Text File Format Identification: An Application of AI for the Curation of Digital Records

Santhilata Kuppili Venkata
The National Archives
United Kingdom
email: santhilata.venkata[a]
0000-0003-2406-073X

Paul Young
The National Archives
United Kingdom
email: paul.young[a]
0000-0002-1102-9664

Alex Green
The National Archives
United Kingdom
email: alex.green[a]
0000-0003-2463-3649

[a]@nationalarchives.gov.uk

Abstract – File format identification is a necessary step for the effective digital preservation of records. It allows appropriate actions to be taken for the curation and access of file types. The National Archives (TNA) has existing processes for dealing with binary file formats, using tools such as PRONOM and DROID. They rely on using header information (metadata) and consistent binary sequences. However, these methods cannot be applied to text file formats as text files do not contain recognisable header information. Even though many text file formats can be opened using a plain text editor, the file type information is often needed to understand the context of these files especially when the file is partially corrupted. We need automated intelligent methods to extract patterns and information from the content of the files and detect the file type. The representative data collected from the GitHub repositories of UK Government departments suggest that there are predominantly source code files (such as python and Java) and data files(such as .csv, .tsv and .txt). The first prototype using AI methodologies has achieved reasonably good performance in successfully detecting five file formats. Current results encourage us to carry out additional experiments to include further text file formats.

Keywords – Text file formats, supervised learning, digital preservation.

Conference Topics – Scanning the New Development.

I. 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 presents 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.

A. 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 file extension 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 files using a simple text editor. However, if the file extension is missing, incorrect or un-

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

[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

DOI: 10.1145/nnnnnnn.nnnnnnn

ambiguous 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. More effective automated processes are needed for text file format identification. The file format would provide additional context to use the document appropriately.

B. *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 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, that programming code and plain text files make up the majority of text based formats received by TNA.

As the 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 is written in, provides important 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. Research in digital forensics is mainly focused on the identification of image file types and their metadata. While most of the research targets binary file formats, very little focuses on plain text files. 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, we have tested an iterative machine learning based approach starting with the five file types most predominant in TNA's collections. Unlike a rule-based approach, the machine learning approach is flexible and can include more and more file types over future iterations of a suitable model's development.

This paper is structured as follows: a literature survey to review existing methodologies, approaches formulated, and their adaptability from other fields is explained in section II. Relevant algorithms reconstructed are explained in section IV. 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 V.

II. 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 as binary files similar to 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 the metadata of a binary file for the identification of its type. The metadata includes information about file extensions, header and footer signatures [1]–[3], and binary information such as magic bytes etc.

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. [4] 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.

A. *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 frag-

[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

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

ments that do not contain header information. The first algorithm was based on the linear discriminant and the second was based on the longest common subsequences of fragments. Their work provided various relevant statistics such as byte frequency, entropy, etc. as features to predict the file type. Ahmed et al. [6], [7] also published two techniques to identify the file types from file fragments. These techniques aim to reduce the time spent in processing 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. [8], [9] published a comprehensive survey of file carving research to detect the file types from their fragments. They have also provided a useful file carving ontology. In a similar work, Evensen et al. [10] 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. [11] 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 [12] presented a combined multi-layer perceptron neural network and linear programming discriminant classifiers for the multiple class file fragment type identification problems. 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. [13], [14] examine a three-stage methodology for AFTI, using feature selection (Byte Frequency Distribution) and genetic algorithm. They have tested this 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.

B. *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 [15], [16] was based on three algorithms: byte frequency analysis, byte frequency cross-correlation and File header/trailer analysis. Li et al.[17] 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 [18] and Byte Frequency Distribution (BFD) to classify file types [19], [20]. Amirani et al. [21] 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. [22] 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 the 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.

III. Methodology

From the existing literature, there are mainly two approaches to work with file type identification that can be adopted for 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 the 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 can be implemented, it is not scalable given the volume of file types. TNA deals with a huge variety of file types for digital preservation. A flexible methodology using the first approach suits this situation well. 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. 1 depicts the pipeline of activities.

a) *The File Corpus* is the set of files that serve as the dataset for the identification task.

b) *External resources* comprise various external tools used for data cleaning and pre-processing. For example, TNA's tool 'DROID' is used to eliminate known file types as a first step in case the file types can be identified.

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. The carefully selected features help to establish rules (and thus knowledge generation) for file type classification using rule-based models. For machine learning models, features can supplement the information gain.

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.

IV. 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 test bed. However, it is an huge 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 and Java with three data file types: .txt, .tsv and .csv.

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

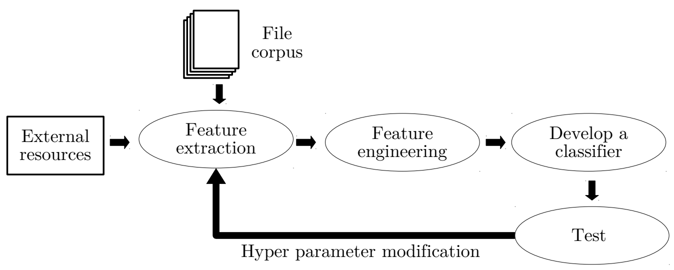

Figure 1: Methodology to include file types progressively

A. 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.

B. 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 the file type classification task to differentiate the two formats. 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. Often 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 follows a pre-defined structure to be able to compile successfully (such as, every line must 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 sets.

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

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

Table 1: Performance of classification models

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

V. 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 algorithms. Since our problem has discrete outputs and non-linear inputs, only approaches which required explicit feature engineering were considered, omitting linear models. All models were trained and hyperparameters were tuned to improve the accuracy over many iterations[11].

A **Decision tree** [23] 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 [23] (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 [24], [25]. 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 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 by trial and error. The parameters set for the MLP are as follows: activation function : *relu*, no.of epochs(iterations): 30 and batch-size set to 20. We chose categorical cross_entropy[12] and accuracy[13] as the parameters for loss and performance metrics.

The evaluation of above models is in Table 1. 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[14] is used to establish the distance between classes. The value for 'K' is set to 3.

VI. Discussion & Scope

The National Archives has initiated the project, 'Text File Format Identification' to identify file formats of corrupted text files during the curation of digital documents. We have chosen an iterative machine learning approach to develop a prototype with the flexibility to include more file types over the next iterations of the its development. Hence we did not experiment with rule-based approaches which would limit the nature of the file types to be included in our next iterations.

Our methodology included extraction of features and feature engineering to build suitable machine learning models from the raw text files corpus. The feature engineering make the models scalable. We have also presented the suitability of deep learning models for classification. These can provide better results when dealing with large numbers of file types with almost similar features, which will be included in the future. This methodology is used for text based files as they do not have signature finger prints as binary file types.

The prototype achieved good accuracy and precision, proving that this approach can be successful for identifying five of the predominant text file formats. However, satisfactory accuracy and precision depend upon the dataset at hand. 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 is able to be defined better than those of .csv and .tsv files. 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. We would like to work on the neural network approach, for the classification of file types with similar features (such as .csv and .tsv) with large volumes of training data.

As of now, 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 also could be investigated in future. 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. In this experiment, we have not considered the 'average length of a line' and 'number of lines of the text files' as special features to train models as they do not provide enough support for classification. However, we would like to add them to see how neural network models utilise such information to learn how to

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

[12]Categorical cross_entropy describes a loss function to improve the performance of a neural network model

[13]Accuracy is a performance measure to show the goodness of a model

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

better identify almost-similar file types.

References

[1] DROID, https://www.nationalarchives.gov.uk/information-management/manage-information/preserving-digital-records/droid/, 2013.

[2] TrID, http://mark0.net/soft-trid-e.html.

[3] Siegfried, https://www.itforarchivists.com/siegfried/.

[4] N. S. Alamri and W. H. Allen, "A taxonomy of file-type identification techniques," in *Proceedings of the 2014 ACM Southeast Regional Conference*, ser. ACM SE '14, Kennesaw, Georgia: ACM, 2014, 49:1–49:4, isbn: 978-1-4503-2923-1. doi: `10.1145/2638404.2638524`.

[5] W. C. Calhoun and D. Coles, "Predicting the types of file fragments," *Digit. Investig.*, vol. 5, S14–S20, Sep. 2008. doi: `10.1016/j.diin.2008.05.005`.

[6] I. Ahmed, K. suk Lhee, H. Shin, and M. Hong, "Content-based file-type identification using cosine similarity and a divide-and-conquer approach," *IETE Technical Review*, vol. 27, no. 6, p. 465, 2010. doi: `10.4103/0256-4602.67149`.

[7] I. Ahmed, K.-S. Lhee, H.-J. Shin, and M.-P. Hong, "Fast content-based file type identification," in *Advances in Digital Forensics VII*, Springer Berlin Heidelberg, 2011, 65–75. doi: `10.1007/978-3-642-24212-0\_5`.

[8] R. Poisel and S. Tjoa, "A comprehensive literature review of file carving," in *2013 International Conference on Availability, Reliability and Security*, IEEE, 2013. doi: `10.1109/ares.2013.62`.

[9] R. Poisel, M. Rybnicek, and S. Tjoa, "Taxonomy of data fragment classification techniques," in *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, Springer International Publishing, 2014, 67–85. doi: `10.1007/978-3-319-14289-0\_6`.

[10] J. D. Evensen, S. Lindahl, and M. Goodwin, "File-type detection using naive bayes and n-gram analysis," in *2014: NISK 2014*, 2014.

[11] S. Gopal, Y. Yang, K. Salomatin, and J. Carbonell, "Statistical learning for file-type identification," in *2011 10th Intl conf on Machine Learning and Applications and Workshops*, IEEE, 2011. doi: `10.1109/icmla.2011.135`.

[12] E. F. Wilgenbus, "The file fragment classification problem : A combined neural network and linear programming discriminant model approach," M.S. thesis, N, 2013. [Online]. Available: `http://hdl.handle.net/10394/10215`.

[13] K. Karampidis, E. Kavallieratou, and G. Papadourakis, "Comparison of classification algorithms for file type detection a digital forensics perspective," *Polibits*, vol. 56, 15–20, 2017.

[14] K. Karampidis and G. Papadourakis, "File type identification - computational intelligence for digital forensics," *The Journal of Digital Forensics, Security and Law*, 2017. doi: `10.15394/jdfsl.2017.1472`.

[15] M. McDaniel and M. Heydari, "Content based file type detection algorithms," in *36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of the*, IEEE, 2003. doi: `10.1109/hicss.2003.1174905`.

[16] M. McDaniel, "Automatic file type detection algorithm," M.S. thesis, James Madison University, 2001.

[17] W. J. Li, S. J. Stolfo, and B. Herzog, "Fileprints: Identifying file types by n-gram analysis," in *Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop*, 2005, 64–71. doi: `10.1109/IAW.2005.1495935`.

[18] J. G. Dunham and J. C. R. Tseng, "Classifying file type of stream ciphers in depth using neural networks," in *The 3rd ACS/IEEE International Conference on CSA, 2005.*, 2005, 97–. doi: `10.1109/AICCSA.2005.1387088`.

[19] M. Karresand and N. Shahmehri, "File type identification of data fragments by their binary structure," in *2006 IEEE Information Assurance Workshop*, 2006, 140–147. doi: `10.1109/IAW.2006.1652088`.

[20] L. Zhang and G. B. White, "An approach to detect executable content for anomaly based network intrusion detection," in *2007 IEEE International Parallel and Distributed Processing Symposium*, 2007, 1–8. doi: `10.1109/IPDPS.2007.370614`.

[21] M. C. Amirani, M. Toorani, and S. Mihandoost, "Feature-based type identification of file fragments," *Security and Communication Networks*, vol. 6, no. 1, 115–128, 2012. doi: `10.1002/sec.553`.

[22] J. Mitlöhner, S. Neumaier, J. Umbrich, and A. Polleres, "Characteristics of open data csv files," in *2016 2nd International Conference on Open and Big Data (OBD)*, 2016, 72–79. doi: `10.1109/OBD.2016.18`.

[23] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*, US ed. Addison Wesley, May 2005, isbn: 0321321367.

[24] H. Ramchoun, M. A. J. Idrissi, Y. Ghanou, and M. Ettaouil, "Multilayer perceptron: Architecture optimization and training with mixed activation functions," in *Proceedings of the 2Nd International Conference on BDCA*, ser. BDCA'17, Tetouan, Morocco: ACM, 2017, 71:1–71:6, isbn: 978-1-4503-4852-2.

[25] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. The MIT Press, 2016, isbn: 0262035618, 9780262035613.

A. Features generated from files

Table 2: Features extracted and Used

Feature	Description
file name	Name of the file along with its complete path
file extension	File extension if available
numlines	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
isLowercase Methods	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)
averageline 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