determine the nature of that source code, for example what language it's written in, provides an important piece of context for future researchers. The 2018 UNESCO 'Paris Call' highlights the importance of 'Software Source Code as Heritage for Sustainable Development'. This recognises that preserving software source code and making it widely available is vital to human cultural heritage. It calls on member states to 'recognise software source code as a fundamental research document on a par with scholarly articles and research data'[6].

Aside from digital archiving, file type identification is a serious problem in the areas of digital forensics and cyber security. The research in digital forensics is mainly focused on the identification of image file types and their metadata. While most of the research is targeted for binary file formats, very little work is done in the plain text file category. Being flat files (without header information), text files are difficult to reconstruct if they are corrupted fully or partially. This leads to our research question

How can we correctly identify the file type of a plain text file from its contents?

To answer this, a literature survey to review existing methodologies, approaches formulated, and their adaptability from other fields is explained in section 2. Relevant algorithms reconstructed are explained in section 4. Given the nature of the problem, we narrowed our investigation to the classification category of supervised learning. A Python-based machine learning prototype was developed to understand the intricacies of different classification models during the 'proof of concept' development phase. The model construction, testing and evaluation are in section 5.

2 Literature Review

The Automated file type identification (AFTI) is a highly researched problem in digital preservation, digital forensics and related fields. Binary files are computer-readable but not human-readable. All executable programs are stored in binary files similar numeric data files. In contrast, text files are stored in a form (usually ASCII - the numeric format of alphabets) that is human-readable. AFTI techniques use metadata of a binary file for the identification of it's type. Metadata includes information about file extensions, header/footer signatures [6, 23, 21] and binary information such as magic bytes etc.

[5] https://www.nationalarchives.gov.uk/documents/the-national-archives-digital-strategy-2017-19.pdf

[6] https://unesdoc.unesco.org/ark:/48223/pf0000366715.locale=gb

All these methods work well when the metadata is available and unaltered. However, traditional approaches are not reliable when the integrity of the metadata is not guaranteed. An alternative paradigm is to generate ‘fingerprints’ of file types based on the set of known input files and use them to classify the type of the unknown file. Another prominent approach is to calculate the centroid[7] for a given file type from its salient features. Each unknown file is examined for the distance from the known set of centroids to predict the file type. The centroid paradigm uses supervised and unsupervised learning techniques to infer a file (object) type classifier by exploiting unique inherent patterns that describe a file type’s common file structure. Alamri et al. [3] have published a taxonomy of file type identification ranging over 30 algorithms and approaches. In this section, we review the literature related to predicting file type from fragments and content-based methods using finger print and centroid paradigms.

2.1 File Type Identification from File Fragments

Researchers have concentrated on the identification of image file types with corrupted metadata and missing chunks from the contents. Methods were developed to reconstruct damaged files from their fragments. Identification of file type from fragments is mainly used as a recovery technique. It allows file recovery or rebuilding of the file without contextual information or metadata. This process is also referred to as ‘file carving’ in some of the literature. Image type files are mainly targeted by this technique.

Calhoun et al. [5] investigated two algorithms for predicting the type from fragments in computer forensics. They have performed experiments on the fragments that do not contain header information. The first algorithm was based on the linear discriminant and the second was based on the longest common sub-sequences of fragments. Their work provided various relevant statistics such as byte frequency, entropy, etc. as features to predict the file type. Ahmed et al. [1, 2] also published two techniques to identify the file types from file fragments. These techniques aim to reduce the time consumed to process the contents. Their first technique selects a subset of features describing the frequency of occurrence of certain fragments. The second technique speeds up classification by randomly sampling file blocks. They have performed experiments on .png, .jpg and .tiff file types. Poisel et al. [19, 18] published a comprehensive survey of file carving research to detect the file types from their fragments. They have also provided a file carving ontology useful for researchers. In a similar work, Evensen et al. [8] explored the use of the naive Bayes classifier combined with n-gram analysis of byte sequences in files to correctly identify the file type. Gopal et al. [10] presented the evaluation and analysis of the robustness of Support Vector Machine (SVM) and k-Nearest Neighbours (kNN) in handling damaged files and file segments. They have restricted their study to the file type identification from metadata. Their evaluation reveals that SVM and kNN methods learn better than any commercial off-the-shelf tools that have been developed based on file extensions. In his thesis, Wilgenbus [24] presented a combined multi-layer perceptron neural network and linear programming discriminant classifiers for the multiple class file fragment type identification problems.

[7] A centroid is the mean position of all the points in all of the coordinate directions in a multi-dimensional space.

This solution could help our text file format identification problem, as neural networks learn from features of the contents and help in classification of discrete file types. In their work, Karampidis et al. [11, 12] examine a three-stage methodology for AFTI, using feature selection (Byte Frequency Distribution) and genetic algorithm. They have tested with classification models including decision tree, SVM, neural networks, logistic regression and kNN. Their methodology showed that artificial neural networks performed with exceptional accuracy in most cases.

2.2 Content-based File Type Identification

Content-based file type detection methods have proved to be more robust and accurate so far. They are built on the principle of extracting features from the files. Initial work on content-based file type identification [16, 15] was based on three algorithms: byte frequency analysis, byte frequency cross-correlation and File header/trailer analysis. Li et al.[14] have provided improvements to these algorithms by generating file prints (file signatures) using the K-means algorithm with Manhattan distance metric. They produced file prints with the help of the statistical features extracted and selected. The file prints are also called as *'centroid'* in literature. An unknown file is tested against a set of known centroids. The distance between the centroids is compared to predict the possible file type. The Mahalanobis distance metric is deployed for the comparison. The file prints (centroids) are developed using Natural Language Processing (NLP) techniques such as pattern matching of n-gram contiguous sequence models. While their work is pioneering for its kind, their approach restricts the input file to follow a specific style only. They also fail to differentiate files when the target file types have almost similar structures, for example, Java and C programming source codes. We need to generate file features and classification models in such a way that they describe file types distinctly.

Other improvements in this area include neural networks [7] and Byte Frequency Distribution (BFD) to classify file types [13, 25]. Amirani et al. [4] proposed a content-based file type detection method for files normalised using BFD. Their model uses principal component analysis for feature selection. The model is then fed into an auto-associative unsupervised neural network. Mitlohner et al. [17] published a comprehensive study of characteristics of open data CSV files. Their work analyzes an open data corpus containing resources from a data consumer perspective. This study provided a deep insight to feature engineering CSV file type.

Predicting the file type from the contents of text files complicates the problem of AFTI. Though several approaches are available, they are highly domain-specific. Hence we could not use them for the identification of all file types from their contents. We need to research generic methods to fill this gap based on existing approaches.

3 Methodology

From the existing literature, there are mainly two approaches to work with file type identification that can be adopted to text files. The first approach is to treat the text file as a plain text file (no prior knowledge about the file type) and search for specific characteristics for possible file types. The signature of the file type is a combination of characteristics of that type. This is a generic method and can be extended to any number of file types. However, this approach needs a thorough knowledge of each file type to generate its characteristic features. The second approach is based on prior knowledge about a file. For example, if we predict a file belongs to a programming language, we could validate the file type by running its compiler(s), or searching for specific text patterns corresponding to the programming language. Though the second approach is implementable, it is not scalable with the increase in the number of file types. TNA deals with a huge variety of file types for digital preservation. A flexible methodology reflecting the first approach suits well for this situation. The methodology should implement an iterative process model to include file features gradually as more file types are included. As and when a new file type is to be included, its features (specific characteristics) should be compared against the existing features of other existing file types and engineered to add to the list. The flow graph in Fig. 2 depicts the pipeline of activities.

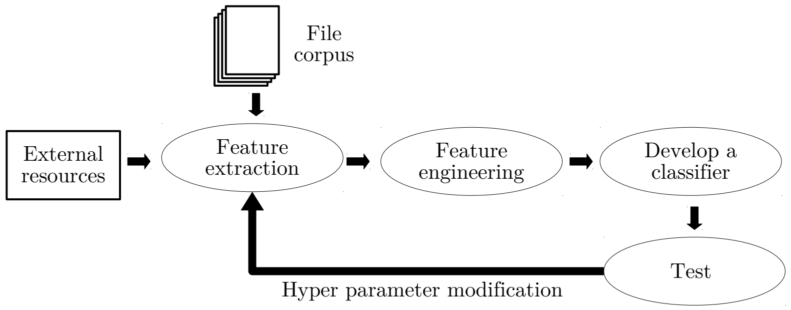


Fig. 2. Methodology to include file types progressively

a) *The File Corpus* is the set of files that serve as the dataset to the identification task.
b) *External resources* comprise of various external tools used for data cleaning and pre-processing. For example, our internal tool 'DROID' is used to eliminate known file types as a first step in case, the file types are specified.
c) *Feature Extraction* is the process of extracting Characteristic features that determine the style and nature of the file type.
d) *Feature Engineering* is the process of using domain knowledge of the data to create features that make machine learning algorithms work. It helps to fine tune the machine learning models by reducing the computational processing overhead.
e) *Classifier Development and Test* Machine learning (ML) is chosen to develop a classifier. ML algorithms are used to understand and extract the patterns from the data and help to predict the outcome.

4 Data Pre-processing

As a representative collection of the type of text format material that TNA would receive, this study cloned files from publicly available Github repositories of the Government Digital Service[8] (GDS) and TNA[9]. In all, we cloned 1457 public repositories from these two sources. They contain over 410,000 files representing 928 file types that can be opened with a simple text editor program as a testbed. However, it is an extensive task to develop a single classifier model that classifies all 928 file types. So we have grouped files into 14 categories to understand the priority file types to start our experimentation. With the help of DROID reports, five file types are shortlisted including programming source files such as Python, Java and data files types: .txt, .tsv and .csv.

4.1 Feature Extraction

The data consists of unstructured text files. So the first phase was to recognise features that describe Python and Java source codes and .txt, .csv and .tsv files correctly. We have identified a total of 45 features that are suitable across five file types. Automated scripts are developed to make the feature extraction uniform across files.

4.2 Feature Engineering

Unlike the common machine learning problems, the text file format identification presents a non-linear learning problem. By non-linear learning we mean, the file features do not represent a direct correlation between different file types. For example, a very high correlation between the Java and Python programming file structures make it difficult for file type classification task. An approach using regression analysis might not find a difference between these two types. Similarly, .csv and .tsv files share some of their file features. Many times a comma separated file (.csv) may contain unformatted textual lines, leaving very little to differentiate between .txt and .csv file formats. Hence feature engineering should be a combined effort for a domain expert and machine learning researcher. For example,

- a Python source code file differs from a Java file by its commenting style, strict indentation requirement at the beginning of each line of the code, the use of specific keywords etc. Whereas, the Java source code needs to follow a pre-defined structure to be able to compile successfully (such as, every line in the Java needs to end with a ';' (semi-colon), Python does not need any specific line encoding).
- even though .csv and .tsv files are largely categorised as text-based, they can be recognised by their use of the number of commas (or other delimiters). A comparison of the delimiters could become a deciding factor in file identification.
- in general, a .txt file has no rules for its layout compared to .csv or .py. It is difficult to extract a pattern from a normal .txt file. Hence the count of common words in normal English can be a good characteristic of text files[10].
- another significant characteristic is the 'word-combination' proximity. For example, the combinations of words such as $<$ def-return $>$, $<$ if-then-else $>$ etc. are likely to appear in closer proximity in the programming codes than in a .txt file. So, we derived a threshold for the word-combination word sets.

[8] https://github.com/alphagov

[9] https://github.com/nationalarchives

[10] We assume the usage of common words is more in normal text files than in programming or data files

After feature engineering, 33 features were selected for classification. Features extracted and used for classification are listed in the Appendix.

5 Classification Models & Evaluation

There are four prominent types of classification algorithms. They are (i) Linear models, (ii) Tree-based algorithms, (iii) k-nearest algorithms and (iv) Neural network based. Since our problem has discrete outputs and non-linear inputs, the linear models are omitted. All models were trained and hyperparameters were tuned to improve the accuracy over many iterations[11].

A **Decision tree** [22] is a flowchart-like tree structure where an internal node represents a feature. The branch represents a decision rule, and each leaf node represents the outcome. Each parent node learns to partition the data based on the attribute value. It partitions the tree recursively until all the data in the partition belongs to a single class.

The **k-Nearest Neighbour** classifier [22] (kNN) is based on feature similarity that determines how we classify a given data point. The output is a class membership (predicts a class — a discrete value). An object is classified by a majority vote of its neighbours, with the object being assigned to the class most common among its k-nearest neighbours.

A **Multilayer perceptron** (MLP) is a deep, artificial neural network, composed of more than one perceptron [20, 9]. MLPs train on a set of input-output pairs and learn to model the correlation between those inputs and outputs. Training involves adjusting the parameters, or the weights and biases, of the model, in order to minimize error.

The MLP model designed for our classification is a 3-layer fully connected neural network shown in Fig. 3 with 33 nodes in the input layer, 12 nodes each in the hidden layers and 5 nodes (one for each of the output classes) in the output layer. The number of nodes in each of the layers was decided on a trial and error basis. The parameters set for the MLP is given in Table 1.

The evaluation of above models is in Table 2. The train-to-test ratio is set ideally as 80:20 to achieve better accuracy. Though the accuracy of classification is very high, we consider the precision metric more significant, given the non-uniform distribution of file types in the file corpus. For the kNN classification, the ‘minkowski’ distance metric[12] is used to establish the distance between classes. The value for ‘K’ is set to 3.

6 Conclusion & Scope

TNA initiated the project, ‘Text File Format Identification’ to identify file formats of text files. The prototype developed has achieved good accuracy and precision, proving that this approach can be successful for identifying text file formats. Python and Java programming code file types were classified with higher accuracy compared to .tsv and .csv files. This is probably due to the inherent structure of programming files, which

[11] The Jupyter notebooks developed as a proof of concept are available here-https://github.com/nationalarchives/Text-File-Format-Identification

[12] https://www.sciencedirect.com/topics/computer-science/minkowski-distance

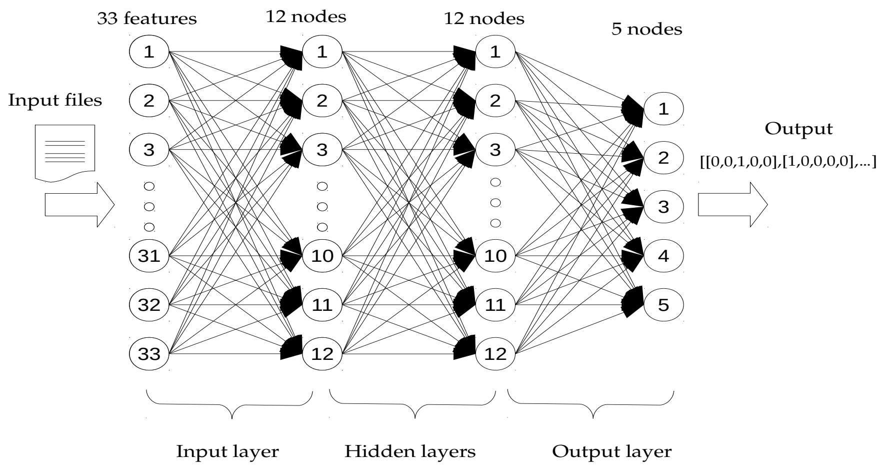


Fig. 3. Multilayer perceptron neural network used for classification

Table 1. MLP parameters

Parameter	Value
Activation (input to hidden layer)	relu
Activation (hidden-hidden layers)	relu
Activation (hidden to output)	softmax
optimizer	adam
loss type	categorical crossentropy
metrics	accuracy
# epochs	30
batch_size	20

Table 2. Performance of classification models

Classification model	Accuracy	Precision
Decision tree	92.58%	86%
kNN	83.4%	80%
MLP	90.28%	88%

is able to be captured better than .csv/.tsv. The decision tree classifier performed better than the other two models. In future we would like to focus on revising the dominant feature identification for .csv and .tsv file types. Also, it was assumed that each .csv file contained only one table. However, it is possible that multiple tables exist within a single .csv file. This issue could be investigated. Even though the current prototype works well for the five file types, a revision of feature engineering will be necessary whenever a new file type is included.

References

1. Ahmed, I., suk Lhee, K., Shin, H., Hong, M.: Content-based file-type identification using cosine similarity and a divide-and-conquer approach. IETE Technical Review **27**(6), 465 (2010). https://doi.org/10.4103/0256-4602.67149

2. Ahmed, I., Lhee, K.S., Shin, H.J., Hong, M.P.: Fast content-based file type identification. In: Advances in Digital Forensics VII, p. 65–75. Springer Berlin Heidelberg (2011). https://doi.org/10.1007/978-3-642-24212-0_5
3. Alamri, N.S., Allen, W.H.: A taxonomy of file-type identification techniques. In: Proceedings of the 2014 ACM Southeast Regional Conference. p. 49:1–49:4. ACM SE '14, ACM, New York, NY, USA (2014). https://doi.org/10.1145/2638404.2638524
4. Amirani, M.C., Toorani, M., Mihandoost, S.: Feature-based type identification of file fragments. Security and Communication Networks **6**(1), 115–128 (apr 2012). https://doi.org/10.1002/sec.553
5. Calhoun, W.C., Coles, D.: Predicting the types of file fragments. Digit. Investig. **5**, S14–S20 (Sep 2008). https://doi.org/10.1016/j.diin.2008.05.005
6. DROID: https://www.nationalarchives.gov.uk/information-management/manage-information/preserving-digital-records/droid/ (2013)
7. Dunham, J.G., Tseng, J.C.R.: Classifying file type of stream ciphers in depth using neural networks. In: The 3rd ACS/IEEE International Conference on Computer Systems and Applications, 2005. p. 97– (Jan 2005). https://doi.org/10.1109/AICCSA.2005.1387088
8. Evensen, J.D., Lindahl, S., Goodwin, M.: File-type detection using naive bayes and n-gram analysis. In: 2014: NISK 2014 (2014)
9. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. The MIT Press (2016)
10. Gopal, S., Yang, Y., Salomatin, K., Carbonell, J.: Statistical learning for file-type identification. In: 2011 10th International Conference on Machine Learning and Applications and Workshops. IEEE (dec 2011). https://doi.org/10.1109/icmla.2011.135
11. Karampidis, K., Kavallieratou, E., Papadourakis, G.: Comparison of classification algorithms for file type detection a digital forensics perspective. Polibits **56**, 15–20 (2017)
12. Karampidis, K., Papadourakis, G.: File type identification - computational intelligence for digital forensics. The Journal of Digital Forensics, Security and Law (2017). https://doi.org/10.15394/jdfsl.2017.1472
13. Karresand, M., Shahmehri, N.: File type identification of data fragments by their binary structure. In: 2006 IEEE Information Assurance Workshop. p. 140–147 (June 2006). https://doi.org/10.1109/IAW.2006.1652088
14. Li, W.J., Stolfo, S.J., Herzog, B.: Fileprints: identifying file types by n-gram analysis. In: Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop. p. 64–71 (June 2005). https://doi.org/10.1109/IAW.2005.1495935
15. McDaniel, M.: Automatic File Type Detection Algorithm. Master's thesis (2001)
16. McDaniel, M., Heydari, M.: Content based file type detection algorithms. In: 36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of the. IEEE (2003). https://doi.org/10.1109/hicss.2003.1174905
17. Mitlöhner, J., Neumaier, S., Umbrich, J., Polleres, A.: Characteristics of open data csv files. In: 2016 2nd International Conference on Open and Big Data (OBD). p. 72–79 (Aug 2016). https://doi.org/10.1109/OBD.2016.18
18. Poisel, R., Rybnicek, M., Tjoa, S.: Taxonomy of data fragment classification techniques. In: Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, p. 67–85. Springer International Publishing (2014). https://doi.org/10.1007/978-3-319-14289-0_6
19. Poisel, R., Tjoa, S.: A comprehensive literature review of file carving. In: 2013 International Conference on Availability, Reliability and Security. IEEE (sep 2013). https://doi.org/10.1109/ares.2013.62
20. Ramchoun, H., Idrissi, M.A.J., Ghanou, Y., Ettaouil, M.: Multilayer perceptron: Architecture optimization and training with mixed activation functions. In: Proceedings of the 2Nd International Conference on Big Data, Cloud and Applications. p. 71:1–71:6. BDCA'17, ACM, New York, NY, USA (2017)

21. Siegfried: https://www.itforarchivists.com/siegfried/
22. Tan, P.N., Steinbach, M., Kumar, V.: Introduction to Data Mining. Addison Wesley, us ed edn. (May 2005)
23. TrID: http://mark0.net/soft-trid-e.html
24. Wilgenbus, E.F.: The file fragment classification problem : a combined neural network and linear programming discriminant model approach. Master's thesis, N (2013), http://hdl.handle.net/10394/10215
25. Zhang, L., White, G.B.: An approach to detect executable content for anomaly based network intrusion detection. In: 2007 IEEE International Parallel and Distributed Processing Symposium. p. 1–8 (March 2007). https://doi.org/10.1109/IPDPS.2007.370614

A Features generated from files

Table 3: Features extracted and Used

Feature	**Description**
file_name	Name of the file along with its complete path
file_extension	File extension if available
num_lines	Number of lines in the file separated by newline character
header_info	File header information if available
trailer_info	Trailer information, if available
indentation	Number of spaces used for indentation (specific to Python)
eol_marker	End-of-line markers, if any (specific to Java)
sol_marker	Start-of-line markers, if any
isLowercaseMethods	Whether methods/functions start with lower case alphabets
num_stopwords	Number of stop words used (specific to text files)
num_Python_keywords	Number of Python key words within the file
num_Java_keywords	Number of Java key words used in the file
Python_comments	Number of Python style of comments
Java_comments	Number of Java style of comments
angular_brackets	Number of angular brackets used
curly_brackets	Number of curly brackets used
round_brackets	Number of round brackets used
square_brackets	Number of square brackets used
num_def	Number of 'def' used (specific to Python
num_returns	Number of times the key word 'return' used
if_else_proximity	Number of words between if and else (specific to programming codes)
num_carat	Number of times the carat symbol used (specific to csv and tsv)
num_comma	Number of times the comma symbol used (specific to csv and tsv)
num_fullstop	Number of times the fullstop symbol used (specific to csv and tsv)
num_tab	Number of times the tab used (specific to csv and tsv)
num_semicolon	Number of times the semi colon symbol used (specific to csv and tsv)
num_colon	Number of times the colon symbol used (specific to csv and tsv)
num_pipe	Number of times the pipe symbol used (specific to csv and tsv)
num_hash	Number of times the hash symbol used (specific to csv and tsv)

Table 3: Features extracted and Used

Feature	**Description**
average_line_length	Average length of a line (in characters)
description	File description in short, if available
programming	Whether the file is a programming code, if known
stopwords_normalised	Normalised stop words across Java and Python
file_type	File Type information